Representation and Recognition of Action in Interactive Spaces

Claudio Santos Pinhanez

Bachelor on Mathematics, University of São Paulo, 1985
Master on Applied Mathematics, University of São Paulo, 1989

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning, in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Media Arts and Sciences at the Massachusetts Institute of Technology

June 1999

© Massachusetts Institute of Technology, 1999. All rights reserved.

Author _____

Program in Media Arts and Sciences

Certified by _____

Aaron F. Bobick
Associate Professor of Computational Vision, MIT Media Laboratory

Accepted by _____

Stephen A. Benton
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Representation and Recognition of Action in Interactive Spaces

Claudio Santos Pinhanez

Submitted to the Program in Media Arts and Sciences, School of Architecture
and Planning on April 30th, 1999, in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Media Arts and Sciences
at the Massachusetts Institute of Technology

Abstract

This thesis presents new theory and technology for the representation and recognition of complex, context-sensitive human actions in interactive spaces. To represent action and interaction a symbolic framework has been developed based on Roger Schank's *conceptualizations*, augmented by a mechanism to represent the temporal structure of the sub-actions based on Allen's *interval algebra* networks. To overcome the exponential nature of temporal constraint propagation in such networks, we have developed the *PNF propagation* algorithm based on the projection of IA-networks into simplified, 3-valued (*past*, *now*, *future*) constraint networks called *PNF-networks*.

The PNF propagation algorithm has been applied to an action recognition vision system that handles actions composed of multiple, parallel threads of sub-actions, in situations that can not be efficiently dealt by the commonly used temporal representation schemes such as finite-state machines and HMMs. The PNF propagation algorithm is also the basis of *interval scripts*, a scripting paradigm for interactive systems that represents interaction as a set of temporal constraints between the individual components of the interaction. Unlike previously proposed non-procedural scripting methods, we use a strong temporal representation (allowing, for example, mutually exclusive actions) and perform control by propagating the temporal constraints in real-time.

These concepts have been tested in the context of four projects involving story-driven interactive spaces. The action representation framework has been used in the *Intelligent Studio* project to enhance the control of automatic cameras in a TV studio. Interval scripts have been extensively employed in the development of *"SingSong"*, a short interactive performance that introduced the idea of live interaction with computer graphics characters; in *"It / I"*, a full-length computer theater play; and in *"It"*, an interactive art installation based on the play *"It / I"* that realizes our concept of *immersive stages*, that is, interactive spaces that can be used both by performers and public.

Thesis Advisor

Aaron F. Bobick
Associate Professor of Computational Vision, MIT Media Laboratory

Thesis Committee

Thesis Reader_____

Rodney A. Brooks
Fujitsu Professor of Computer Science and Engineering, MIT
Director, MIT Artificial Intelligence Laboratory

Thesis Reader_____

Joseph Bates
Professor (Research), Computer Science Department, Carnegie Mellon University
President, Zoesis, Inc.

Thesis Reader_____

Ken Perlin
Associate Professor, Dept. of Computer Science, New York University
Director, NYU Center of Advanced Technology

Table of Contents

1. Introduction 17
   1.1 Motivations 18
   1.2 Problems and Proposed Solutions 20
   1.3 The Projects 26
   1.4 Summary of Contributions 28
   1.5 Structure of the Thesis 28
2. Representing Action 30
   2.1 A Definition for Action 32
   2.2 Methods for Representing Actions 33
   2.3 Action Frames 37
   2.4 Reasoning Using Action Frames 44
   2.5 Formalizing the Language: ACTSCRIPT 50
   2.6 Problem: Actions Have Complex Temporal Structures 51
   2.7 Summary 53
3. Representing Temporal Structure Using PNF-Networks 54
   3.1 Reasoning with Temporal Constraints 56
   3.2 IA-Networks 59
   3.3 Past, Now, and Future 66
   3.4 A Better Representation: PNF-Networks 69
   3.5 Computing an Approximation of the PNF-Restriction 81
   3.6 PNF Propagation 86
   3.7 Future Directions 92
   3.8 Summary 93
4. Action Recognition using PNF-Networks 95
   4.1 Limitations of Current Methods for Action Recognition 96
   4.2 Representing the Temporal Structure 97
   4.3 Recognition Using PNF-Networks 101

4.4 Temporal Constraints vs. Detection Power 103
4.5 Detector Instances and Detectors 106
4.6 Other Examples of Results 108
4.7 Recovery From Errors 110
4.8 Experiments in the *"It"* Scenario 114
4.9 Future Directions 118
4.10 Summary 121
5. Interval Scripts 122
5.1 Problems with current Scripting Techniques 124
5.2 A Proposal: Interval Scripts 127
5.3 Basic structures of Interval Scripts 128
5.4 The Interval Script Engine 135
5.5 Handling More Complex Structures 143
5.6 Future directions 150
5.7 Summary 152
6. An Architecture and a New Domain for Story-Driven Interactive Spaces 153
6.1 Interactive Spaces 154
6.2 The Story-Character-Device Architecture 158
6.3 Computer Theater 162
6.4 Summary 169
7. Building Interactive Spaces 171
7.1 The *Intelligent Studio* Project 172
7.2 *"SingSong"* 184
7.3 *"It / I"* 192
7.4 *"It"* 211
7.5 Summary 218
8. Conclusion 219
8.1 Results and Contributions 220
8.2 Directions to Explore 222
8.3 A Final Word 224

Appendices 226
A. Script and Inferences in the Cooking Show Example 227
Action Frame Script 227
List of All Inferences for Each Action 228
B. Grammar of ACTSCRIPT 234
C. Action Frames for "Mix" and "Wrap" 237
Action Frame Representation for *"Mixing Ingredients"* 237
Action Frame Representation for *"Wrapping Chicken"* 240
D. Grammar of Interval Scripts 244
References 246

Acknowledgements

All the work described in this thesis in one way or another benefited from the ideas, sharp criticisms, and support from my thesis advisor, Prof. Aaron Bobick. As detailed in the text, we share the intellectual credit of many of the concepts and proposals. Aaron has been an admirable supporter of some of my craziest ideas (starting with my coming to the Media Laboratory) and boldest projects, including here his courage to back up my interest in computer theater. Aaron, it has been great working with you for the last six years and I thank you deeply for all I have learned from you.

I have to thank a lot of people, starting with my thesis committee, Rod Brooks, Ken Perlin, and Joe Bates, for all the insightful comments and for the needed criticisms. Rod, Ken, Joe, thank you for all the ideas and for your time, and I hope we can work together in the future.

Many great teachers were essential in indirectly shaping the ideas presented here, and, more generally, in my Ph.D. formation. I want especially to thank David McAllister, Roz Picard, Mitch Kapor, Ted Adelson, Bruce Blumberg, and Jae Lim for all the thoughts you gave and provoked.

A pivotal presence during my MIT years was Prof. Janet Sonenberg from the MIT Music and Theater Arts Department. Thank you for believing in me, for all the questioning and challenging, and for the support in pushing theater and drama into the Media Laboratory agenda. Prof. Claire Mallardi, from Radcliffe College, taught me how to look to my body, and attending her two courses gave a very much-needed air to my artistic aspirations. Later, Beth Soll and Joan Jonas were responsible for bringing dance and visual (performing) arts to my soul. Thank you very much, from the rejuvenated left side of my brain.

Each project that I was part of at the Media Lab brought me a wealth of knowledge, mostly from the people I worked with. The *SmartCam* project happened with the support of Frank Kao and Michael Bove, who helped me with the digitizing of the cooking sequences. Later, Andrew Wilson was essential in the process of digitizing the Alan Alda show. Also, very special thanks to Alan Alda, Graham Chedd, and all the crew of *Scientific American Frontiers* for the great help in recording the Alan Alda cooking show.

Going to Japan in the summer of 1996 marked a deep change in my career directions and moved computer theater from the drawing board to the stage. Thanks to the MIT Japan program and the Starr Foundation, and the support of Patricia Gercik, I spent an incredible summer at the ATR Laboratory, in the heart of oldest capital of Japan. First, I have to thank Kenji Mase and Ryohei Nakatsu for believing that I could materialize *"SingSong"* in 10 weeks, and for the support they gave me. Kenji also shares the credit for the first development of interval scripts, later to become a central piece of my work. At ATR I want to especially thank

Naoko Tosa for all the great discussions about interactive art, and the help of Sydney Fells, Kazushi Nishimoto, Armin Bruderlin, and all the other friends there. Domo arigato gozaimasu.

For all who participated in *"The KidsRoom"* project (which is not described here) I want to thank you for the opportunity of making me part of something really amazing. Our interactive bedroom was one of the finest moments of the Media Laboratory, and I learned immensely from all of you: Aaron Bobick, Stephen Intille, Freedom Baird, Jim Davis, Lee Campbell, Yuri Ivanov, Andrew Wilson, Arjan Schutte, Jon Klein, Alex Weissman, and my UROP working on the lighting control, Hugo Barra.

My full gratitude to the whole crew of *"It / I"*, for all of you who contributed their ideas and hard work to materialize my dream. First of all, thanks to Aaron Bobick, transformed into a theater producer, and to Janet Sonenberg for all the key insights and criticisms. John Liu and Chris Bentzel shared most of the hard-working, sleepless nights needed to make the play run, and without their commitment the play would have never happened. Raquel Coelho, thank you for not allowing me ever to forget my duty to beauty and for your incredible work as the art director of the play. Nathalie van Bockstaele volunteered ideas and heart to shape and deepen the script. Freedom Baird and Richard Marcus, thanks for the great music/sound and for the strength of the light design, respectively. Joshua Pritchard came from nowhere to become the perfect actor for the play, less than two weeks before opening. Leslie Boundaryk worked very hard to make the dream and the designs concrete. Monica Pinhanez faced the biggest piano she has ever played and provided the musical presence I needed in the performances. Thanks also to Andrew Wilson, Jim Davis, Maria Redin, Alice Cavallo, the HLV crew, Greg Tucker, Dennis Irving, Kate Mongiat, Erik Trims, Michal Hlavac, Sabrina de Carlo, and my two brave UROPS, Alicia Volpicelli and Nick Feamster. Finally, thanks to James Brown's music, who kept us awake in the middle of the long nights in the Cube (with our great, ping-pong-"playing" neighbors, Craig Wisneski and Julian Orbanes). Thank you all for making the idea of computer actors in a theatrical performance into a reality and a success.

This six years at the Media Laboratory have been a great experience, mostly due to friends and colleagues with whom I shared the best and the worst of my moments. Stephen Intille has been the perfect officemate, and our exchanges of ideas throughout these years have been incredibly rewarding. Thanks also to the rest of the HLV gang, Jim Davis, Andrew Wilson, Yuri Ivanov, Lee Campbell, Martin Szummer. And the friends from all the different regions of the Media Lab forest, now spread all over the world: Marina Umaschi, Tomoko Koda, Greg Kimberley, Linda Peterson, Laurie Ward, Kate Mongiat, Amy Bruckman, David Cavallo, Deb Roy, Sumit Basu, Teresa Marrin, Baback Moghadan, Sara Elo, Michal Hlavac, Tony Jebara, Dave Becker, Nuria Oliver, Warren Sack, Thad Starner, Fernanda Viegas, Adriana Vivacqua, Janet Cahn, Irfan Essa, Martin Hadis, Steve Mann, Kris Popat, David Tames, Ali Azarbayejani, and Martin Friedman. Also, thanks to all the people who I met in the Narrative Intelligence group, which was my true introduction to life at the ML and in the new information age. Too bad it ended. Thanks also to all students who are or have gone through the Vision and Modeling group, for the occasional and fundamental exchange of ideas. It has been awesome.

Thanks to all friends, at MIT, Boston, US, Japan, and Brazil who contributed their ideas and challenges, but specially their support and love throughout these years. Agnaldo Valentim,

Livia Pedallini, the *fofomac* crew (Dilma Menezes, Carlos E. Ferreira, Jose Coelho, Kunio Okuda, and all the others), Ellen Nakamizu, Tom Harsanyi, Linda Carrubba, Vinit Mukhija and all other friends from DUSP. A special thanks to my parents-in-law, who keep reminding me that Brazil need some severe fixing and who prayed so much for my success. And for the always-caring presence of my mother, whose example of surviving the worst always inspired me when things were difficult. Love to you all.

My last and deepest thanks go to Monica, my wife, who I owe the opportunity of being at MIT. Without her support I would not have applied, neither survived the many crisis and bumps of the last six years. Without your smile, your laughs, your support, and, most of all, your love, there would be no *SmartCams*, no PNF, no *"SingSong"*, absolutely no *"It / I"*. I hope I can be as supportive to you, in your quest at MIT, as you have been to me. Thank you, from the very bottom of my heart. I love you.

Thank you all for the most incredible years of my life.

My father, Roberto Santos, a movie director in Brazil, used to tell me about his dream of the day when his characters would become holograms and follow him around the town. The day when they would talk to him on the streets, cry while telling their stories inside a bus, and argue to each other at a party. The day when cinema would leave behind the darkness of the movie theater and the flatness of the screen.

My dream is to make his dream come true.

To my father.
(I miss you)

1. Introduction

The conjunction of real-time computer graphics, body-position sensors, fast computer vision and speech processing, and synthesized music and voice is creating a new medium where people are physically immersed in an environment by a world of images, sounds, and objects that react to their actions. We call these environments *interactive spaces*, where the world "space" is used to emphasize the difference to the non-materiality of virtual reality.

A central assumption of this thesis is that *action* is the fundamental building block of interactive spaces. This assumption creates a need for powerful methods to deal with the different aspects of actions and particularly with their temporal structure. In our work we employ the term *"action"* referring to both the human and the computer activities, and *"interaction"* as a coordinated set of multi-agent actions where the goal of each agent's action is to induce change in the other agents.

Throughout this thesis we propose solutions for the use, representation, and communication of actions and for the scripting of interaction in interactive spaces. A major emphasis is placed on the development of models for representing the complex temporal structures associated to human actions and on fast algorithms to perform basic reasoning on such structures. We also focus on interactive spaces that possess some form of narrative structure — *story-driven interactive spaces* — since the amount of coordination required by a narrative dramatically increases the need for the representation and recognition of actions.

Previous work on interactive spaces has mostly focused on either low-level issues as, for instance, user tracking, character motor systems, or process communication [2, 125, 186]; or meta-story level problems such as the automatic generation of interactive narratives [30, 52]. Our research addresses mostly middle-level issues involving the relationship between users and characters and the story, including the understanding of users' actions, the control of computer characters, and the development of the story in the context of an interactive space.

An important component of our work is an effort to test our ideas by building real, complex interactive environments. We chose this path firstly because it is often necessary to ground the development of novel concepts into real systems. But our main reason to build actual spaces is our belief that as important as to develop technology for interactive environments is to

understand how users act in such spaces, how their disbelief can be suspended, and how stories can be told — or, maybe we should say, lived — in this new medium.

1.1 Motivations

Our research is the result of the convergence of some trends observed in different areas. A first trend is the growing interest of the computer vision community in the analysis of video streams depicting human beings and their activities. Although tracking of body parts is still a problem attracting considerable amount of attention, there has been increasing interest on the recognition of human actions [20].

Recognition of human actions is important for many different applications. A traditional domain is surveillance, which is evolving from the automatic recognition of static images such as satellite pictures towards the detection of human activity in stores, airports, parking lots, and walkways (see, for example, [57, 72, 118]). It is also envisioned that action recognition can be used in systems for helping people with disabilities [165], and general, in human-machine interaction [122].

However, it is debatable if the pattern recognition-based approaches in use today (for example [26, 43, 181, 182]) are appropriate to handle the recognition of human actions. Recent work in vision has stressed the necessity of adequate symbolic representations for action [97, 111, 159]. The framework presented in this thesis supports this view and extends the previous research by proposing more expressive methods of representing the temporal structure of actions.

A second direction followed by our research aligns with recent developments in the field of artificial intelligence concerning methods for representing and controlling action. While there is still considerable controversy between employing deep reasoning [50, 101, 149] or reactive models in the design of intelligent creatures [16, 33, 37, 99], recent works have tried to find specific domains where middle-term solutions can be applied [39, 130]. Our work on action representation is based on the use of a small number of symbolic primitives (based on [149]) and on a simple reasoning system (inspired by [144]) that compensates for the weaknesses of the inference methods with extensive sensing and actuation in the real world.

A third trend is the increasing interest in the construction of interactive, immersive systems for entertainment [42, 44, 94, 121, 172], workplace [98], and simulation of real world conditions [77]. After the initial surge of interest in virtual reality (VR) applications (see [79]), a current research trend points towards the creation of full-body immersive worlds that have narrative structure [18, 41, 84]. Our research has been focused on such story-driven systems and especially on the representation of interaction through paradigms that enable the sensing system to exploit the knowledge about the user's actions by using the constraints coming from the story or narrative.

We can trace the human interest in full-body, immersive experiences as far as the history of ritual goes. However, it is in the 19th century that we start to see the creation of immersive illusions based on technology. The most notable example is the *panorama*, whose aim was *"…to reproduce the real world so skillfully that spectators could believe what they are seeing was genuine."* ([117], pg. 49). As shown in fig. 1.1, a panorama consists of an observation

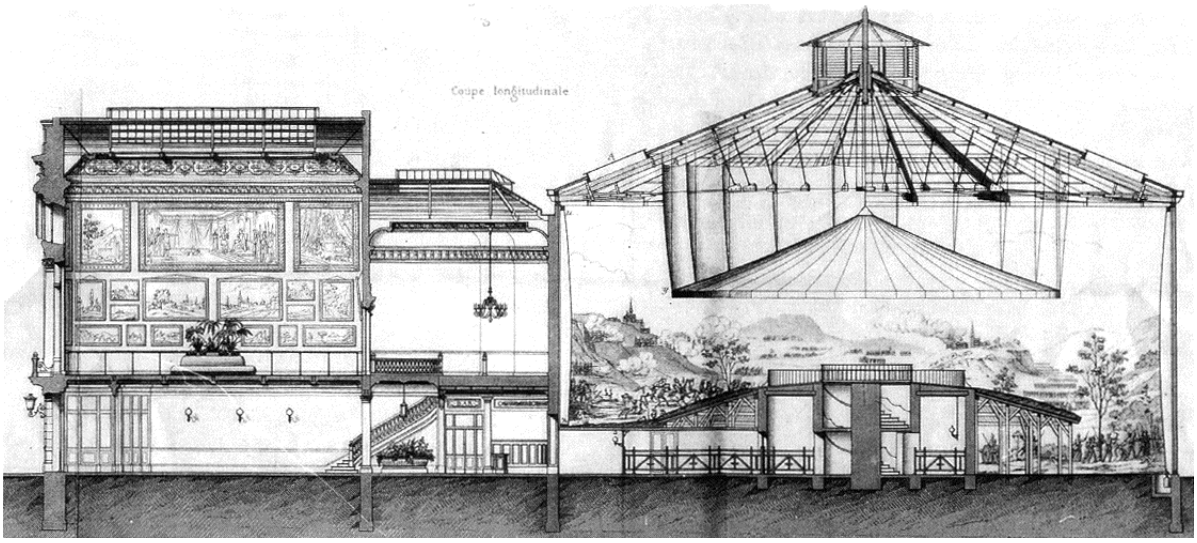


Figure 1.1 Cross section of the *Panorama Français* (reprinted from [117], pg. 142, from an original of the Bibliotèque Française).

platform surrounded by a painted canvas. To create the illusion of reality, the edges of the canvas are not visible, hidden by an umbrella-shaped roof that covers the platform and by a "false terrain" projecting from the observation platform. Panoramas were extremely popular throughout the 19th century, depicting landscapes, battle scenes, and journeys (see [117] for a history of panoramas).

From panoramas, the quest for an immersive experience has evolved through world fairs, zoos, and more recently, theme parks, and adventure rides (see [183] for a history of leisure centers). With the advent of computer technology, video games brought to life the possibility of interactive experience, though restricted to the world behind the screen. Virtual reality technology (3D stereo goggles, data gloves) tried to increase the immersion by erasing the contact of the user's body, as much as possible, with the real world. Our interest runs in the opposite direction, that is, to create a real space that is augmented by images, sounds, and interaction controlled by computers.

Finally, the fourth trend corresponds to the increasing interest in fusing art and electronic technology that is observed in the last half of the 20th century. Starting with the pioneering work in music by Stockhausen [91] and John Cage, electronic and computer technology have become an essential part of music [153], including automatic, interactive music systems [147], in what is commonly known as *computer music*. Similarly, computers are increasingly being used in dance, film, and visual arts.

Interestingly, the presence of computers in theater is surprisingly timid (for one of the few examples, see [142]). Many reasons contribute to this, but, according to our view, a major difficulty is the lack of standard computer representations for the fundamental element of theater: action. In this thesis we contribute to *computer theater* in two different ways: firstly, by

investigating computer mechanisms to represent and reason about action and interaction; and secondly, most importantly, by creating and producing theater pieces that actively employ computers.

1.2 Problems and Proposed Solutions

The main goal of this thesis is to investigate how human action and human-computer interaction can be represented in a computer to allow the control of an interactive space. Although we pay some attention to representational issues in the traditional AI sense, that is, by exploring their inference power, our focus in this thesis is on a more specific problem, the **mechanisms for representing and reasoning about the temporal structure of action.**

A common belief we challenge is that *finite-state machines* are good representations for the temporal structure of human action. Finite-state machines are embedded in the commonly used models for gesture and action recognition (especially in the probabilistic versions, using dynamic programming [40] or HMMs [26, 165, 181]). The drawback of such models is that they can not efficiently handle parallel actions or events. In particular, the number of nodes necessary to represent parallel actions increases exponentially with the number of actions.

Our research proposes temporal models based on **networks of temporal constraints**, where a particular instance of an action is not a path through the network (like in finite-state machines) but a collection of time intervals corresponding to the occurrence of each sub-action and world states. In our work we have used such models for both action recognition and interaction control.

Besides this main theme, this thesis also discusses other aspects that are more specific to the realm of interactive spaces. To facilitate the understanding of the scope of our contribution, we can divide our ideas and results as related to six basic problems, as described below.

1.2.1 How to represent human action?

The first problem we address in this thesis is how to represent human action in a computer in order to make recognition possible. We see *action* as movement happening in a context (according to Bobick [20]) and as such, representing an action comprises of both methods to describe movements (and their evolution through time) and the context where the action happens. Also, representing an action involves the definition of mechanisms by which the movements and context are integrated to characterize a particular action.

This thesis does not explore how to represent and recognize movements (see [20] for references of relevant work). Instead, our focus has been on methods to associate a movement and context to a specific high-level description. The goal is to have systems that, for example, given the information that *X* is doing a cyclic, left-right movement of the hand and that there is another person *Y* looking in the direction of the moving hand of *X*, could conclude that the action corresponding to the hand movement is "*X* is waving to *Y*".

Most attempts at logical formalization of action and, particularly, the effects of an action or movement in a context, appeared in research in either philosophy and linguistics (e.g. [68, 69,

138, 149]) or formal logic [50, 55, 148]. The linguistic approaches were in general aimed at the processing of natural language and are normally cluttered with semantic problems. Logic approaches have been basically aimed to develop formalisms to infer logical consequences of actions through deep reasoning mechanisms.

Logical systems typically experience difficulties when connected to real-world perceptual data, (typically present in movement detection). As an answer to this problem, procedural representations for actions have been proposed in different ways by Brooks [31, 32], Rosenchein and Kaelbling [146], and Chapman [37]. Actions in these systems are represented by the connections between actuators and sensors directly, but these representations present difficulties for the exploration of any kind of symbolic knowledge, making them vulnerable to context change.

Our work has been taking a middle ground between the approaches described above. We have proposed a paradigm called *action frames* based on Schank's *conceptualizations* [149] as a formalism to represent action. According to this formalism, actions are represented by the composition of smaller units that represent the sub-actions that are part of the main action and particular states of the agent, objects, and the world (see [115, 171] for psychophysical evidence). Action frames is a classic frame-based representation [105], where each unit belongs to one of 11 basic categories of primitive actions.

This framework for representation possesses a very desirable feature. It is possible in specific domains to implement inference systems that can infer from a high-level description of an action its component sub-actions and some of the world states that are changed by the sub-actions. This decomposition can ultimately lead to the generation of the movements and states that can be perceptually detected by a recognition system. As we introduce in the next sub-sections, it is possible to connect such sub-actions and states to real sensors and, together with information about temporal structure, assign a high-level action to a specific period of time given the pattern of occurrence of the sub-actions and states.

Our answer to the problem of how to represent action is quite biased towards our intended application: action recognition. In this sense, we are less interested in the inference system that produces the low-level representation of sub-actions than in the feasibility of the representation method as a model for recognition. Nevertheless, we have implemented a simple version of an inference system for a specific domain.

Action frames were initially developed as a way to represent context for vision routines in an application involving automatic cameras — *SmartCams* — framing a cooking show, as explained below. Action frames have been developed further into the definition of a multi-purpose language, called ACTSCRIPT, that has been used mostly to communicate actions, action requests, queries, and goals in interactive spaces.

1.2.2 How to represent the temporal structure of human action? How to efficiently handle complex temporal structures?

Assuming that an action is decomposable into sub-actions and states of the agent, objects, and the world, and that there is a method of representing them, there is still one missing element: time, that is, how these sub-actions and states happen through time. Such temporal structure can define the difference between two actions. For instance, a system that only detects whether or not a person's mouth is open can distinguish between the actions *"opening the mouth"* and *"closing the mouth"* just by considering the order in which the two states happen.

Most of the systems used in recognition of actions assume that the basic temporal structure of every action is a sequence of sub-actions or states and represent time evolution through *finite-state machines* (or probabilistic equivalents such as *HMM*s). However, our analysis of the data obtained in the *SmartCam* project showed that, in fact, single-agent actions involving three or more parallel threads of sub-actions are very common. Also, the number of parallel actions tends to increase in the case of multi-agent activity. Since finite-state machines have to exponentially increase the number of states as the number of parallel actions increase, we started to look for more compact representations for the temporal structure of an action.

As the underlying structure for the construction of characters, representations of users, and interaction scripts, we have been using a method for representing the temporal structure of an action as an *interval algebra network*, or simply an *IA-network*, as defined by Allen [3]. An IA-network is a constraint satisfaction network where the nodes correspond to variables on time intervals and the arcs correspond to binary temporal constraints between the intervals — expressed using disjunctions of Allen's primitive temporal relationships [3].

An interval algebra network is one of the most expressive methods to represent time devised by the AI research community. Besides making simple the representation of parallel threads, even when they have multiple synchronization points, the interval algebra allows the representation of mutually exclusive actions and states, unlike weaker methods such as *point-based* temporal algebras [177]. However, there are difficulties in doing temporal constraint propagation in an IA-network. For instance, to determine the occurrence of actions we have developed an approach based on the computation of the *minimal domain* of the IA-network, the set of all time intervals that belong to at least one solution. The problem is that most problems in temporal constraint propagation in interval algebra networks are computationally difficult [177] and, in particular, the computation of the minimum domain is *NP-hard* [176].

A central theoretical result of the thesis is the development of a method that circumvents the computational complexity of the computation of the minimal domain by projecting of the IA-network onto a specialized, 3-valued *(past, now, future)* constraint network called a *PNF-network*. Although computing the minimal domain in finite constraint satisfaction networks is also, in general, an *NP-hard* problem, in our experiments we have been employing an *arc-consistency* algorithm (based on [90]) to compute an approximation of the minimal domain of the PNF-network in time linearly proportional to the number of constraints. The approximation is shown in this thesis to be conservative, that is, it always contains the minimal domain and, moreover, also contains the projection of all the solutions of the original IA-network.

We have been applying the approximation method for recognition of actions and for the control of interactive spaces. In practice, we have found no situation where the difference between the approximation and the actual value would have had a negative impact either in the recognition of an action or in the activation of a control structure.

1.2.3 How to recognize actions with complex temporal structures?

Given an action represented as a set of sub-actions, some of which are directly detectable by perceptual routines, and the corresponding temporal structure, our goal is to determine whether the action — and/or each of its component sub-actions — is currently happening given the input from perceptual sensors and, if necessary, the previous state of the world.

Kuniyoshi and Inoue [81] used finite automata to recognize human manipulations of blocks. Probabilistic methods for visual action recognition have been proposed in the computer vision community using *HMMs* [24, 26, 165] and stochastic grammars [22]. However, as noted above, such approaches are not able to represent efficiently parallel threads of actions that occur in many everyday actions.

Moreover, we believe that it is important to exploit the fact that logical impossibilities prevent the occurrence of some sequences of sub-actions, and that it is necessary to incorporate such constraints into vision systems designed to recognize human action. Some work in the vision community (e.g.[65, 111, 159]) has attempted to incorporate logical definitions of action into perceptual mechanisms. However, these systems are unable to cope with most of the complex temporal patterns of everyday actions that include external events, simultaneous activities, multiple sequencing possibilities, and mutually exclusive intervals [6].

Using the IA-network representation for temporal structure described in the last section, we have developed an effective method for using that representation to detect the occurrence of actions. In our approach, we project the IA-networks into PNF-networks and the state of the main action and of each sub-action is determined by computing the minimal domain of the network, considering as input the state of the nodes corresponding to perceptual routines.

This process is the basis of our PNF propagation algorithm that determines the current state of the sub-actions and states of a PNF-network considering not only information from perceptual sensors but also the previous state of the sub-actions. PNF propagation represents sensor values and previous states as unary constraints on the PNF-network and, by computing the minimal domain, is able to rapidly determine whether the nodes in the action are happening or not. In this thesis, we prove that the PNF propagation method is conservative, that is, its result always contains the true state of the IA-network.

A problem with the PNF approach is the common brittleness associated with constraint satisfaction systems. Although it is not clear how to integrate probability without losing the linear complexity, we have experimented with some methods for recovery from sensor errors. The basic idea is to keep track of the multiple threads of possible, non-contradictory configurations of states that correspond to the different error conditions in each time step. Surprisingly, we have found that the potential explosion in the number of different threads is prevented by the occurrence of frequent contradictions between the error conditions through

time. This thesis presents experiments with manually extracted data and with real sensor values, showing that actions are detected by this method in spite of noisy sensors.

1.2.4 How to represent the temporal structure of human-machine interaction?

In an interactive system, complex temporal structures exist not only in the actions performed by the user but also due to the interaction among the multiple output modules and the characters. The management of how the multiple, parallel events unfold can become difficult because it is necessary to follow the story or interaction scheme considering response delays and generation constraints from the input and output devices (for example, when only one sound file can be played a time). Successful story development also requires precise timing among the actions of different characters.

The objective of our research in this area is to create an interaction scripting language that handles such complex temporal structures. This work extends the work on action representation for recognition by using the same underlying structure, PNF-networks, and by applying a variation of the PNF propagation method in the run-time control engine of the interactive system.

Previous work on scripting languages for systems with narrative structure has focused on different aspects. Bates [14], Perlin [125] and Blumberg [16] experimented with different ideas for behavior design, using traditional AI planning and behavior-based approaches. There has also been interest in languages for description of multimedia interaction as for example *Director* [1] and the works of Buchanan and Zelllweger [34] and Hamakawa and Rekimoto [59]. Those languages, however, lack appropriate ways to represent the duration and complexity of human action in immersive environments: hidden in the structure is the assumption that actions are pinpoint-like events in time (coming from the typical point-and-click interfaces for which those languages are designed) or a simple sequence of basic commands.

Moreover, most of the work referred above assumes either that device response is immediate and reliable or that duration of human actions is very short. Those assumptions are almost never true in an actual interactive space. To overcome these limitations we propose in the thesis the *interval script* paradigm that associates a temporal interval without pre-fixed duration to every action and sensed state of the world.

An interval script is a computer file containing the description of each action and statements about how the intervals are related temporally. The paradigm introduces many improvements over previous languages for scripting interactions. First, interval scripts de-couple the desire or need for an action and its actual occurrence. This is a major difference to other languages for handling device-level events like *Isis* [2]. Interval scripts allow the designer to forget the details involved in managing delays and device failures. Second, it is easy in the interval script language to compose actions from previously defined sub-actions and, at the same time, to automatically control the activation and scope of the occurrence of the sub-actions.

Third, interval scripts also allow subjecting actions to explicit temporal constraints declared by the designer of the system, including the declaration of mutually exclusive actions. Unlike in

André and Rist's work [9], with PNF propagation it is possible to employ strong temporal algebras and still handle high levels of interactivity in real-time.

The temporal constraints in an interval script file define an IA-network that is automatically projected onto a PNF-network prior to run-time. In our implementation, the file defining the interaction is translated into C++ code that describes both the associated PNF-network and the functions used to start, stop and compute the actual state of intervals. The methods and functions defined in the C++ code are used by the interval script run-time engine. During run-time, the engine first computes the current state of each node of the PNF-network (corresponding to the occurrence of sub-action and states) and then compares it to the desired state of that node in the next instant of time, as determined by a variation of the PNF propagation method. In the case that the desired state of a node is different from its current value, an action of starting or stopping the corresponding sub-action is taken accordingly. A major advantage of employing PNF-based temporal constraint propagation is that it is possible to compute simultaneously desired states for all the nodes so they satisfy every constraint.

The validation of our scripting method comes from its use in three complex projects of interactive spaces that involved IA-networks of up to 300 nodes in situations where fast design and development of the interactive space and high reliability were crucial.

1.2.5 How to build physical, story-driven interactive spaces?

There has been an increasing interest in creating interactive spaces for entertainment with a strong narrative structure or with an underlying story [18, 71, 112, 174]. These systems have employed different control architectures, each of them with several shortcomings. In this thesis, we are also interested in determining what would be a good architecture for a story-driven interactive space.

In our view, centralized story control is essential for an interactive system to achieve successful story development. However, such a claim is clearly contentious. Perlin and Goldberg [125] as well as Bates et al. [14] built (semi-) autonomous computer-actors where the story was distributed among characters or seen as the natural result of the interaction between the characters and the user. However, as noted by Langer [83] and Murray [109], well constructed stories require coordination and synchronicity of events and coincidences; also, dramatic actions have to forecast the future, especially in the context of theatrical stories.

To answer these concerns, we have proposed a three-level architecture for story-driven interactive systems called *story-character-device* architecture, or *SCD*. The defining characteristic of the SCD architecture is the separation between character and story control. In the SCD architecture the characters are semi-autonomous, receiving commands from the story control module although possessing the ability to sense the on-going action and to coordinate the accomplishment of their goals according to the reality of the stage. Also, as in other approaches [16, 125], characters are separated from device specific concerns.

Evaluating the usefulness of a system's architecture is quite difficult. In our case we have implemented two similar stories firstly without and later with the SCD approach and have found the implementation considerably simpler in the second situation.

1.2.6 How can computers be incorporated to theater?

Instigated by the realization that theater is the performing art with the most inexpressive use of computers (compared to music, dance, and even visual arts), we have started to ask why theatrical performances involving active use of the computers — baptized by us as *computer theater* [129] — are so rare (with few exceptions, among them the work of George Coates in San Francisco, Reaney's *"The Adding Machine"* [142], and a recent experiment of Bob Wilson, *"Monsters of Grace"*).

We have answered this question in two different ways. First, we analyzed what fundamental elements of theater must be represented in a computer and concluded that a major obstacle for computer theater performances is simple and powerful representations for human action. In this sense, this thesis is also a contribution for the technology of computer theater. Second, we have been actively experimenting with computers in theater plays, particularly in the case where the computer plays one of the characters in the play. Our conclusion is that there is an amazing space in theater to be creatively explored by play-writers, actors, and directors.

1.3 The Projects

Our proposals have been tested in four different projects that involved the design and implementation of an interactive space. The four projects have many elements in common: first, all of them have a narrative structure, that is, they are *story-driven* interactive spaces. Also, in all projects the script of the story is available in some format for the computer controlling the space. Second, they all employ cameras as their main sensing device. Third, the projects, not surprisingly, involve people performing actions. And fourth, in all projects but the last one the interactive space is used for a performance.

The first project we developed was the *Intelligent Studio*, a TV studio where cameras automatically frame TV shows under the direction of a human director. The main goal was to implement a *SmartCam*, a "robotic" camera that could answer, through video processing, requests for specific shots such as *"medium shot of X"* or *"close-up of object Y"* — the normal commands used by TV directors. In particular, we selected a cooking show as the domain of our experiments because of the difficulty normally associated in framing cooking actions and the variety of types of actions that are performed in such shows. So far, the *Intelligent Studio* project has been the major testbed for the *action frames* formalism proposed above, and it was the particular domain where we have developed our inference system for determining the sub-actions of an action.

Starting in the fall of 1994, and in parallel to the development of the *Intelligent Studio*, we have grown increasingly interested in exploring dramatic performance with computers. In the spring of 1996 we realized that *computer theater* could be a very good domain for action recognition for many reasons: the basic element of drama is action; what happens in a play is described in the script; and gestures and movements in theater are explicitly and purposefully performed. Also, if a computer plays one of the characters, we have a situation where the recognition of actions is critically important and in which the results are clearly visible.

The first opportunity for a project combining our scientific interests in action recognition with our desire to bring computers to the realm of theater came during a summer spent at the ATR Laboratories in Kyoto, Japan. There we conceived, designed, implemented, and presented *"SingSong"*, a short performance where a human clown interacts with four computer graphics characters projected on a large screen. The characters are automatically controlled by a system using both vision processing and a loose, non-timed representation of the script.

In *"SingSong"* we started developing our work in paradigms for scripting languages. To develop the paradigm further, as well as to expand our work with computer theater, we decided to embark in a more ambitious computer theater project that culminated with live performances of the play *"It / I"* in November of 1997.

"It / I" is a pantomime where a human character — called ***I*** — struggles against a machine-like character controlling his environment — ***It***. The machine character is played automatically by a computer monitoring the stage with a three-camera system. The character ***It*** appears on stage in the form of animated computer graphics objects projected on two stage screens. The play is composed of four scenes telling the journey of ***I*** in his interaction with ***It***, exploring different interaction modalities such as the human actor's position on the stage, his gestures, and manipulation of props.

The project *"It / I"* posed enormous technical challenges. First, the control system had to run smoothly for a long period of time while progressing through the different scenes of the play. The implementation of the automatic character had to be flexible enough to be improved after each rehearsal, and thus, the play became the major test for our concepts and proposals about scripting paradigms.

After each performance of *"It / I"* we invited the public to go up on stage and re-enact one of the scenes of the play. This follows our concept of *immersive stages* where we envision spaces that contain an interactive story that can be both explored by a performer or experienced directly for a user or member of the public. Although the experience was very interesting, we found that it was necessary to refine further the control system in order to really achieve user immersion.

The project *"It"* aimed exactly on building an interactive space that recreates the basic story of the play *"It / I"* for a user without previous exposure to the material of the play. Unlike the other three projects, *"It"* is not a performance space but a story-driven interactive space for a user. Although we employed a physical structure similar to the one used in the play, with *"It"* we had the opportunity to correct some problems in the interval script paradigm that were observed during the production of *"It / I"*. In fact, in this thesis we present interval scripts following the syntax of this last implementation.

1.4 Summary of Contributions

The contributions of this thesis are:

i. The **action frames** formalism to represent actions that is an augmentation of Schank's conceptualizations [150] by the inclusion of *IA-networks* [3] to represent the temporal structure of the sub-actions.

ii. **PNF-networks**, a specialization of *IA-networks* that allows the computation of a good approximation of the *minimal domain* in linear time;

iii. the **PNF propagation** method that combines sensor input, past information, and the temporal structure of actions to enhance recognition;

iv. the **interval scripts** paradigm for describing interaction that allows the use of strong temporal constraints between actions and events in a script;

v. the **story-character-device architecture (SCD)** for interactive story-driven systems that separates story from characters and devices into different levels, allowing centralized story-control of semi-autonomous characters;

vi. the definition and classification of **computer theater**;

vii. ***"SingSong"*** and ***"It / I"***, the first public performances of autonomous computer characters in theater.

1.5 Structure of the Thesis

The structure of the thesis roughly follows the sequence of problems and solutions described in section 1.2. In **chapter 2** we discuss methods for representing actions and present the idea of *action frames*. In that chapter we also examine a particular inference system we have built for the *Intelligent Studio* application. It is important to notice that the main goal of chapter 2 is to introduce the framework that we assume for the rest of our discussion: actions are characterized both by specific movements and changes and by contextual information and can be decomposed into smaller units that are related to each other by a complex temporal structure.

Within that framework, we focus **chapter 3** on the discussion of temporal representation and reasoning. That chapter reviews the most common formalisms for time representation and examines in depth interval-algebra networks. Following we define *PNF-networks*, investigate the properties that link the two networks, and demonstrate that the minimal domain of a PNF-network can be approximately computed in linear time (in the number of constraints). We end the chapter formally defining the *PNF propagation* method and by proving that it preserves all the solutions of an IA-network. This chapter is essentially theoretical and aims to define formally the methods and ideas used in the rest of the thesis.

The goal of **chapter 4** is to present our results in using *IA-networks* to represent actions and *PNF propagation* to detect their occurrence. We first review the literature in the area and follow by examining an example of representation and recognition of a simple action using our method, assuming perfect sensors. After examining two, more complex examples, we propose a

method to accommodate a limited amount of error in the sensor information and demonstrate its adequacy in an experiment using real sensor data from the project *"It"*.

In **chapter 5** we introduce through simple examples the concept of *interval scripts*. The core of the chapter is the description of the run-time engine that reads the interval script file, monitors sensors, and takes actions to assure that the sensed state of the world remains compatible with the structure described in the script. Additional structures allowed by the language are examined afterwards in an example describing one of the scenes from *"It"*.

Story-driven interactive spaces constitute the theme of **chapter 6**. The basic goal of this chapter is to analyze the concept of interactive spaces and to propose the *SCD architecture* for story-driven spaces. We also include in this chapter our analysis of *computer theater*, composed of a review of theatrical experiences involving computers, a proposal for a classification, and a report of our involvement with computer theater.

Chapter 7 presents the four projects used to evaluate the ideas of the thesis. Besides examining the impact of the different proposals in each project, the chapter also describes the technology, design, and final results of each project. A reader interested in building actual spaces will find an enormous amount of practical information in the descriptions of these projects.

We conclude the thesis with **chapter 8**, which summarizes the contributions, evaluates the results, and comments on directions to be explored. The thesis also includes a list of references and four appendices containing extra information on the inference results (appendix A), the language ACTSCRIPT (appendix B), the representation of two actions in action frames (appendix C), and the syntax of the interval script language (appendix D).

Most of the ideas presented in this thesis are the result of joint work with my advisor at the MIT Media Laboratory, Prof. Aaron Bobick. The initial work on interval scripts included researcher Kenji Mase from the ATR Laboratories (Kyoto, Japan). The writing and directing of *"It / I"* was advised by Prof. Janet Sonenberg from the MIT Music and Theater Arts Department. Because of this joint work, but mostly for stylistic reasons, we opted to have the thesis written in the first person of the plural ("we"). However, all the opinions expressed in the thesis should be attributed solely to the author unless explicitly noted.

2. Representing Action

The goal of this chapter is to introduce the basic concepts on representation of actions that frame our work in action recognition described in the next chapters. We also introduce one of the basic assumptions that permeates this thesis, that actions are discrete units of change where the object of change is either the state of the actions' agents, other agents, or the environment.

We differentiate action from body and object movement and from gestures and activities. This is addressed in section 2.1 of this chapter where we adopt Bobick's definition of action [20] that views action as a movement happening in a context. In this view the consequences of an action and the intention of the agent are intrinsic components of what an action is. Empirical evidence for this comes from Newtson et al. [115] and Thibadeau [171], based on experiments where people perceived actions as discrete units of time where agents accomplish goals.

The main issue discussed in this chapter is how to represent an action, particularly in a form that is suitable for action recognition. From the different methods proposed in the past (as described in section 2.2) we adopted a symbolic representation paradigm based on Schank's *conceptualizations* [150]. Our representation, called *action frames*, is a simplification of the original proposal and aims to create a framework suitable for automatic recognition of human action. In section 2.3 we describe, through examples, the basic ideas behind the translation of linguistic descriptions of actions into the *action frames* formalism. Later, in chapter 4, we propose the augmentation of conceptualizations with a mechanism to represent temporal structures based on Allen's interval algebra networks [3].

Throughout this chapter we present examples drawn from the script of a TV cooking show where a chef demonstrates how to make a dish with chicken fillets. The choice of this particular example and domain comes from our project to create an *Intelligent Studio*. As described in chapter 7, this project focused on developing a space where automatic cameras frame elements of a TV show under the command of a TV director. During the development of the project we found that the script of a TV show contains a great deal of information about the actions happening in the studio. In the *Intelligent Studio* project we developed an architecture in which the action frames corresponding to the actions in a script provide critical information for the control of the automatic cameras [130].

In the definition of the action frame representation we paid particular attention on how the formalism entails easy decomposition of the action into its component sub-actions and how it is possible to automatically identify some state changes implied by an action. Section 2.4 explains how the information in an action frame can be detailed by simple common sense reasoning inference procedures based on Rieger's work [144]. As an example we describe the implementation of the inference system for visual attributes used in the *Intelligent Studio* project.

The action frame paradigm has been developed further into a formal language we call ACTSCRIPT, which we briefly describe in section 2.5 In ACTSCRIPT the action frames are expressed in plain ASCII text strings that can be interpreted into C++ objects. Although the basic mechanisms for inference have been implemented, we have not used ACTSCRIPT in situations where inference is necessary. Instead, ACTSCRIPT has been primarily used as a communication language between high-level modules in interactive spaces.

However, as pointed above, the aim of our representation is automatic recognition of actions. We conceive recognition systems using a representation scheme where perceptual features can be associated with the sub-actions and states that correspond to an action. During recognition these features are matched to the output of sensor routines. However, an action is characterized not only by its sub-components, but also by the order in which they occur. In section 2.6 we argue that the sub-actions of an action often occur across multiple threads subject to mutual temporal constraints. However, most of the currently used schemes for representing temporal structure of actions, based on finite-state machines, implicitly assume a sequential structure of the occurrence of the sub-actions.

This incompatibility between the temporal mechanisms adopted by most computer vision recognition systems and the reality of everyday actions is the main motivation for most of the work developed in this thesis. In many aspects, the main goal of this chapter is to establish the need for more elaborate temporal structures. The next chapter will discuss this issue in detail and propose the use of Allen's time interval algebras [3] to model the temporal structure. As we will show throughout the rest of the thesis, it is possible to develop fast mechanisms to recognize human actions and to control human-machine interaction based on this formalism.

Finally, we present here a framework where actions are not in the exclusive realm of human beings. When machines move bits or atoms in a context and with a perceived intention, it makes sense to talk about recognition of *machine actions*. Since action is a movement in a context, even if a machine has complete access to its own state the recognition of the actions it executes also depends on determining the context in which the machine is immersed. For example, when a computer displays on the screen a message about a printer error, we can say that the action "*warning the user about a printer error*" is happening only if the user is there to actually read the message. Although the movement corresponding to the action had happened (the message was displayed), the context (the user reading it) might not be occurring.

Finally, there are actions that involve multiple agents, such as "*arguing*" or "*negotiating*". Their representation can be accomplished by considering sub-actions performed by different

agents. In particular, the considerations in this chapter about representation of human action can be applied to the problem of representing human-machine interaction.

2.1 A Definition for Action

The first issue to be considered is to define the term *action*, particularly in the case of how we perceive the occurrence of an action. First, it seems that every action is associated with a change or movement. Miller and Johnson-Laird [104] argue that human perception of action happens at least in four different categories: objects, attributes, states, and events where *"Events are perceived when changes occur; not all changes are perceived as events, of course, but all events involve changes of some kind… The generic term for the thing that changes is 'state'."* ([104], pg. 85). Movements, following this definition, can be seen as events that have duration. However, not all movements are associated with actions: as pointed by Newtson et al. [115] and Thibadeau [171], actions are perceived as discrete units with a clear begining and end. These discrete events are termed *behavior units* in Newtson et al. [115].

Moreover, as discussed by Thibadeau [171], *"The perception of causality is important in the perception of many actions, but it may not be involved in the perception of all actions."* ([171], pg. 118). Bobick [20] advocates this distinction between discrete units of behavior that have causality and contextual components and those that do not. His classification is suggested in the context of computer vision and distinguishes three categories of problems in recognizing human motion, according to an increasing use of knowledge: *movements*, *activities*, and *actions*. Basically, a *movement* is *"... a space-time trajectory in some configuration space."* ([20], pg. 1258), whose recognition is independent of contextual information, except for viewing conditions. An *activity* is a sequence of movements that can be recognized based on the statistical properties of their temporal relationships. Finally, an *action* is defined to *"…include semantic primitives relating to the context of the motion."* ([20], pg. 1258) By defining action as a movement or set of movements in a context, its recognition requires the inclusion of knowledge about the context and the domain, and the hypothesizing and verification of goal achievement.

Similarly, Israel et al. define action as *"…the content properties of agents that agents have in virtue of performing [those] movements in certain circumstances"* ([65], pg. 1060). The interesting point brought by their discussion is the fact an action must be always associated with an agent and does not exist as a temporal entity but, instead, it is a temporal property. In other words, actions are always executed by agents, and are interesting *"… insofar as they are done purposively, intentionally, vindictively, and the like."* ([65], pg. 1061).

We adopt here Bobick's definition for an action [20], implicitly assuming the discreteness advocated by Newtson et al. [115] and the agency of Israel et al. [65]. That is, for the scope of this thesis, an *action is a discrete unit of behavior that involves a movement in a given context*,

action = movement + context

In a general sense, context is the situation where the movement happened. In the case of machine recognition of action, we can define context more precisely to be all the information about the situation that the perceiving system has besides the movement itself (see also [19]).

Of course a change in context produces a change in the associated action. For example, the same movement of the fingers is perceived as *"typing"* if it happens on a computer keyboard, and as *"playing music"* if it happens on a music keyboard. Recognizing an action is, thus, the process of attaching a symbolic label to a movement. This requires both the detection of the movement and an assessment of the current state of its agent and of the world.

Notice that our definition includes as actions the movements of machines and objects in a context: *"copying part of the memory to a disk"* is, after all, a movement — a "behavioral" unit with duration. In the context where the chunk of memory is a self-contained unit (as perceived by the user), the movement can be labeled as the action *"saving a file to disk"*. Even in the case where a machine can access every bit of their memory and state, determining the machine's action also involves determining the current context that, for practical purposes, normally will require some assessment of the state of the environment or, in human-machine interaction, the state of the user.

Moreover, actions are discrete segments of time they can be perceived in different granularities. The action *"picking-up a glass from a table"* can be regarded as a unit, but also as composed of the sub-actions *"approaching the glass"*, *"grasping the glass"*, and *"lifting the glass"*. Actions can be sub-divided but the smaller sub-actions respect the boundaries of the original action (see [115]).

Another situation is when different agents perform different sub-actions. For instance, when we say that two persons are arguing, we are describing an action that involves sub-actions from two different agents. We use the term *interaction* for the special case of a multi-agent action where the objective of each agent's actions is **to cause change** in the other agents. For instance, when two people argue, each is trying to produce a change in the other person's mental state. When a user types in a keyboard, his intention is to produce a change in the machine's memory state that is usually answered by a change in the screen to produce a change in the user's visual field. Therefore, according to our definition, action and interaction are considered similar concepts, and applicable both to people, machines, and human-machine interaction.

2.2 Methods for Representing Actions

Having defined the meaning of the terms *action* and *interaction*, we can go back to the main theme of this chapter: how to represent action in a computational form. First, we assume the fundamental hypothesis of Newtson et al. and Thibadeau [115, 171], that is, that actions are discrete units. Our second assumption is that an action can be decomposed into smaller actions, or *sub-actions*. Finally, we assume that the result of an action changes the *state* of the agent, of another agent, or the environment.

Since action is movement in a context, it is essential that any computational representation for action provide a mechanism to store context. While a representation for movements can mostly rely on describing functions on features spaces, representing an action must include the

contextual elements that differentiate that action from any another action with the same associated movement. Consider the examples of *"typing"* and *"playing piano"*: in order to distinguish the two actions, the representation of each action must differ from each other about the place where the identical pattern of movement of the fingers is happening.

2.2.1 Representation and Recognition

Although representations for actions have multiple uses, we examine here primarily representations suitable to be used by a perceptual recognition system. This imposes some requirements in the representation method. First, it must provide means for the description of the movement and its different qualities, speed, shape, appearance, and flow, in a format that enables machine recognition. As discussed below, the majority of the work in visual recognition of actions is, in fact, concentrated in the recognition of movements in a fixed context.

However a full action recognition system needs to identify both movement and context. While movement must be perceived by sensory systems, context can be obtained not only by perceptual mechanisms but also by accessing information from other sources. For instance, a vision system that knows that there are no pianos in a room can automatically rule out that a *"playing piano"* action is happening when movement of fingers is detected over a surface in that room.

Given a representation of an action and information about the occurrence of movements and about the context, how can the action be determined to be happening — that is, in our terminology, *detected*? In general, a recognition system compares the temporal evolution of the sensory data and of the context with the representation and, according to some metric, decides if the two are similar or not. Notice that the similarity metric has to consider two aspects: whether the sub-actions and states of the representation correspond to the sensory and contextual data, and whether the temporal sequence of movements and states is compatible with the temporal structure of the action.

We postpone until the next chapter the consideration of the methods to represent the temporal structure. Here, our intention is to discuss representations for the sub-actions and states and, particularly, how a representation of an action can be used to infer its composition and its effects in an environment.

2.2.2 Representing Movement

First, let us examine briefly the ideas used to represent the movement component of an action. Many different computational representations of movement have been used in computer graphics. In most commercial software, movements are represented as temporal trajectories in a space of variables, normally associated with joint angles and translational changes of the center of coordinates (see [8, 10, 170]).

Such representations pose many problems for the inverse problem, that is, to recognize and classify observed movement. Rohr [145] used 3D cylindrical models to recognize walking movements. Polana and Nelson [139] used the distribution of motion in the area of the action to

recognize among seven types of activities such as running and jogging. Gavrila and Davis [53] used 3D models to track upper body movements. Recent work by Bregler and Malik [29] proposes exponential maps and twist motions to simplify the recognition of movements of articulated structures such as the human body. Bobick and Wilson [24] proposed the use of sequences of states of eigen-vector projection coefficients in a configuration space, allowing the direct use of imagery in the representation of movement. Davis and Bobick [43] proposed the use of view-dependent temporal templates for the representation of full-body movements and a recognition mechanism based on Hu moments.

2.2.3 Representing Change and Causality

A great deal of work in artificial intelligence has been concerned with how to represent change and causality. Starting with the pioneering work of Nilsson [50], McCarthy [101], and Sacerdoti [148], action representation has gone into increasingly levels of sophistication. A good survey, exploring in detail the temporal structure, is given by Allen and Ferguson in [6]; see also [166]. More recently there has been much interest in the concept of *fluents*, particularly in the work of Gelfond and Lifschitz [55], later extended by Lobo et al. [85].

The general problem of logic approaches is that the representations are very computationally expensive, making unfeasible their use in practical situations of recognition or control. Also, most of the formalisms can not cope with the noise of sensory systems and typically have not been used in conjunction with real data. Exceptions are the works of Maddox and Pustejovsky [92], Kuniyoshi and Inoue [81], and Siskind [159].

Robotic systems have used normally simplifications of full first-order logics where each action of a robot is represented in terms of pre-conditions and post-effects [116]. Normally these systems assume that when an action is being executed no other action happens, allowing the use of simple search methods for planning. Another popular scheme employs AND/OR trees (see [116]) to represent actions and goals as alternate layers, as described in [38], where an action accomplishes a conjunction of goals, and to achieve a goal there are multiple possible actions. A limitation of this approach is that the behavior of an agent can only be represented as a sequence of actions that accomplishes a sequence of goals, therefore excluding actions that involve parallel, coordinated execution of sub-actions. As we will see later, human actions tend to include multiple threads of sub-actions and states that are concurrent, making this representation inadequate.

2.2.4 Representing Sub-Actions

As mentioned above, actions can often be decomposed into simpler sub-actions. A variety of methods have been employed to represent the sub-actions and their temporal structure. We defer the review of different temporal formalisms to the next chapter and briefly summarize here how the sub-actions themselves are represented.

Decomposition of problems into smaller parts have always attracted the interest of researchers in artificial intelligence, and in 1963 Slagle introduced the use of AND/OR trees to decompose problems [160], in the context of solving symbolic calculus integration. At AND nodes, the sequence of sub-actions necessary to achieve the action is represented as the node’s children.

At OR nodes, actions that have identical effect are encapsulated in the children nodes. Therefore, an instance of an action is represented by any sub-tree of the original tree with at most one OR node per level, and consists of a particular sequence of sub-actions. As in the case described above, AND/OR trees are hardly adequate to represent parallel actions. In fact, using a more generic definition of AND/OR trees, it is possible to prove that they are equivalent to context-free grammars [58].

Minsky's idea of *frames* [105] have been adapted by some authors to represent actions and sub-actions. In particular, Roger Schank has proposed *conceptualizations* [149] where all verbs are represented by a composition of 11 primitives. In the next sections we discuss how to adapt his model for the representation of actions in machine recognition systems and for communication among modules in an interactive system. Notice that although Schank's original formulation does not allow the expression of more complicated temporal structures, it does not exclude them like some of the other proposed representations for action. Frames were also used by Neumann [113, 114] to represent actions in the domain of traffic understanding.

A very popular model for an action composed of sequential sub-actions is a *finite-state machine*. In one common approach, each state of the machine represents a *state* of the system (as defined by Miller and Johnson-Laird [104]) and the transitions correspond to actions. Alternatively, each state of the finite-state machine can correspond to an individual sub-action and the transitions correspond to the boundaries between actions. Notice that it is implicit in these definitions that any instance of a particular action is composed of a sequence of sub-actions, with no parallel threads of sub-actions or states.

Following the success of *Hidden Markov Models* (*HMMs*) in speech recognition, there has been a great deal of research in representing sub-action structure using these probabilistic finite-state machines (for example, in [182], and in an application for recognition of American Sign Language, [165]). An important feature of HMMs is that the transition probabilities can be learned by repetitive training. Examples, in the realm of visual recognition, are the works of Starner and Pentland [165], Wilson and Bobick [181], and Brand et al. [27]. Recently there has been some work with HMMs to represent human interaction [118], and human-machine interaction [70].

Intille and Bobick [63] have employed a combination of a frame-based language and Bayesian networks to represent and recognize American football plays. This is one of the few recognition methods that explicitly acknowledges the need for and investigates the issue of considering both context and movement in the recognition of an action. The most interesting feature of this work is that the large Bayesian networks are automatically assembled based on the description of the play, using a frame-based language that also includes some temporal representation.

Finally, context-free grammars have been used to represent actions with recursive sub-structure. In particular, Nagel has done extensive work in the representation of traffic actions using grammars [111]. Recently Bobick and Ivanov [22] proposed the use of probabilistic grammars in a formalism where training is still possible and that supercedes HMMs.

2.2.5 Representing Context

Efficient ways to represent the context of an action have eluded for many years the researchers in artificial intelligence. The difficulties are basically related to two issues: first, the *frame problem*, that is, what parts of a true state remain true in the next instant of time [101]; and second, *common-sense reasoning*, or the ability to extract the basic, "trivial" knowledge about the context by inference.

Even though reasoning about the state of the environment can be difficult, it is necessary that an action representation formalism have at least ways to specify them. Schank's *conceptualizations* [150] offer some convenient ways to represent the different elements of a scene, and basic inference has proved to be possible in specific domains [144]. Although there are many shortcomings and criticisms to Schank's ideas [180], we decided to adapt the idea of conceptualizations for the problem of action representation and recognition in an interactive space. In particular, as we will describe in the next chapters, we augmented the language with a mechanism to represent the temporal structure of the sub-actions based on Allen's interval algebra [3].

2.3 *Action Frames*

The choice of a representation depends not only on its intrinsic expressive capabilities but mostly on the intended use. Our initial objective is to have a computational mechanism in which high-level descriptions of the main actions in an interactive space could generate low-level, basic perceptual relations between objects and body parts that could simplify the working of a computer vision system. Later we expand our goals to design a good representation for visual recognition purposes that is also adequate for the scripting of interaction and to be a protocol for the communication between modules in an interactive space.

Our chosen method is an adaptation of Roger Schank's *conceptualizations* [150] that we call *action frames*. Our representation method, introduced in [130], is a simplification of the original idea where Schank's memory model is not implemented. Although there are many criticisms to Schank's work (for example, [180]), we find his treatment of verbs (especially verbs of action) extremely elegant and powerful. In fact, we believe that *conceptualizations* are more appropriate as an action representation scheme than as a general representation paradigm for natural language.

Action frames is a frame-based representation where each action is represented by a frame whose header is one of Schank's primitive actions:

- `propel:` the agent produces a movement in an object;
- `grasp:` an object is secured to the agent's "body";
- `move:` the agent moves one of its component parts;
- `ingest:` an object is moved to the inside of the agent;
- `expel:` an objects is removed from the inside of the agent;
- `speak:` the agent produces a sound, image, or any other non-material phenomenon;

- `attend:` one of the sensor systems of the agent is directed to an object;
- `ptrans:` an agent or object physically moves;
- `atrans:` an attribute of an object is transferred;
- `mtrans:` a mental concept is transferred from one agent to another;
- `mbuild:` a mental concept is built in the agent;
- `do-something:` the agent performs some unspecified action (usually towards an intended goal).

In the following subsections we clarify the meaning of the different primitives through examples. Before doing so, it is convenient to examine briefly other elements of *action frames*. Besides verbs, we can also represent states of agents or objects using the primitives `have`, and `attribute`.

In our implementation, each action frame begins with a primitive action or the primitive `have` (in the case of states) and contains different slots that supply the essential elements of the action. Undetermined symbols begin with a question mark (?); those symbols are defined only by the relationships they have with other objects. Some of the most commonly used slots are:

- `actor:` determines the performer of the action;
- `object:` contains the object of an action or the owner of an attribute;
- `result:` establishes the result of the action (a change in attributes or another action);
- `change:` indicates that an action produces a change in attributes of the object of the action or a change in some state of the world.
- `to, from:` determine the direction of a change
- `instrument:` describes another action or attribute that happens as part of the action.

The following subsections present examples to illustrate the use and meaning of the different action primitives and frame slots. For a more detailed discussion about how to represent generic actions, covering more complex cases, refer to Schank's description of his theory [149].

2.3.1 Representing the Script of a Cooking Show

To understand how an action is translated into an action frame, we will present a series of examples drawn from one of our projects in interactive spaces, the *Intelligent Studio*. The project is detailed in the last chapter of the thesis, consisting of a TV studio with automatic cameras that frame, upon request, specific elements of the scene (see also [133]).

Cooking-show scenario with a table on which there are bowls, ingredients, and different kitchen utensils. A microwave oven is in the back. Camera 1 is a centered camera, camera 2 is a left-sided camera, camera 3 is a camera mounted in the ceiling. Chef is behind the table, facing camera 1.

Chef, facing camera 1, greets the public.

"Welcome to the Vismod Cooking Show where we explore the wonderful world of research-oriented cuisine."

Chef turns to camera 2 and talks about today's recipe.

"Today's recipe is basil chicken. A delicious recipe with that special touch that only basil can give to a dish."

Chef turns back to camera 1 and mixes bread-crumbs, parsley, paprika, and basil in a bowl.

"Stir together 3 tablespoons of fine dry bread crumbs, 2 teaspoons snipped parsley, 1/4 teaspoon paprika, and 1/8 teaspoon dried basil, crushed. Set aside."

Chef wraps chicken with a plastic bag.

"Place one piece of chicken, boned side up, between two pieces of clear plastic wrap."

Chef puts the chicken on the chopping-board and shows how to pound the chicken.

"Working from the center to the edges, pound lightly with a meat mallet, forming a rectangle with this thickness. Be gentle, meat become as soft as you treat it."

Chef pounds the chicken with a meat-mallet.

Figure 2.1 Part of the script of a TV cooking show.

We have experimented the *Intelligent Studio* in the situation of shooting TV cooking shows. Throughout the rest of this chapter, we will examine our methods for representation and reasoning about actions using as example the segment of a cooking show whose script is depicted in fig. 2.1. The example is interesting not only because it has been implemented in a real system but also because cooking actions involve different types of transformation, and, in the TV show case, it also includes the communication of ideas.

Let us start by examining the simplest type of action in the script of fig. 2.1, when the chef turns his head towards a camera. Figure 2.2 shows the action frame corresponding to the action "*chef turns to side camera*". The action is characterized by the movement of the chef toward the "side" camera, what is translated into a `ptrans` primitive. The agent (`actor`) of the action is the `chef` himself as well as the object being physically moved. The result of the action is a change in the direction of the chef, as stated by the last three lines of the example. Notice that

"chef turns to side camera "

```
(ptrans
   (actor chef)
   (object chef)
   (result
      (change (object chef)
         (to (direction :side)))))
```

Figure 2.2 Action frame corresponding to the action of turning to a camera.

"chef mixes bread-crumbs, parsley, paprika, and basil in a bowl"

```
(do-something (actor chef)
   (result
      (change
         (object
            (group bread-crumbs parsley
               paprika basil))
         (to (contained bowl))
         (to (phys-cont
            (group bread-crumbs parsley
               paprika basil))))))
```

Figure 2.3 Action frame corresponding to the action of mixing ingredients in a bowl.

the representation of the action talks not only about the physical movement of the chef but also about the consequences of the action, that is, the final direction of the head of the chef.

2.3.2 Representing "Mixing Ingredients"

To describe the action *"chef mixes bread-crumbs, parsley, paprika, and basil in a bowl"* from the script of the cooking show, a different primitive action is used. In the mixing case, the way the mixture is accomplished is less important than the fact that ingredients are mixed. In fig. 2.3 we show a translation for mixing based solely on the results of the action. The primitive action is `do-something`, that is, there is no mention of what is actually done to achieve the intended result. The mixing action of fig. 2.3 uses the `group` header to cluster the different ingredients into a single object. Literally, the translation of the mixing action becomes *"chef does something that makes the group of ingredients (bread-crumbs, parsley, paprika, basil) to be contained in a bowl and in physical-contact to each other"*.

By now it should be clear that there is no unique representation for an action in action frames (as well as in Schank's conceptualizations). According to the focus, the inference needs, and the degree of detail, the same action can be represented in different forms. We choose to represent *mixing* only by its results and certainly some details are missing. For instance, we do

"chef talks about today's recipe (to side camera) "

```
(mtrans (actor chef)
   (to public)
   (object text2)

   (instrument
      (speak (actor chef)
         (object sound)
         (to (direction :side))
         (from (location
            (description (object mouth)
               (concept
               (have
                  (object mouth)
                  (attribute (part chef)))))))))
```

Figure 2.4 Action frame corresponding to the action of talking to a camera.

not specify how mixing is achieved: through a blender, by rotating a spoon in the mixture, etc. In other words, we did not specify the `instrument` of the action in this particular case.

2.3.3 Representing a "Talking" Action

Instrument actions are exemplified in fig. 2.4 that contains the description of the action *"chef talks about today's recipe (to side camera)"*. Talking involves two ideas, the first that there is a transfer of mental concepts from the agent to the listener and, secondly, that the communication happens through speech. In the representation shown in fig. 2.4 we employed a `mtrans` primitive to represent the communication act, which has as its `instrument` a `speak` action. Notice that `speak` is a general primitive for production of communication acts that encompass sound, gesture, mime, drawing, etc. Therefore it is necessary to typify the action by stating that its object is `sound`.

Figure 2.4 also displays a more elaborate structure — marked by the headers `description` and `concept` — that is used to include information about the elements in the scene. In this case, the information is that the sound is coming from the mouth of the chef. This is in fact a combination of statements: sound is coming from the object mouth, and mouth is part of the object chef. This is precisely the effect of the description structure: it allows linking the two concepts.

2.3.4 Representing a "Pounding Chicken" Action

To represent adequately a complex action such as *"chef pounds the chicken with a meat-mallet"* we have to resort to a combination of the structures defined above. In fig. 2.5 *"pounding"* is represented as an action where the chef propels a meat-mallet from a place which is not in contact with the chicken to a place which is in contact, and whose result is an

"chef pounds the chicken with a meat-mallet"

```
(propel (actor chef)
   (object meat-mallet)

   (from (location
      (description (object ?in-contact)
         (concept (have (object ?in-contact)
            (attribute (phys-cont chicken))))))

   (to (location
      (description (object ?not-in-contact)
         (concept (have (tense negation)
            (object ?not-in-contact)
            (attribute (phys-cont chicken)))))))

   (result
      (change
         (object
            (description (object chicken)
               (concept (have (object chicken)
                  (attribute
                     (phys-cont chopping-board))))))
         (from (flatness ?X))
         (to (flatness (greater ?X))))))
```

Figure 2.5 Action frame corresponding to the action of pounding a chicken with a meat-mallet.

increase in the flatness of the chicken on the chopping-board. To describe that the meat-mallet contacts the piece of chicken we employ an undetermined location, represented by `?in-contact`, which, although unspecified, has the attribute of being in contact with the chicken. Similarly, we define `?not-in-contact` to describe locations that are not in contact with the chicken.

The result of the propelling action is an increase in the `flatness` attribute of the object `chicken`. Using the `description` mechanism explained above, we also included the information that while this is happening, the chicken is on the chopping-board (through the less specific attribute of being in physical contact, `phys-cont`).

2.3.5 Representing a "Showing How to Do" Action

Our final example is the description of the *action "chef shows how to pound the chicken"*. The main element of the action is a `mtrans` primitive that corresponds to the transfer of the idea of `how-to-pound` to the `public`. The main action has two instrument actions: one corresponding to the talking and another that defines the gestures used to describe the way to pound. In particular, it is also stated that the gestures are performed by the chef's hands in the vicinity of the chopping-board.

"chef shows how to pound the chicken (on the chopping-board)"

```
(mtrans
   (time (group (begin 337) (end 380)))
   (actor chef)
   (object how-to-pound)
   (to public)

   (instrument (group
      (speak (actor chef)
         (object sound)
         (from (location
            (description (object mouth)
               (concept
               (have
                  (object mouth)
                  (attribute (part chef))))))
         (to (direction :center)))

      (speak (actor chef)
         (object gestures)
         (from (location
            (description (object hands)
               (concept
               (have
                  (object hands)
                  (group (attribute (part chef)
                   (proximity chopping-board))))))
         (to (direction :center)))))
```

Figure 2.6 Action frame corresponding to the action of explaining how to pound chicken with a meat-mallet.

The complete translation to action frames of the cooking show script depicted in fig. 2.1 can be found in appendix A. The listing is the actual input used in some of our experiments in the *Intelligent Studio* as described in chapter 7. It is also the representation used in the inference procedure detailed in the next section.

Currently we manually translate the sentences of the script into action frames statements. Building an automatic translator able to handle more generic scripts seems to be a natural language processing (NLP) problem and, as such, it is not a fundamental point in our research. Schank's team addressed the problem in the context of conceptualizations [150] and was able to build quite sophisticated NLP translators.

Notice that although Schank's conceptualizations provide powerful mechanisms to represent the basic elements and the consequences and context of an action, it is still quite a simplistic language for the description of the movements themselves, the objects, and the qualities of the objects. In many respects, Schank's formalism fails to accomplish with nouns and adjectives

the compactness and clarity of representation provided for verbs. Also, it is very limited in its ways to describe spatial and temporal attributes.

2.4 Reasoning Using Action Frames

A representation is as good as the information that can be obtained from it. In particular, we are interested in two capabilities in a representation for action. First, we want a representation that facilitates recognition and interaction control. This is the central theme of this thesis and, as we will see in the next chapters, the development of appropriate methods for the representation of the temporal structure of action and interaction plays a fundamental role in this respect.

The second capability we are seeking for a representation of actions is the possibility of automatic derivation of its component sub-actions and states. This comes from the observation that in many situations, it is possible or desirable to feed a computerized recognition system with high-level information that describes approximately the actions that are going to happen. An example is the case of TV shows where there is a script available, in textual form, prior to the actual recording of the show. Alternatively, it would be extremely helpful if surveillance systems could be instructed to detect new patterns of activity by a formal, high-level description of the actions to be recognized.

To accomplish that, it is necessary to investigate inference schemes that can take as input the action frames corresponding to the actions and derive the sub-actions that occur as part of that action. For instance, if the high-level description involves a physical move of a bowl, a system should infer that it is necessary first to grasp the bowl, and, in more detail, that grasping involves approaching, touching, and lifting the bowl. Ultimately, it would be quite desirable if the system could infer basic primitive perceptual properties that can be detect by the system. In the example above, if the recognition has a-priori knowledge about how to detect and track hands and bowls, it can try to automatically deduce a method to detect the hands approaching the bowl, getting close to it, and the hand movement that characterizes the lift. There has been some work in computer vision in this direction, notably in object recognition [163, 164], and navigation [21].

The current goals of our research are certainly much more modest. We have built an inference system, based in Rieger's work [144], that derives visual attributes from the action frames describing simple cooking-related actions. Typical inferences are related to the direction of the head of the actor, the amount of activity with the hands, and proximity relations among hands and utensils in the field of view. As described in chapter 7, such simple relations can be effectively used to control the application of pre-constructed visual routines in a real interactive space.

2.4.1 Inference using Action Frames

It is important to make clear that the inference system we developed is very simple and designed only to meet the demands of our particular domain. The system was designed to infer position and movement information about human bodies and hands, and physical contact and proximity among objects.

```
;;; propel implies grasping if propeled object is handleable

(rule
  (propel (actor ?actor) (object ?obj))
  (when (category ?obj 'handleable-object)
    (grasp
      (actor ?actor)
      (object ?obj)
      (to hands)
      (instrument
        (have hands
           (attribute (part-of ?actor)))))))
```

Figure 2.7 Example of a rule used by the inference system (with Lisp details omitted for clarity).

The system is based on Rieger's inference system for Schank's *conceptualizations* [144]. It outputs action frames representing sub-actions and collections of attributes about the humans and objects in the scene. At each moment of time, we consider just the actions known to be happening in that moment. In this implementation no mechanism was devised to include the information of what happened in the past.

The inference engine uses rules that construct new action frames from any action frame that matches the rule's pre-conditions and structure requirements. A typical rule is shown in fig. 2.7. The rule antecedent contains an action frame with wild-card fields (marked with "?"). The inference process attempts to match this antecedent to every action frame; upon success, the wild-card fields are instantiated and the consequent part of the rule is evaluated. In the example of fig. 2.7, a `propel` action causes the system to infer a `grasp` action only if the category of the propelled object is `handleable-object`, that is, the object is known to be small and handleable. In this case, the `actor` and `object` fields of the `grasp` action are instantiated accordingly. Moreover, it creates a field which describes the fact that the grasping occurs with the hands of `?actor`. This is specified by the inclusion of an instrument action stating that the symbol `hands` has the attribute of being `part-of` the actor.

The following is the list of all rules that we have implemented in our inference system:

(1) `result of change is phys-cont` ⇨ `ptrans`

(2) `ptrans of object` ⇨ `propel`

(3) `propel of small object` ⇨ `grasp`

(4) `propel of small object` ⇨ `move hands`

(5) `hands, mouth, body` ⇨ `part of actor`

(6) `propel, grasp` ⇨ `physical-contact`

(7) `certain attributes` ⇨ `symetric attribute`

(8) `physical-contact` ⇨ `proximity`

(9) `three-object proximity` ⇨ `transitive closure`

(10) `multiple ids for the same object` ⇨ `unification`

The first rule states that, if the result of an action is a change in the physical contact attribute of two objects, then there must be some physical movement involved. Rule 2 infers that the physical movement of an object involves propelling the object. Rule 3 is exactly the one depicted in fig. 2.7. Rule 4, similarly, determines that propelling (small objects) involves movement of the hands. Rule 5 simply determines that `hands`, `mouth`, and `body` are parts of the actor of an action frame. Rule 6 states that propelling or grasping involves physical contact between the object being moved and the body part of the actor involved in the action (typically the hands). Rule 7 takes some attributes (such as physical contact) between two elements of the scene and derives that the symmetric relation also exists. Rule 8 states that physical contact implies proximity. Rule 9 is, in fact, a procedure to compute the transitive closure of the objects that have proximity attributes (in certain cases). Finally, in the process of derivation it may happen that new identifications are generated by the inference system and rule 10 runs a procedure that unifies these ids.

Some of these rules are based on Rieger's inference system for *conceptualizations* [144]. However, unlike in his system, we do not iterate until all the possible inferences are done. In fact, we have observed in our experiments with two different, multi-action scripts that it was sufficient to apply the rules in the order given above, in a l-pass algorithm. This guarantees fast termination although it is only possible because the rules we used always infer action frames that are structurally simpler than the input. A similar observation, in a more elaborate inference system, is also made by Rieger [144].

2.4.2 Example of Inference: "Wrapping Chicken"

Below we list a manually commented printout of the action frames generated by our inference system using as its input the action frame of *"chef wraps chicken with a plastic bag"*. The action frame corresponding to this action is shown as item 0 of the list. In this case the system deduces that the chef's hands are close to the chicken. Only the relevant inferences are shown from the approximately 20 action frames that were actually generated.

0 :action frame obtained from the script

```
(do-something
   (actor chef)
   (result
      (change (object chicken)
         (to (group (contain plastic-bag
            (phys-contact plastic-bag))))))
```

1 : to put an object in physical contact with other (0) physical transportation is required (rule 1)

```
(ptrans
   (actor chef)
   (object chicken)
   (to (location plastic-bag)))
```

2 : to physically move (1) a small object, propelling is requited (rule 2)

```
(propel
   (actor chef) (object chicken)
   (to (location plastic-bag)))
```

3 : to propel a small object (2), grasping is required (rule 3)

```
(grasp
   (actor chef)
   (object chicken)
   (to hands))
```

4 : grasping (3) requires physical contact (rule 6)

```
(have
   (object hands)
   (attribute (phys-contact chicken)))
```

5 : physical-contact (4) implies proximity (rule 8)

```
(have
   (object hands)
   (attribute (proximity chicken)))
```

The reasoning process goes as follows: according to rule 1, a change that produces physical contact requires physical transportation of the object (in this case, the chicken); and physical transportation requires propelling (rule 2). From the propel action the system can infer grasping (rule 3), and from that it concludes that there is physical contact between the chicken and the actor's hands (rule 6); physical contact implies proximity (rule 8), and the system deduces that the hands are close to the chicken.

Notice that from a perceptual point of view, such inference gives an indication of where the piece of chicken can be found — near the hands! A typical use of such inference in the systems described later in this thesis would be in the case where the position of the hands is known (through a hand-tracker), and therefore, even if the chicken is not visible, the system still would be able to point to the approximate position of the chicken in the scene. Therefore, given knowledge about the action being performed (as movement plus context) it is possible to determine **from the context of the action** even the position of elements not immediately detectable.

2.4.3 Inferences in the "Pounding Chicken" Example

Let us now examine a more complex example where, from the action "*pounding the chicken with a meat-mallet*", the system obtains the fact that the hands are close to the chopping board. The following is commented printout of the action frames generated by our inference system,

using as input the action frame corresponding to the sentence *"chef pounds the chicken with a meat-mallet"*. Only the relevant inferences are shown from the 44 action frames actually generated. Notice that the transitive rule used for the inference of proximity (rule 9) is sensitive to the size of the objects, avoiding its use if one of the objects is larger than the others.

0 : action frame obtained from the script

```
(propel
   (time (group (begin 381) (end 400)))
   (aider (group back-and-forth fast strong))
   (actor chef)
   (object meat-mallet)
   (from (location
      (description (object ?in-contact)
         (concept (have (object ?in-contact)
            (attribute (phys-cont chicken)))))))
   (to (location
      (description (object ?not-in-contact)
         (concept (have (tense negation)
            (object ?not-in-contact)
            (attribute (phys-cont chicken)))))))
   (result
      (change (object
         (description (object chicken)
            (concept (have (object chicken)
               (attribute (phys-cont chopping-board))))))
         (from (flatness ?X))
         (to (flatness (greater ?X))))))
```

1 : propelling an object (0) requires grasping (rule 3)

```
(grasp
   (actor chef)
   (object meat-mallet)
   (to hands))
```

2 : grasping (1) requires physical-contact (rule 6)

```
(have
   (object hands)
   (attribute (phys-contact meat-mallet)))
```

3 : the end of propelling (0) implies physical contact (rule 6)

```
(have
   (object meat-mallet)
   (attribute (phys-cont ?in-contact)))
```

4 : physical-contact (0) implies proximity (rule 8)

```
(have
   (object ?in-contact)
   (attribute (proximity chicken)))
```

5 : physical-contact (3) implies proximity (rule 8)

```
(have
   (object meat-mallet)
   (attribute (proximity ?in-contact)))
```

6 : physical-contact (0) implies proximity (rule 8)

```
(have
   (object chicken)
   (attribute (proximity chopping-board)))
```

7 : physical-contact (2) implies proximity (rule 8)

```
(have
   (object hands)
   (attribute (proximity meat-mallet)))
```

8 : transitiveness of proximity, (4) and (5) (rule 9)

```
(have
   (object chicken)
   (attribute (proximity meat-mallet)))
```

9 : transitiveness of proximity, (6) and (8) (rule 9)

```
(have
   (object chopping-board)
   (attribute (proximity meat-mallet)))
```

10 : transitiveness of proximity, (7) and (9) (rule 9)

```
(have
   (object chopping-board)
   (attribute (proximity hands)))
```

The reasoning behind this inference is altogether similar to the previous example of *"wrapping chicken"*. The main difference is that, in this case, the transitive closure procedure is fundamental to determine the target information, that the hands of chef are close to the chopping-board.

2.4.4 Inferences in the Cooking Show

The second part of appendix A contains all the inferences made by our system for the action frames corresponding to the script of fig. 2.1. Each action and inference is associated with the interval of time when the action is happening or the attribute is known to be valid. Table 2.1 provides a summary of the process, describing how many of the different types of action frames were inferred for each action in the script.

action frame from script	actions				description			
	ptrans	propel	grasp	move	part-of	contain	phys-cont	proximity
greets the public					1			
turns to side camera		1		1				
talks about recipe					1			
turns to center camera		1		1				
mixes ingredients	1	1	1	1	1	1	6	9
wraps chicken with plastic-bag	1	1	1	1	1		6	12
shows how to pound chicken					1			3
pounds the chicken with mallet			1	1	1		11	30

Table 2.1 Distribution of the inferred actions and descriptions for each action in the script of fig. 2.1.

As mentioned above, the main objective of this section is to demonstrate that it is possible, through simple inference mechanisms, to infer action frames from other action frames. We also had the opportunity to test the inference system with a difference script, involving a completely new recipe and two chefs dialoguing to each other (the *Alan Alda cooking show*, as described in chapter 7). We were pleased to see that, with minimal adjustments and augmentations, we were able again to infer the needed visual attributes from the script of the new show (see chapter 7 for details).

2.5 Formalizing the Language: ACTSCRIPT

We are currently developing the idea of action frames further into a language, ACTSCRIPT, able to represent actions and to communicate requests, queries, and goals in a physical environment. Like action frames, ACTSCRIPT allows recursive decomposition of actions but it also includes the possibility of specification of complex temporal relationships (using the formalisms described in the next chapter).

The structure of ACTSCRIPT is similar to the action frames formalism described above. The complete syntax of the language is given in appendix B. There are only some minor differences with the syntax of action frames and the language was augmented to support the complete set of Schank's structures, including the ideas of *reason*, *enable*, and *enabled by*.

A major difference is the way the language was implemented. While in the case of action frames we employed straight LISP code, in ACTSCRIPT all the statements are composed of ASCII text strings. To recognize the language we designed and implemented an interpreter in C++ code that, given a string of ACTSCRIPT, returns a C++ object describing the syntactic structure of the ACTSCRIPT string. These C++ objects can be compared, partially matched, and searched with wildcards. An analysis of Rieger's implementation of an inference system for conceptualizations [144] reveals that those are the basic computational procedures needed to build inference mechanisms.

Until now we have not implemented an inference system for ACTSCRIPT. Instead we have used the language for the communication among the different computational modules in the interactive spaces we built. As we discuss in chapter 6, we employ in our systems an architecture that requires its high-level modules to exchange communication concerning actions, intentions, and requests, and to make and answer questions. To facilitate the communication of requests and answers, we have explicitly incorporated into the ACTSCRIPT language a header description for every string, containing the type of message, the sending module, and a serial number.

From this experience, we believe that ACTSCRIPT can become a protocol for high-level communication among modules and agents in an interactive, physical environment. The protocol is especially suitable for applications where the different computational components need to exchange information about the users, that is, the humans that are interacting with the machines.

Of course the appropriateness of text strings can be argued, for instance, by alternatively considering the use of JAVA objects and methods. In our view, the real issue is not the actual computational mechanism but the fact that a language that communicates what people or machines do or want to do should use representational mechanisms at least as expressive as Schank's conceptualizations.

2.6 Problem: Actions Have Complex Temporal Structures

From the examples in this chapter we can see that an action can be decomposed into sub-actions and that we can associate with the sub-actions states and attributes of the elements of the scene. Both the sub-actions and the states of the world or its agents can be seen as discrete units that are perceived to happen in definite intervals of time.

There is a tendency, especially among the visual recognition community, to assume that sub-actions occur sequentially. [26, 27, 35, 40, 43, 67, 118, 152, 165, 181]. This is also evidenced by the popularity of the use of *Hidden Markov Models* (*HMMs*) to represent gestures and movement for recognition purposes, which recognizes an action by looking for the most likely sequences of states or sub-actions given the sensory input.

We have observed that human action, in fact, does not follow clear sequential paths of units, and that parallel actions are very common. For instance, in the cooking show case, the action *"chef shows how to pound the chicken"* is clearly composed of two major parallel sub-actions: the chef is talking and at the same time gesturing and pointing to what he is going to do with the chicken. There are moments when the talking and the gesturing are synchronized, for instance, when the chef use expressions such as *"as you see here…"* coupled with pointing to a particular feature of the chicken.

Figure 2.8 Example of the temporal occurrence of the sub-actions and related states in an action of mixing ingredients in a bowl.

In some ways the performing of multiple actions (at the same level of granularity) in a human is restricted by the number of limbs and by limitations in the perceptual mechanisms. A drummer is a pathological example of parallelism at its peak. However, when we consider the agent plus its context, or interacting with other agents and machines, then the possibilities for parallel actions increase dramatically, making finite-state machines very unlikely candidates for representing interaction or multi-agent action. State machines are inadequate to represent multi-thread situations because the number of nodes increases exponentially as the number of parallel states or actions increases.

In our experience with the cooking show, we have found that parallel states that are only partially synchronized to the other states and sub-actions are very common. Figure 2.8 shows a diagram of the temporal occurrence of sub-actions and of attributes of objects in the action *"chef mixes bread-crumbs and basil in a bowl"* (chapter 4 details the meaning of the different sub-actions and states depicted in the picture).

For the basic sub-actions of *"mixing ingredients"* we have a quasi-sequential performance of the actions of *"picking up the bowl"*, *"getting breadcrumbs"*, *"getting basil"*, *"stirring the ingredients"* and *"putting the bowl down"*. However, when we consider the associated states of the objects in the scene, as shown in the bottom half of fig. 2.8, there is considerable overlap and a general lack of common boundaries. To represent such a structure in a state machine would require many different nodes just to define the multiple combinations for the occurrence of these states.

At the same time, there are some moments when states and sub-actions have their occurrence temporarily related, such as when the states are altered by the result of an action. For instance, at some moment after `physical-mix` starts, it becomes true that the ingredients become

mixed (`mixed-ingredients`). First, notice that the later can not occur before the former; and second, that the perception of ingredients starting to becoming mixed occurs at some undetermined point inside the sub-action `physical-mix`. This is the typical situation where we observe that, although the sub-action and the state are not sequentially linked, there are **temporal constraints** on the occurrence of one based on the occurrence of the other.

In the next chapters we will discuss methods to represent the temporal structure of the sub-actions and states related to an action that can cope with similar situations. Basically, we are looking for ways to augment Schank's conceptualizations so such complex, loose temporal constraints can be integrated into the representation of action. In particular, we propose a method that enables an action recognition system to effectively use the temporal constraints to enhance the recognition capabilities by discarding situations that contradict the temporal structure of the action.

2.7 Summary

This chapter introduced the concept of action as a combination of movement and context. As such, action recognition systems require mechanisms to represent both movement and contextual information. From the different proposals for action representation we recycled Roger Schank's *conceptualizations* [150] into a paradigm for action representation we call *action frames*. Among the good qualities of the paradigm, we highlight the use of a very small number of primitive actions, the expressiveness, and the possibility of inferring the internal structure of an action. Using a TV cooking show domain, we demonstrated that it is possible to infer the sub-actions that compose an action frame in specific situations by considering simple inference rules.

The action frame concept has been refined into a language called ACTSCRIPT that we have been employing mostly for communication of actions and intentions between control modules of interactive spaces. The complete syntax of ACTSCRIPT is given in appendix B.

Schank's original proposal of conceptualizations fails in providing adequate mechanisms to represent temporal relations between actions. In the last section of this chapter we argued that such temporal structures are in fact quite complex, involving multiple and parallel sub-actions that can not be easily represented by sequential paradigms such as finite-state machines (or HMMs). This is a key observation that motivates the search in the next chapters for more expressive temporal representations for actions and that constitutes the core of the theoretical contribution of this thesis.

3. Representing Temporal Structure Using PNF-Networks

The fundamental problem addressed in this chapter is how to represent the temporal structure of an action such that the representation can be effectively employed in action recognition and interaction control. In the previous chapter we discussed methodologies for representing the meaning of an action and how its component sub-actions can be derived by simple inference schemes. In this chapter we concentrate on how just the temporal structure of an action or interaction, if coupled with appropriate occurrence data of its sub-actions and sensor data, can provide enough information for the detection of its occurrence, or for the triggering of reactions in an interactive space.

During the occurrence of an action, each sub-action, event, and concept of the action occurs in a defined time interval. It is important to notice that there are natural temporal constraints among these intervals. For example, the time interval corresponding to drinking from a cup always follows the time interval corresponding to the action of grasping the cup (except for pathological drinkers). Therefore, we can represent the temporal structure of the occurrence of these sub-actions by a temporal constraint stated that one interval happens before the other. In section 3.1 we survey mechanisms for representing temporal relationships among actions, both in terms of expressiveness and efficiency of their reasoning algorithms.

In our work with action recognition and interaction control we have been employing *interval algebra networks*, or simply *IA-networks*, for the representation of temporal relations between actions. An IA-network is a constraint satisfaction network where the nodes correspond to variables to which time intervals can be assigned and the arcs correspond to binary temporal constraints between these intervals expressed using Allen's temporal relationships [4]. In section 3.2 we formally define IA-networks and analyze important aspects of these networks.

To compute the state of a variable corresponding to the occurrence of an action we develop an approach based on the idea of determining the *minimal domain* of nodes of the action's corresponding IA-network. The *minimal domain* of a node in an IA-network is the set of all the time intervals that belong to at least one solution. Given the intervals corresponding to the actual occurrence of some nodes in the network (typically, sensor data), we then compute the

minimal domain of all nodes of the network by constraint propagation. To determine if the action — or any of its component sub-actions — is happening, we check whether the current instant of time belongs to every interval in the minimal domain of a node. If this is true, then the corresponding action must be happening, since otherwise there would be a time interval in the minimal domain of the node that would not contain the current instant of time.

Unfortunately, temporal constraint propagation in interval algebra networks is *NP-hard* in most cases [177], and in particular, in the computation of the minimum domain [176]. In this thesis, we demonstrate that for control and recognition purposes it is not necessary to compute the minimal domain of an IA-network as a set of time intervals. Instead, it is sufficient just to determine, for each instant of time, whether its nodes are happening *now*, already happened (*past*), or has not happened (*future*) — the *PNF-state* of the node. The PNF-states of an IA-network can be more efficiently computed by projecting the IA-network into a specialized constraint network where the domain of each node is simply *{past, now, future}* — a *PNF-network*. This technique, a central result of this thesis, is explained in section 3.4.

However, computing the minimal domain in a finite constraint satisfaction network is also, in general, a *NP-complete* problem. In our experiments we have been employing an *arc-consistency* algorithm (based on [90]) to compute an approximation of the minimal domain of the PNF-network in time linearly proportional to the number of constraints. A fundamental objective of this chapter is to show that the result of the arc-consistency algorithm in PNF-networks is a conservative approximation to the minimal domain of the original IA-network. That is, a PNF-network after arc-consistency contains the mapping of all solutions of the original IA-network. Moreover, if the PNF-network has no solutions, then the IA-network does not have solutions either. Beyond that, our experience with this method tells that in most practical cases the fast arc-consistency algorithm actually computes the correct PNF-state of the nodes.

The last section of this chapter introduces *PNF propagation*, a technique by which information about the occurrence of actions in previous instants of time can be consistently used to enhance the detection of other actions in the current moment of time. At the heart of PNF propagation is the representation of the (obvious) fact that after an action has happened it can not happen again. Using this technique, we can compute consecutive PNF-states of nodes in a given PNF-network.

We introduced the concept of PNF propagation in [132], and the first results appeared in [134] in a different formulation than the one presented in this chapter. We later refined the original formulation into the constraint satisfaction framework [135]. The formal definition of PNF networks based on *now* intervals as presented in this chapter is completely new.

This chapter is essentially theoretical and its main goal is to formalize the PNF propagation methodology and to prove that it is mathematically sound. We defer to the next chapter the issue of how the temporal formalisms studied here can be applied in real action recognition and interaction problems.

3.1 Reasoning with Temporal Constraints

From databases to robotics, numerous problems in computer science involve time. This has generated a variety of research that considers different assumptions about how temporal information is obtained and about what kind of reasoning is needed. A good introduction to the different methods was written by Allen [5].

In this section we review the main approaches for temporal representation and reasoning and comment on their strengths and shortcomings. Since the work presented here deals primarily with constraint-based temporal representations, we study those in greater detail and provide a short review of constraint propagation.

To start, it is necessary to establish some terminology. An *event* is an instant of time with no duration. A *time interval* is a continuous segment of time that has duration. Given an interval, the earliest instant of time that belongs to the interval is its *begin point*, and the latest is its *end point* (although some models, as we will see below, do not require the explicit reference to the begin and end points of an interval).

But what is a good representation for time? First, as noted above, it depends on the specifics of the situation being modeled (e.g., a database or a control system). However, there are properties that can be generally considered important on temporal representation schemes. The following is a list (based on [3]) of good characteristics of time representations:

- A temporal representation should allow imprecise knowledge since typically absolute time information is not available. For example, although we know that grasping must precede drinking from the cup, it is hard to determine in advance how long the act of grasping takes.
- A representation must accept uncertain information; for instance, a system might know only that an action *A* happens before **or** during another action *B*. For instance, we consults maps before and/or while driving to an unknown place.
- Time representations should provide facilities for reasoning about persistence, that is, that a state often continues to be true after its cause ceases to happen. When a cup is put on a table, the state corresponding to the cup being on the table remains "happening" until some other action affects the cup or the table.
- A temporal representation should allow the expression of mutual exclusion, for instance, when two actions or states *I* and *J* can not happen at the same time. A person can not read a book while sleeping, or driving, or taking a shower.

Allen, in [3], discusses these issues in more detail. We would like to stress here that in situations of action recognition, the ability to represent mutual exclusion is very useful. Although this kind of negative knowledge can not contribute to the recognition of a particular action, it is a powerful tool to eliminate competing alternatives. Although representing mutual exclusion involves a considerable increase in computational complexity, as discussed below, we found that such negative information decisively increased the recognition power of our systems, and, therefore, was worth the extra cost. Moreover, when describing an action we

realized in many situations that it is easier to state clearly what can not happen than what must happen.

3.1.1 Formalisms to Represent and Reason about Time

One of the first approaches to temporal representation was the use of state-spaces as proposed by Sacerdoti [148] and Fikes and Nilsson [50], where actions were modeled as functions mapping between states that represent the situation of the world at some instant of time. In these schemes, the duration of a transition between states corresponds to the duration of an action, and the amount of time in the same state is associated to the duration of that state. Both schemes could hardly represent uncertain and mutual exclusion knowledge, although they could provide a notion of persistence.

Dating schemes (for example in the works of Hendrix [60] and Kahn and Gorry [75]) are characterized by simple, fast ways to compute the ordering information between two *dates* which represent known or unknown points in time. However, these representations require that all events have assigned dates to them and all durations to be known, failing to comply with our requisite of handling uncertain and imprecise information. As noted by Allen in [5], such dating schemes also fail to represent partial ordering and simultaneous events.

The representation of duration is the main component of techniques based on network representation such as PERT networks (allowing simple dynamic programming methods). A more expressive formalism was proposed by Dean and McDermott [45] where relations between events are represented as boundaries on the duration between them. If a symbol for infinity (∞) is introduced, it is possible to represent qualitative information. However, reasoning is based on graph search and, unless a great deal of information is known in advance, the combinatorics of the approach is just too large.

Temporal logic is the foundation of McCarthy's situation calculus [101] where knowledge is represented as a series of situations describing what is true at every instant of time. A major shortcoming of this theory is the difficulty in handling multiple events occurring at the same instant of time. A point-based logic formalism was further developed by Shoham in [156].

The next subsection details the most popular formalisms for temporal representation that are based on constraint propagation. Before that, we should mention that probabilistic methods are very popular in sensor-based applications like computer vision and speech processing. Among them, *hidden Markov models (HMMs)* are by far the most popular (see [140] for a good introduction). HMMs consist basically in a probabilistic extension of the state space model described above. The major advantage is the possibility of fast learning and recognition. The major shortcomings are the difficulties of representing duration, parallel events, and negative knowledge.

3.1.2 Temporal Constraint Networks

The most popular time representation methods in the AI community employ constraints to encode the temporal information. Typically, the temporal knowledge is encoded as a set of variables, domains on those variables, and constraints about the values that the variables can

assume. A *constraint network* is simply a network where the nodes are variables, the domains of the variables are unary constraints on the nodes, and the arcs correspond to the constraints between the variables. Although arcs normally connect only two nodes, if there are constraints involving more than two nodes, special, multi-ended arcs are employed. Original work on temporal constraint networks dates back to the research performed by Montanari [108] and Mackworth [90]. Good introductions to algorithms for constraint networks are provided by Nadel in [110] and by Kumar in [80]; a good analysis of networks with finite domains is done by Dechter in [46].

Dechter, in [47], gives a good general overview of the use of constraint networks for the representation of time. Beginning in 1983, Malik and Binford [96] used simple ordering constraints between nodes representing events in time. To solve the constraint problem, they used linear programming techniques to solve systems of linear inequalities. Besides problems with temporal representation similar to those of point-algebras as discussed below, the approach also suffers from the need for full re-computation whenever a constraint is added.

In the so called *point-algebras*, originally proposed by Vilain and Kautz [177] and later revised in [178], the nodes of the constraint network represent **events** in time, and the constraints are typically subsets of the set $\{<,>,=\}$. If an arc between nodes *a* and *b* is labeled $\{<,=\}$, this signifies that either $a < b$ or $a = b$. There are some problems with this approach as discussed by Allen in [3]. First, it becomes very hard to represent intervals, in particular because of the problem of deciding if time intervals are open or close at their boundaries (see [3], pg. 834).

But, most important, it is not possible to represent that two intervals can not happen at the same time, i.e., mutual exclusion. To see this, suppose we have two intervals: *A*, with begin point a_1 and end point a_2; and *B*, delimited by b_1 and b_2. To say that A and B occur in different times is equivalent to saying that $a_2 < b_1$ or $b_2 < a_1$. In either case, these are constraints between 4 nodes that can not be decomposed into binary constraints, and therefore are beyond the representation power of point-algebras.

If we restrict the expressiveness further and disallow the use of the “different” relation between nodes ($\neq$, represent by $\{<,>\}$), point-algebras allow very fast constraint propagation algorithms based on the fact that in those cases there is a partial order in the network. Gerevini and Schubert [56] discuss heuristics methods to efficiently propagate constraints in networks with the “different” ($\neq$) relation.

The competing formalism, called *interval algebra* [3, 4], differs radically by assuming that nodes in the constraint network represent **time intervals** and the arcs are temporal constraints between them. The next section formalizes such algebras and, in the following discussion, most of the important properties are examined. We would like to point out now that, as formalized by Allen in [7], interval algebras do not suffer from the problem of endpoints mentioned above and also allow the representation of negative information such as that two intervals do not happen at the same time. The latter is the main reason for our interest in employing interval algebras in our work.

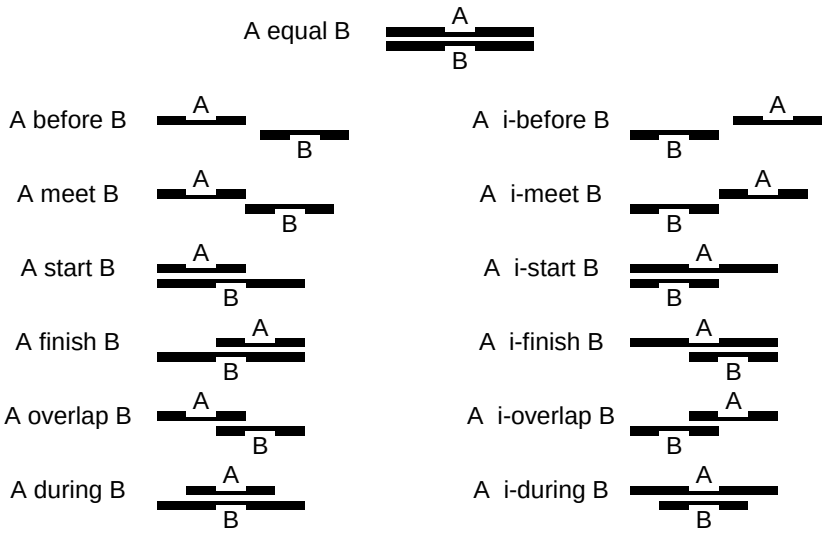


Figure 3.1 The 13 possible primitive relationships between two time intervals.

However, most constraint propagation algorithms in interval algebras are exponential in the number of constraints, although partial solutions are adequate for some practical applications (see some empirical studies by Ladkin and Reinefeld [82] and Baptiste and Pape [12]). Generalizations of interval algebras have been proposed by Freksa [51]. Also, Meiri [103] and van Beek [176] suggested ways to combine point- and interval algebras. For the scope of problems and applications considered in this thesis, we consider that interval algebras provide a good compromise of expressiveness and efficiency if improved by the PNF algorithms described below.

3.2 IA-Networks

Time intervals are the fundamental concept of the *interval algebra* proposed by James Allen [3]. Unlike in point-algebras [177], the intervals are not defined by beginning and ending points but instead they are a primitive concept themselves. Relationships and temporal constraints are defined in terms of disjunctions of the 13 possible primitive relationships between two time intervals: the relations equal *(e)*, before *(b)*, meet *(m)*, overlap *(o)*, during *(d)*, start *(s)*, finish *(f)*, and their inverses, *ib*, *im*, *io*, *id*, *is*, and *if* respectively. Figure 3.1 (from [3]) provides an intuitive understanding of the meaning of those relationships.

We denote by Λ the set of all primitive relationships, and by $\overline{\Lambda}$ the set of all possible disjunctions of primitive relationships, seen here as subsets of Λ; notice that the number of elements of $\overline{\Lambda}$ is,

$$\left\|\overline{\Lambda}\right\| = 2^{\|\Lambda\|} = 2^{13} = 8192$$

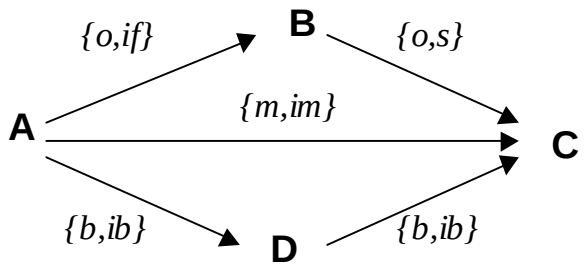


Figure 3.2 An example of an IA-network.

An *interval algebra network X*, or simply an *IA-network* (also called by many authors as an *Allen interval network*), is a binary constraint satisfaction network (as defined, for instance, by Kumar in [80]) where the nodes $x_1, x_2, \ldots, x_n$ correspond to variables to which one time interval can be assigned. To each node x_i there is associated a set of intervals, called the *domain* of the node x_i (also referred as the *unary constraints* of the network), that specifies the time intervals that can be assigned to the nodes. The arcs of the network correspond to binary temporal constraints between the two nodes expressed as disjunctions of Allen's temporal relationships, that is, as elements of $\overline{\Lambda}$.

Figure 3.2 shows an example of an IA-network composed of four nodes, *A*, *B*, *C*, and *D*, and five temporal constraints. The temporal constraint between *A* and *B*, *{o,if}*, determines that the interval *A* either overlaps or is finished by interval *B*. Similarly, *B* overlaps *(o)* or starts *(s)* interval *C*. The relations *{b,ib}* between *A* and *D* and between *D* and *C* make the corresponding intervals mutually exclusive but do not constraint them to occur in a pre-determined order. The relation between *A* and *C* is similar, but the *{m,im}* relation forces one interval to be immediately followed by the other.

Following the convention adopted by most of the researchers in the field, fig. 3.2 does not show the implied inverse relationships as well as the "unconstrained" relations — the disjunction of all primitive relationships — that exist in our example, for instance, between *B* and *D*.

3.2.1 Solutions of an IA-Network

Given a set of unary constraints between nodes $\sigma_1, \sigma_2, \ldots, \sigma_n$, a *solution* of the IA-network *X* is an assignment of time intervals to all the nodes $x_1, x_2, \ldots, x_n$ of *X* which satisfies both the binary constraints between any two nodes and the unary constraints $\sigma_1, \sigma_2, \ldots, \sigma_n$ on each node. If at least one solution exists the IA-network is said to be *consistent under* $\sigma_1, \sigma_2, \ldots, \sigma_n$, or simply, *consistent*.

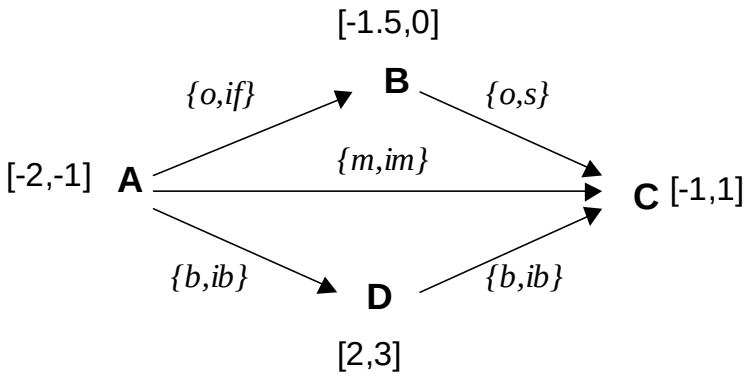


Figure 3.3 A solution for the IA-network of fig. 3.2.

In the examples shown in this chapter we will assume a model for the IA-algebra where intervals are closed continuous segments of the real line $\Re$. That is, an interval in our examples is a pair $[a,b] \in \Re \times \Re$ where $a < b$. The zero point has no special meaning in our model, so intervals can be defined with negative values. As pointed by Allen [5], there are other models for the interval algebra; the choice of segments on the real line is arbitrary and is done for the sake of simplicity of exposition.

Figure 3.3 shows a solution of the IA-network of fig. 3.2 in the case that there are no unary constraints on the nodes. It is easy to see that the assignment of each of the displayed intervals to the nodes satisfies the constraints between each pair of nodes. Since there is a solution, according to the definition above, the network is consistent. Notice that, for a different set of unary constraints, the same network might not be consistent. For instance, consider the case where the unary constraints on the domains of *A* and *C* are, respectively, $\sigma_A = \{[-\infty,-1]\}$ and $\sigma_C = \{[1,+\infty]\}$. Since the constraint *{m,im}* requires any solution to have no gaps between the intervals corresponding to *A* and *C*, there is clearly no solution to the network and therefore under those constraints the network is not consistent.

3.2.2 Transitive Closure of IA-Networks

Given a constraint between two intervals, that is, a disjunction of primitive temporal relationships, the more elements it has the less restrictive is the constraint. For example, a constraint enumerating all the primitive relationships does not impose any constraint on the corresponding intervals.

However, given an IA-network, there can be constraints where some of the primitive relationships are superfluous, that is, all solutions in all nodes are preserved if those primitive relationships are removed from the constraints. For example, a more closer examination of fig. 3.2 reveals that the relation *im* between *A* and *C* is superfluous. Since *A* finishes before or at the same time as *B*, and *C* starts after the beginning of *B*, then *A* can not happen after *C*. That

Table 3.1 The transitive closure *tc(r,s)* for the primitive relations between two intervals (from [3]).

tc (r,s)	*e*	*b*	*ib*	*d*	*id*	*o*	*io*	*m*	*im*	*s*	*is*	*f*	*if*
e	*e*	*b*	*ib*	*d*	*id*	*o*	*io*	*m*	*im*	*s*	*is*	*f*	*if*
b	*b*	*b*	*<all>*	*b o m d s*	*b*	*b*	*b o m d s*	*b*	*b o m d s*	*b*	*b*	*b o m d s*	*b*
ib	*ib*	*<all>*	*ib*	*ib io im d f*	*ib*	*ib io im d f*	*ib*	*ib io im d f*	*ib*	*ib io im d f*	*ib*	*ib*	*ib*
d	*d*	*b*	*ib*	*d*	*<all>*	*b o m d s*	*ib io im d f*	*b*	*ib*	*d*	*ib io im d f*	*d*	*b o m d s*
id	*id*	*b o m id if*	*ib io id im is*	*e o io s d f is id if*	*id*	*o id if*	*io id is*	*o id if*	*io id is*	*id if o*	*id*	*id is io*	*id*
o	*o*	*b*	*ib io id im is*	*o d s*	*b o m id if*	*b o m*	*e o io s d f is id if*	*b*	*io id is*	*o*	*id if o*	*d s o*	*b o m*
io	*io*	*b o m id if*	*ib*	*io d f*	*ib io im id is*	*e o io s d f is id if*	*ib io im*	*o id if*	*ib*	*io d f*	*io ib im*	*io*	*io id is*
m	*m*	*b*	*ib io id im is*	*o d s*	*b*	*b*	*o d s*	*b*	*e f if*	*m*	*m*	*d s o*	*b*
im	*im*	*b o m id if*	*ib*	*io d f*	*ib*	*io d f*	*ib*	*e s is*	*ib*	*d f io*	*ib*	*im*	*im*
s	*s*	*b*	*ib*	*d*	*b o m id if*	*b o m*	*io d f*	*b*	*im*	*s*	*e s si*	*d*	*b m o*
is	*is*	*b o m id if*	*ib*	*io d f*	*id*	*o id if*	*io*	*o id if*	*im*	*e s is*	*is*	*io*	*id*
f	*f*	*b*	*ib*	*d*	*ib io im id is*	*o d s*	*ib io im*	*m*	*ib*	*d*	*ib io im*	*f*	*e f if*
if	*if*	*b*	*ib io id im is*	*o d s*	*id*	*o*	*io id is*	*m*	*is io id*	*o*	*id*	*e f if*	*if*

is, the relation *im* between *A* and *C* can be eliminated from the constraint without the loss of any solution (in any model).

A major feature of the interval algebra is the possibility of elimination of those superfluous relationships between variables by simple constraint propagation. The method proposed by Allen [3] is based on the concept of *transitive closure* of pairs of primitive relations *(r, s)* of Λ. Given three variables, *A*, *B*, and *C* such as *A r B* and *B s C*, the set of all possible relations between *A* and *C* is denoted by $tc(r,s): \Lambda \times \Lambda \rightarrow \overline{\Lambda}$. That is, any relation that does not belong to

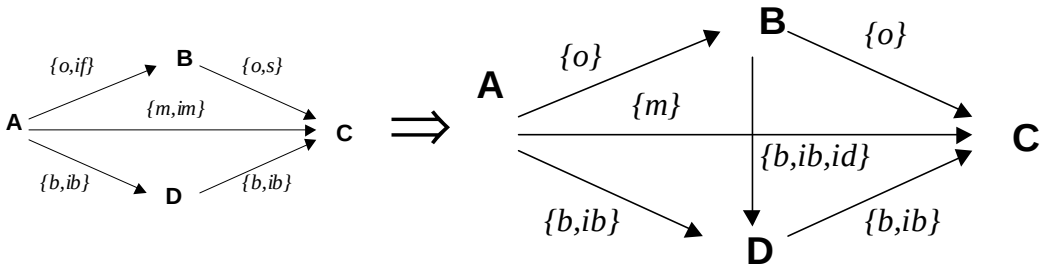


Figure 3.4 The IA-network of fig. 3.2 after Allen's path-consistency algorithm. The resulting network still contains the superfluous relationship *id* between nodes *A* and *C*.

$tc(r,s)$ is guaranteed to be superfluous and can be removed from an IA-network without the loss of solutions.

Table 3.1 contains the values of *tc (r,s)* for all primitive relations *r* and *s*, according to the semantics model proposed by Allen [3]. To compute the transitive closure of a generic pair of relations $(R,S) \in \overline{\Lambda} \times \overline{\Lambda}$, denoted by *TC (R,S)*, we simply take the union of the transitive closure of all pairs of its constituent primitive relations,

$$TC\ (R,S) = \bigcup_{\substack{r \in R \\ s \in S}} tc(r,s)$$

3.2.3 Path-Consistency and Global Consistency

Table 3.1 is used by Allen in an algorithm to infer stronger temporal constraints between the intervals of an IA-network. The algorithm (described in detail in [178]) basically reduces the set of disjunctive relations between two nodes *A* and *B* by computing its intersection with the transitive closure, for every node *I*, of the constraints between *A* and *I* and *I* and *B*. Constraints are checked whenever a neighboring node has one of its constraints modified and until there is no more changes in the IA-network. It is a classical *path-consistency* algorithm (according to Montanari's classification [108]) which assures only that, for every sub-network of three nodes *A*, *B*, and *C*, the set of primitive relations representing the temporal constraint between *A* and *C*, R_{AC} is **contained** in the actual transitive closure of the constraints between *A* and *B*, R_{AB} and *B* and *C*, R_{BC}:

$$R_{AC} \subseteq TC\ (R_{AB}, R_{BC})$$

Figure 3.4 shows the IA-network of fig. 3.2 after the use of Allen's path-consistency algorithm. Notice that path-consistency tightened some of the temporal constraints of the network. The constraints between *A* and *B* and *B* and *C* were reduced to *{o}*. In particular, path-consistency removed the superfluous *im* relation mentioned above from the constraint between *A* and *C*. Also, the algorithm reduced the constraint between nodes *B* and *D* from the set of all possible

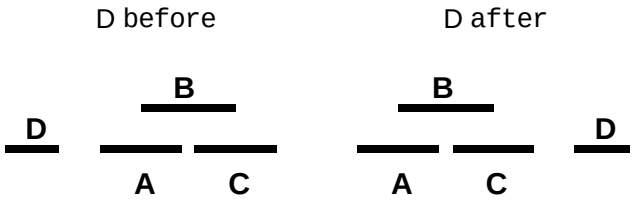


Figure 3.5. The two possible situations for the IA-network of fig. 3.4.

relations (which, following the conventions, is not represented in the figure) to the constraint *{b,ib,id}*.

Allen's algorithm does not remove all the superfluous temporal relations from an IA-network, as noticed by Allen himself and other researchers [3, 177]. This can also be verified by our example in fig. 3.4. As show in fig 3.5, the interval *D* must occur either before or after the group of *A*, *B*, and *C*, and therefore, the relation *id* between *B* and *D* corresponds to an impossible case. Allen's algorithm fails to remove this inconsistency because this case involves constraints among four nodes, and, as explained above, path-consistency assures consistency only for sets of three nodes.

Vilain, Kautz, and van Beek [178] show that, in fact, removing all the superfluous relations of an IA network is an *NP-complete* problem. However, Allen's algorithm has been used by many researchers in practical applications because of its speed and also because it seems to provide a good approximation of the transitive closure in most cases. For instance, in [3] James Allen argues that *"... we have not encountered problems from this deficiency in our applications of the model."* (from [3], pg. 837). Similarly, in our experience with IA-networks, we could never detect a situation where the approximation computed the Allen's path-consistency algorithm was insufficient.

3.2.4 The Minimal Domain of IA-Networks

IA-networks are normally used in tasks involving planning or scheduling. Typically, unary constraints are imposed on the nodes, for example, constraining the duration of some of the intervals, or requiring that some of the intervals to be contained in pre-determined intervals (see [103] for an account of different methods).

Given a node x_i, we define a *feasible value* for x_i to be a time interval that belongs to at least one solution of *X*. The set of all feasible values of a node x_i is called the *minimal domain* of x_i. The minimal domain of an IA-network with unary constraints $\sigma_1,\sigma_2,\ldots,\sigma_n$ is the collection of the minimal domains of all nodes, denoted in this chapter by $\overline{\sigma_1},\overline{\sigma_2},\ldots,\overline{\sigma_n}$. Notice that, by definition, $\overline{\sigma_i} \subseteq \sigma_i$

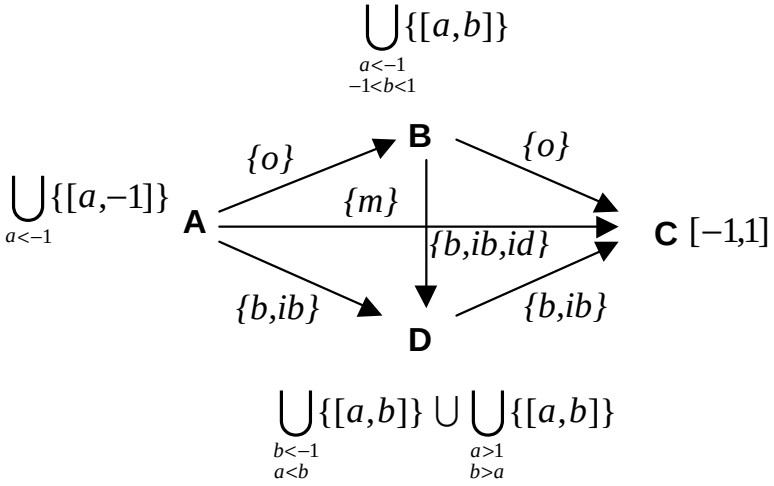


Figure 3.6 The minimal domain of the IA-network of fig. 3.4 given that the interval C happens in the interval [-1,1].

The minimal domain of a network X should not be confused with the concept of *minimal network* (see [47] for a clear treatment of the difference between the two), and explained in the next section. The first is applicable to every constraint network and concerns feasible values of nodes; the second is a concept specific to binary networks and refers to the construction of a network where all the constraints satisfy a consistency property.

As it will become clear in the future chapters, our applications do not require the computation of specific solutions, but instead require the determination of the feasible values of intervals in an IA-network. Therefore, our analysis concentrates on issues related to the problem of **determining the minimal domain of the nodes of a IA-network given an initial set of unary constraints on its nodes**. In the case of IA-networks, both the minimal domain and the unary constraints are represented by sets of time intervals.

In general, it is *NP-hard* to determine if there are *solutions* in a IA-network [103]. The computation of the minimal domain in IA-networks is also *NP-hard* in the number of constraints [177], preventing their use in most practical problems. The combinatorial explosion does not come only from the handling of the constraints of the networks but in fact mostly from the operations on the sets of time intervals associated to each variable. In particular, mutual exclusive constraints can yield minimal domains that are composed of disjunct sets of intervals, making difficult any compact representation of sets of intervals.

This is shown in fig. 3.6 that displays the minimal domain of the network of fig. 3.1 when it is known that interval C is $[-1,1]$ and no other unary constraints in the other nodes. Notice that it is necessary to describe the minimal domain of variables as infinite sets of intervals with restriction in their beginning and ending values. Moreover, the mutually exclusive constraint between C and D generates a set that is a union of two sets of intervals without common

elements. Notice that if the minimal domain of *C* was described by a set of two intervals without intersection, than the minimal domain of *D* would have to be described by four sets of intervals. In fact, *D* is a typical case of the combinatorial explosion of the computation of the minimal domain in IA-networks (see [177] for a complete analysis of the issue). Such problems do not occur if the IA-network does not contain mutually exclusive relations because in such cases there is a topological order in the network and therefore dynamic programming techniques can determine solutions in polynomial time.

Even the computation of simple arc-consistency [90], a conservative approximation to the minimal domain, can become exponentially long and/or large in IA-networks. Hyvönen examines this problem in general [62] and suggests some optimizations that improve the performance. The lack of fast methods to compute the minimal domain motivated us to search for a simplification of IA-networks where an approximation of the minimal domain can be computed in polynomial time but still produces the information required by our applications.

3.3 Past, Now, and Future

In the context of the applications described in the next chapters, it is sufficient to determine only whether the action corresponding to a node of an IA-network is happening or not. In particular, we found that it was not necessary for recognition and, to a lesser extent, for interaction control to know precisely the time when an action will happen in the future or how long the action is going to take. This motivated us to narrow our search for the minimal domain so that it provides exclusively this kind of answer.

We can represent those situations by three symbols, *past*, *now*, and *future*, where intuitively *past* stands for an action that happened before the current instant of time; *now* refers to actions that are happening; and *future* corresponds to the actions which have not started yet. Given a node of an IA-network in a given instant of time, and information about the occurrence of some nodes (typically, sensor data), we want to assign one of these three symbols to identify if the action corresponding to that node has happened or not.

The purpose of this section is to define a formal method to compute *past*, *now*, or *future*, such that the result is compatible with the semantics of an IA-network. In particular, if the current instant of time belongs to every interval of the minimal domain of an IA-network, we want the *now* symbol to be assigned to that node, representing that the interval must be happening.

Before going in this direction, we should first examine the question of why do we use three states and not two, that is, by representing only happening (on) and not happening (off)? Temporal constraint propagation is polynomial in 2-valued constraint networks [46] and, in particular, for the computational of the minimal domain. However, the problem is that in on/off networks it is impossible to distinguish past from future: off stands both for intervals that happened and those that will happen. In fact, some results in the next chapter show that differentiating between past and future occurrences in a time line dramatically enhances the recognition of actions and the detection of contradictory states.

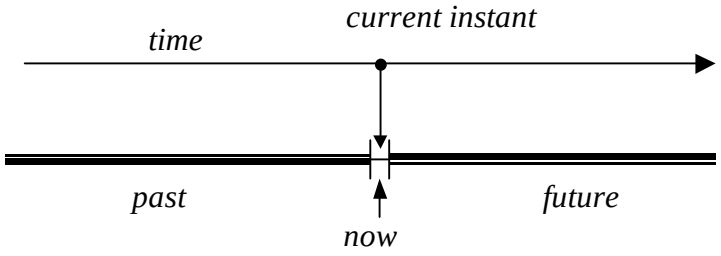


Figure 3.7 Diagram of the relations between the intervals *past*, *now*, and *future*.

3.3.1 The PNF-state of an IA-network

The fundamental concept behind our methods is the division of the whole timeline into three consecutive intervals, which, for simplicity of notation, are called respectively *past*, *now*, and *future*. In this formulation, *past* is an interval that starts in $-\infty$ and ends just before *now*, that is *past {m} now*. Similarly, *now* finishes exactly when future starts, *now {m} future*, and *future* continues to $+\infty$. Notice that, as show in fig. 3.7, *now* is not an instant in time, but an interval; in practice, the current instant of time will be contained in this interval.

We denote by m the set of these three intervals, $m = \{past, now, future\}$. To simplify notation, let us also define the set M of subsets of m,

$$M = \{\varnothing, \{past\}, \{now\}, \{future\}, \{past, now\}, \{now, future\}, \{past, future\}, \{past, now, future\}\}$$

whose elements are abbreviated as

$$M = \{\varnothing, P, N, F, PN, NF, PF, PNF\}$$

The idea of a *now* interval is related to the method proposed by Allen in [3] to control the memory usage in IA-networks. However, our goal here is to use a conceptually similar technique directly to obtain information about occurrence of intervals, while in Allen's case the interval **NOW** is basically used only as a reference point to control propagation. Moreover, our research has shown that the concept can be extended to define a new class of networks (defined in section 3.4) where constraint propagation can be computed much faster.

As a fundamental part of our theoretical framework, the interval *now* must have the property that **no other interval starts or finishes inside** *now*. That is, given any interval *I*, it never happens that *now* overlaps *I (now o I)*, *now* is overlapped by *I (now io I)*, *now* is started by *I (now is I)*, *now* is finished by *I (now if I)*, or *now* is i-during *I (now id I)*, as depicted in fig. 3.8.

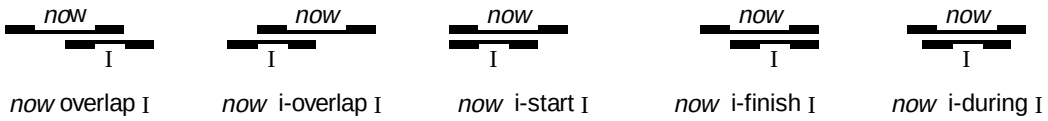


Figure 3.8 Relations that never happen between the interval *now* and any other interval I.

In the application of this model, *now* always contains the current instant of time and therefore the assumption of its existence is related to assuring that sampling rates of the other intervals' changes are high enough to guarantee the fundamental property.

For a given interval of time, the above conditions mean that either *now* is contained in the interval or the interval is contained in *past* or in *future*. Formally, given an interval $I \in \Re \times \Re$, we define the *PNF-state of I* by the function $s(I): \Re \times \Re \to M$ where

$$\begin{aligned} s(I) = P \quad &\Leftrightarrow \quad I \ \{d, f\} \ past \\ s(I) = N \quad &\Leftrightarrow \quad now \ \{e, s, d, f\} \ I \\ s(I) = F \quad &\Leftrightarrow \quad I \ \{d, s\} \ future \end{aligned}$$

Applying Allen's transitive closure, we can equivalently define the function *s* by

$$\begin{aligned} s(I) = P \quad &\Leftrightarrow \quad I \ \{b, m\} \ now \\ s(I) = N \quad &\Leftrightarrow \quad now \ \{e, s, d, f\} \ I \\ s(I) = F \quad &\Leftrightarrow \quad I \ \{ib, im\} \ now \end{aligned}$$

Considering that by our definition of *now* it never happens, for any interval *I*, that $now \ \{o, io, d, is, if\} \ I$, it is clear that the function $s(I)$ is well defined — since it exhausts all the remaining possibilities. Notice that, although the range of the function *s* is *M*, its image is restricted to $\{P, N, F\}$.

We can extend *s* to sets of intervals by considering the union of the PNF value for each interval of the set. We then define the *PNF-state of a set of intervals* Σ by the function $S(\Sigma): 2^{\Re \times \Re} \to M$ where

$$S(\Sigma) = \bigcup_{I \in \Sigma} s(I)$$

3.3.2 Computing the PNF-state of an IA-network

Using those definitions, we are ready to define a method to determine whether an action has happened or not, as explained in the beginning of this section, given the intervals corresponding to the occurrence of some nodes in the network (typically, sensor data). First, we have to define a procedure to compute the PNF-state of the nodes of an IA-network given the intervals corresponding to the occurrence of some of the nodes of the network.

Procedure 3.1

i. define *now* as a very narrow interval containing the current instant of time;

ii. obtain all the known intervals that correspond to the occurrence of some of the nodes in the IA-network;

iii. compute the minimal domain of the IA-network using as unary constraints the known intervals;

iv. project the computed minimal domains through *S*, computing therefore the PNF-states of the minimal domain of all nodes.

Given the PNF-state of the nodes of the IA-network constructed by procedure 3.1, it is clear that an interval is happening only if its PNF-state is $N = \{now\}$. Similarly, if the PNF-state is a subset of $PF = \{past, future\}$, it is assured that the interval is not happening in this instant of time. Of course, underlying this method there is the assumption about the minimal length of the *now* interval as described above.

The main problem of procedure 3.1 is that it relies on the computation of the minimal domain of an IA-network that we have already characterized as very computationally costly. In the next sections, we introduce faster ways to compute a conservative approximation of the PNF-states of the nodes of an IA-network.

3.4 A Better Representation: PNF-Networks

The computation of the PNF-state of the minimal domain of an IA-network is *NP-hard* due to two factors, as mentioned above: the exponential search through all possible solutions; and the exponential growth of the sets of intervals corresponding to the minimal domain. In this section, we provide a method to calculate an approximate solution that solves the latter problem; a solution for the former is described in the next section.

The method described in procedure 3.1 computes the PNF-state of a node after the minimal domain is determined. The basic idea of the approach described in this section is to project the initial set of unary constraints into the PNF space and to compute the minimal domain there using special mechanisms, completely eliminating the explosion in the number of intervals. As we will show in the end of this section, this process can not guarantee to produce the same results as procedure 3.1, but it is conservative, in the sense that no possible solution is lost.

In our method not only the intervals are projected into the PNF space, but the IA-network itself. We started our research developing minimal domain methods for PNF constraint propagation running on IA-networks, but later we found that it would be much more computationally efficient to project the IA-networks into special, 3-valued networks that we named *PNF-networks*.

3.4.1 PNF-Networks: Definition

Let us start by providing a formal definition of our special, three-valued networks, and by establishing some useful notation. A *PNF-network* is a 3-valued binary constraint satisfaction network where the domain of the all nodes $w_1, w_2, \ldots, w_n$ is the set of symbols $m = \{past, now, future\}$. An unary constraint R_i on a node w_i can be represented by an element of $M = \{\varnothing, P, N, F, PN, NF, PF, PNF\}$. For instance, the constraint $R_i = PN$ constrains the node w_i to assume only the elements *past* or *now*. We alternatively denote this by saying that R_i is the *restricted domain* of the node w_i.

A binary constraint $\hat{R}_{ij}$ between two nodes w_i and w_j is a truth matrix which determines which the admissible values are for the pair of nodes (w_i, w_j). We represent these 3x3 matrixes using a triad $\langle r_P, r_N, r_F \rangle \in M \times M \times M$ where r_P represents the admissible values for w_j when $w_i = past$, and similarly for r_N and r_F.

For example, suppose two nodes w_i and w_j are constrained by:

$$\hat{R}_{ij} = \langle PN, F, F \rangle$$

According to this constraint $\hat{R}_{ij}$, the only values that the pair of nodes (w_i, w_j) can assume are: from the constraint $r_P = PN$, they can be either (*past*, *past*) or (*past*, *now*); if w_i is *now*, the constraint $r_N = F$, forces w_j to be *future*, and therefore (w_i, w_j) can only by (*now*, *future*); and from $r_F = F$, (*future*, *future*).

3.4.2 Projecting IA-Networks into PNF-Networks

Given an IA-network *X* with a set of unary constraints $\sigma_1, \sigma_2, \ldots, \sigma_n$, we can associate a PNF-network X' with the same number of nodes and node domains $S(\sigma_1), S(\sigma_2), \ldots, S(\sigma_n)$. To understand how to map binary constraints between nodes in an IA-network into binary constraints in a PNF-network, we start with the following example.

Suppose a pair of nodes (x_i, x_j) is constrained in the original IA-network by the relation meet *(m)*, that is, in any solution $I_1, I_2, \ldots, I_n$, the interval I_i of must be followed immediately by I_j. Intuitively, if I_i is happening now, $s(I_i) = N$, then I_j can only occur in the future, yielding $s(I_j) = F$. Similarly, if I_j has not happened yet, $s(I_i) = F$, then I_j must also be *future*,

Table 3.2 The function $\gamma(r)$ which maps Allen's primitives into PNF-network binary constraints.

$\gamma(r)$	
r	$\langle r_P, r_N, r_F \rangle$
e	<*P,N,F*>
b	<*PNF,F,F*>
ib	<*P,P,PNF*>
m	<*PN,F,F*>
im	<*P,P,NF*>
o	<*PN,NF,F*>
io	<*P,PN,NF*>
s	<*PN,N,F*>
is	<*P,PN,F*>
d	<*PN,N,NF*>
id	<*P,PNF,F*>
f	<*P,N,NF*>
if	<*P,NF,F*>

$s(I_j) = F$. Finally, if it is know that $s(I_i) = P$, the interval I_i has already finished, meaning that the interval I_j must have started at some point in the past, although it may have finished or not — therefore $s(I_j) = P$ or $s(I_j) = N$.

As we saw above, the relationship meet between the nodes x_i and x_j can be mapped into the binary constraint $P_{ij} = \langle PN, F, F \rangle$ such as to preserve the temporal ordering contained in the original IA-network (as we will formalize later).

Table 3.2 displays the function $\gamma: \Lambda \rightarrow M \times M \times M$ that maps the 13 Allen's primitive relationships into their equivalent binary-constraints between PNF symbols.

Given a binary constraint in an IA-network, i.e., a disjunction of primitive relationships *R*, we can define its projection by the function $\Gamma: \overline{\Lambda} \rightarrow M \times M \times M$:

$$\Gamma(R) = \bigcup_{r \in R} \gamma(r)$$

where the union of the PNF-network binary constraints is defined as their component-wise union. For instance, if $R = \{m, im\}$, then

$$\begin{aligned} \Gamma(R = \{m, im\}) &= \gamma(m) \cup \gamma(im) \\ &= \langle PN, F, F \rangle \cup \langle P, P, NF \rangle \\ \Gamma(R) &= \langle PN, PF, NF \rangle \end{aligned}$$

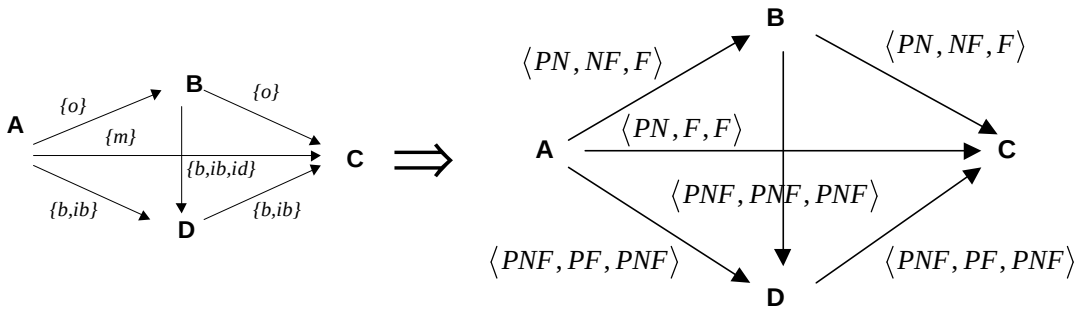


Figure 3.9 The projection into a PNF-network of the IA-network of fig. 3.4.

Notice in particular that $r_N = PF$, that is, if the first interval is *now* then the second interval can only be in the *past* or in the *future* states. This is the constraint that typifies mutually exclusive nodes in a PNF-network.

Given an IA-network, we can project it into a PNF-network where each binary constraint R_{ij} is projected into its PNF-equivalent $\Gamma(R_{ij})$. For example, fig. 3.9 displays the result of projecting the IA-network of fig. 3.2, which is the path-consistent version of the network of fig. 3.1. In general, it is always recommendable to run path-consistency before projecting to assure tighter constraints between the nodes. Of course, higher degrees of consistency are even more desirable: as noted before, the consistent relation between variables *B* and *D* is in fact *{b,ib}*, and not *{b,ib,id}* as determined by path-consistency. If that was the case, the PNF constraint between *B* and *D* in fig. 3.9 would be strengthened from $\langle PNF, PNF, PNF\rangle$ to $\langle PNF, PF, PNF\rangle$.

Notice that, following the reasoning of the example in the beginning of this subsection, the projection process preserves the "projected truth" of a solution. That is, if two intervals satisfy the relations between a pair of nodes, the projection into PNF of the intervals satisfies the projection of the relations. This is captured by the following proposition.

Proposition 3.1 Let I_1 and I_2 be two time intervals that satisfy a primitive Allen relationship *r*, $I_1 \ r \ I_2$. Then, their PNF-state satisfy the projection of *r* through γ,

$$I_1 \ r \ I_2 \Rightarrow s(I_1) \ \gamma(r) \ s(I_2)$$

Proof. By enumeration of all possibilities. Let us do here just the proof for one primitive relationship, during *(d);* the other relationships are proved in similar ways. So, assume the primitive relation *r* is during (*d*), and therefore by table 3.2, $\gamma(d) = \langle PN, N, NF\rangle$.

Suppose $s(I_1) = P = \{past\}$. By definition, $now\ \{ib,im\}\ I_1$, which, combined with the assumption $I_1\ \ d\ \ I_2$, yields, by transitive closure, $now\ \{ib,im,io,d,f\}\ \ I_2$. By our definition of *now*, it never happens that $now\ \ io\ \ I_2$, leaving thus only two possibilities: $now\ \{ib,im\}\ \ I_2$, yielding $s(I_2) = P$; or $now\ \{d,f\}\ \ I_2$, yielding $s(I_2) = N$. Therefore, if $s(I_1) = P$, then $s(I_2) = P$ or $s(I_2) = N$, and thus $s(I_1)\ \ \gamma(r)\ \ s(I_2)$.

Now, suppose $s(I_1) = N$. Similarly, $now\ \{s,d,e,f\}\ \ I_1$, and by transitive closure, $now\ \{d\}\ \ I_2$, and therefore $s(I_2) = N$, satisfying the relation $\gamma(d) = \langle PN, N, NF \rangle$. Last, if $s(I_1) = F$, the transitive closure produces $now\ \{b,m,o,s,d\}\ \ I_2$, and thus $s(I_2) = N$ or $s(I_2) = F$ after the exclusion of the impossible situation $now\ \ o\ \ I_2$.

The proof for the other primitive relationships is altogether similar.◆

Notice that the converse is not true. Consider, in our model of intervals in the real line, the intervals $I_1 = [1,3]$ and $I_2 = [2,4]$ when the reference interval *now* is [0,0.1]. It is not true that $I_1\ \ b\ \ I_2$, but $s(I_1)\ \ \gamma(b)\ \ s(I_2)$ is true, since $s(I_1) = F$, $s(I_2) = F$, and $\gamma(b) = \langle PNF, F, F \rangle$.

We can easily extend proposition 3.1 to sets of intervals and relationships, as stated in proposition 3.2.

Proposition 3.2 Let σ_1 and σ_2 be sets of intervals, and R a set of primitive Allen relationships. If every pair of intervals $I_1 \in \sigma_1$, $I_2 \in \sigma_2$ satisfies R, $I_1\ \ R\ \ I_2$, then the PNF-projection of σ_1 and σ_2 satisfy the projection of R through Γ

$$I_1\ \ R\ \ I_2 \Rightarrow S(\sigma_1)\ \ \Gamma(R)\ \ S(\sigma_2)$$

Proof. Let $s_1 \in S(\sigma_1)$ and $s_2 \in S(\sigma_2)$ be particular PNF-states of the projected sets of intervals. Since they are in the projection, there exist $I_1 \in \sigma_1$ and $I_2 \in \sigma_2$ such as $s(I_1) = \{s_1\}$ and $s(I_2) = \{s_2\}$. By the hypothesis, $I_1\ \ R\ \ I_2$, and therefore there exists a primitive relation $r \in R$ that satisfies $I_1\ \ r\ \ I_2$. According to proposition 3.1, $s(I_1)\ \ \gamma(r)\ \ s(I_2)$. Since $r \in R$, by definition, $\gamma(r) \subseteq \Gamma(R)$, yielding that $s(I_1)\ \ \Gamma(R)\ \ s(I_2)$, and $\{s_1\}\ \ \Gamma(R)\ \ \{s_2\}$. This is valid for every element $s_1 \in S(\sigma_1)$ and $s_2 \in S(\sigma_2)$, and therefore $S(\sigma_1)\ \ \Gamma(R)\ \ S(\sigma_2)$.◆

Proposition 3.2 assures that constraints that are satisfied in the original IA-network are also satisfied in the projected network, although the converse is not true (just consider the same counter example used for proposition 3.1). We soon will show that a similar result holds for minimal domains but before that we have to examine how to compute the minimal domain of a PNF-network.

3.4.3 The Minimal Domain of PNF-Networks

It is straightforward to extend the concepts of solution, feasible values and minimal domains to a given a PNF-network and its unary and binary constraints. As we will see later, most of our

algorithms and methods for action recognition are related to the computation of the minimal domain of a PNF-network subjected to a fixed set of binary constraints but with time-varying unary constraints. This motivated us to have a common representation for minimal domains and unary constraints that we call a component domain.

Formally, given a PNF-network on nodes $w_1, w_2, \ldots, w_n$, a *component domain* W of the PNF-network is any combination of domains W_i on each of the nodes w_i of the PNF-network, denoted by $W = (W_i)_i = (W_1, W_2, \ldots, W_n)$, where each W_i is a subset of $m = \{past, now, future\}$, that is $W_i \in M$. The value of each W_i in a component domain W is called the *PNF-domain* of the node w_i. We also denote by U the set of all possible component domains of a PNF-network, $U = M^n$, where n is, as always, the number of nodes in the PNF-network.

Given a PNF-network and a component domain W, consider the problem of finding the minimal domain of the network by imposing the components of W as unary constraints on the network's nodes. First, notice that both solutions of PNF-networks and minimal domains are also representable by component domains. To simplify notation we define a function that computes the minimal domain of a PNF-network given unary constraints W, called the *restriction of W*, $R(W):U \to U$ where each component $R(W)_i$ of $R(W)$ is the minimal domain of the node w_i:

$$R(W) = \big(R(W)\big)_i = \big(R(W_1), R(W_2), \ldots, R(W_n)\big)$$

Procedure 3.2, described below, computes the minimal domain of a network constrained by a component domain W of unary constraints by (almost) brute force search through the space of solutions.

Procedure 3.2

Input: a PNF-network and a component domain W.

Output: the minimal domain of W, $R(W)$.

```
L ← W                                          (1)
for i from 1 to n                              (2)
  for each x_i ∈ L_i                           (3)
      S ← (W_1, W_2, ..., {x_i}, ..., W_n)     (4)
      P ← FindASolution(S)                     (5)
      if P = ∅                                 (6)
          W_i ← W_i \ {x_i}                    (7)
      else                                     (8)
          L ← L \ P                            (9)
return W                                       (10)
```

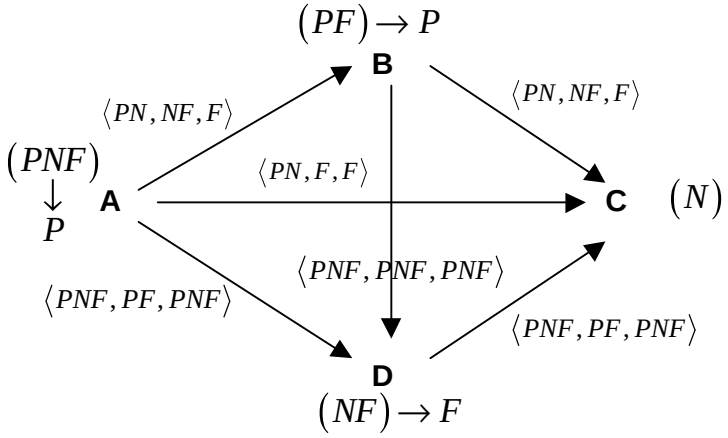


Figure 3.10 Minimal domain of a PNF-network where the original constraints are inside the parenthesis.

Procedure 3.2 uses a search procedure *FindASolution (W)* that, given a component domain *W*, returns one solution of the network given the unary constraints represented by *W*, or the empty set if there is no solution. Typically, *FindASolution* is implemented using back-tracking mechanisms and with a running time proportional to 3^n, where *n* is the number of nodes in the PNF-network. Procedure 3.2 is quite straightforward, searching for a solution which includes every element of each variable domain. In line 6, elements of variables that do not belong to solutions are excluded from the result of the procedure.

Since each domain of the component domain *W*, and therefore of *L*, has at most three elements, *past*, *now*, or *future*, it is clear that *FindASolution* is called at most *3n* times. However, procedure 3.2 includes an optimization that, in practice, increases considerably the running time. In line 9, all the elements of a found solution *P* are eliminated from future search (contained in *L*) because the procedure have already found a solution for them. Therefore, if the PNF-network is consistent, that is, has at least one solution, the solution found for an element of the first domain examined, W_1, eliminates at least one element from every other domain, and *FindASolution* is called at most *2n+2* times.

Figure 3.10 shows the minimal domain of the network in fig. 3.9 given the original constraints $W = (PNF, PF, N, NF)$ (shown inside parenthesis in fig. 3.10). In these conditions, the minimal domain is $R(W) = (P, P, N, F)$. In this case, as we see, the computation of the minimal domain determines completely the PNF-state of its intervals.

The problem is that procedure 3.2 is still exponential, as it should be expected, since it is known that computing the minimal domain of a 3-valued constraint network is *NP-complete* (see [46]). Notice, however, that unlike in the case of IA-networks (as discussed by Hyvönen [62]), the exponential growth comes only from the search of solutions. Section 3.5 will examine how we can compute an approximation of the minimal domain in linear time. But

first, we have to determine how the minimal domain of projected PNF-networks relates to the minimal domain of the original IA-network.

3.4.4 Minimal Domains in IA-Networks and in PNF-Networks

The objective of this subsection is to show that the process of computing minimal domain in PNF-networks is a conservative approximate of the minimal domain of the original IA-network. That is, the minimal domain of a projected PNF-network contains the PNF-state of the minimal domain of the IA-network. This is formally stated and proved in proposition 3.3.

Proposition 3.3 Let X be an IA-network, $\sigma_1,\sigma_2,\ldots,\sigma_n$ a set of unary constraints on its variables, $\overline{\sigma_1},\overline{\sigma_2},\ldots,\overline{\sigma_n}$ the minimal domain of X under those constraints, and $S(\overline{\sigma_1}),S(\overline{\sigma_2}),\ldots,S(\overline{\sigma_n})$ the PNF-states of the minimal domain. Let X' be the PNF-projection of X, with unary constraints $S(\sigma)=(S(\sigma_1),S(\sigma_2),\ldots,S(\sigma_n))$ corresponding to the projection of the unary constraints of X. Then the minimal domain of X' under $S(\sigma)$, $R(S(\sigma))$, contains the minimal domain of X, that is, for every variable i,

$$S(\overline{\sigma_i}) \subseteq R(S(\sigma))_i$$

Proof. Let us consider an element $s_i \in S(\overline{\sigma_i})$. Since $\overline{\sigma_i} \subseteq \sigma_i$ implies, by definition, $S(\overline{\sigma_i}) \subseteq S(\sigma_i)$, we have that $s_i \in S(\sigma_i)$. Since $\overline{\sigma_1},\overline{\sigma_2},\ldots,\overline{\sigma_n}$ is the minimal domain, there exists a solution $I_1,I_2,\ldots,I_n$ of X, $I_i \in \overline{\sigma_i}$, where $s(I_i)=s_i$. Let us consider the projection of this solution, $(s(I_1),s(I_2),\ldots,s(I_n))$. According to proposition 3.2, these values satisfy every relationship between two variables in X'; in fact, since $I_i\ R_{ij}\ I_j$, we obtain that $s(I_i)\ \Gamma(R_{ij})\ s(I_j)$. Together with the fact that $s(I_i) \subseteq S(\overline{\sigma_i}) \subseteq S(\sigma_i)$, it is clear that $(s(I_1),s(I_2),\ldots,s(I_n))$ is a solution of X' under $S(\sigma)=(S(\sigma_1),S(\sigma_2),\ldots,S(\sigma_n))$, and therefore, is contained in the minimal domain. Since, by construction, $s_i \in s(I_i)$, we obtain $s_i \in R(S(\sigma))_i$, and given that this is valid for every element $s_i \in S(\overline{\sigma_i})$ we obtain $S(\overline{\sigma_i}) \subseteq R(S(\sigma))_i$. ◆

Unfortunately the converse is again not true. Consider a network composed of two nodes, *A* and *B*, linked by a meet constraint, *A {m} B* as shown in fig. 3.11. Now, suppose unary constraints $\sigma_A=\{[0,2],[1,3]\}$ and $\sigma_B=\{[2,4]\}$; clearly, the minimal domain is $\overline{\sigma_A}=\{[0,2]\}$ and $\overline{\sigma_B}=\{[2,4]\}$. In the situation where *now* happens in $[0,0.1]$, the corresponding PNF-states are $S(\overline{\sigma_A})=N$ and $S(\overline{\sigma_B})=F$. However, if we first project the network and the unary

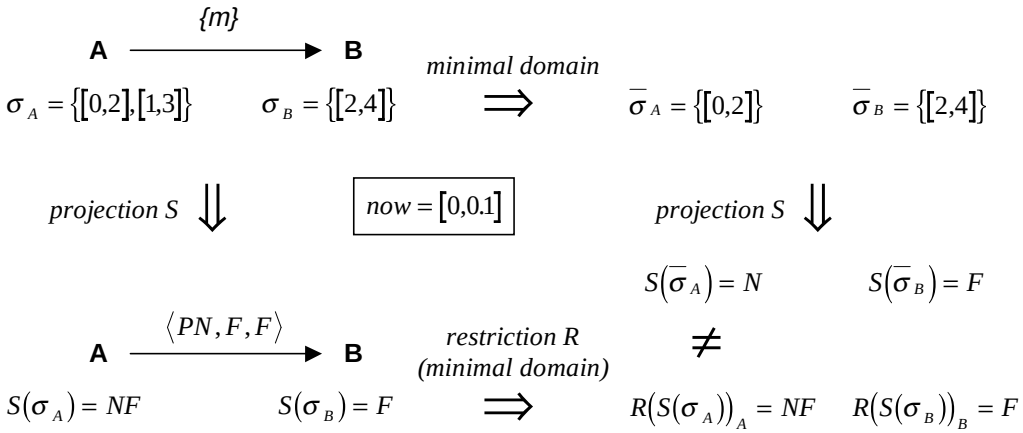


Figure 3.11 Example of an IA-network with a minimal domain whose projection is properly contained in the minimal domain of the projected PNF-network.

constraints, obtaining $S(\sigma_A) = NF$ and $S(\sigma_B) = F$, and then compute the restriction, we obtain the same values, $R(S(\sigma_A))_A = NF$ and $R(S(\sigma_B))_B = F$, since $\gamma(m) = \langle PN, F, F \rangle$. As we see, in this example $S(\overline{\sigma_A}) = N$ is different, although contained, in $R(S(\sigma_A))_A = NF$.

Proposition 3.3shows that the computation of the minimal domain through PNF-networks is a conservative estimation of the true PNF-state of the minimal domain. That is, the PNF-states corresponding to any solution of the IA-network are always contained in the minimal domain of the projected PNF-network, under the same initial constraints. Notice also that if the minimal domain of the PNF-network is empty, that is, the network is inconsistent, then the original IA-network has no solutions.

In practice, the fact that computing the minimal domain of PNF-networks overshoots the minimal domain of the IA-network has not been a problem, and it has been quite overshadowed by the speedup enabled by the use of PNF-networks as we discuss in the next chapters.

3.4.5 The Space of Projected PNF-Networks

To finalize our exposition about PNF-networks we want to examine the structure of the space of the projected PNF-networks and some of its properties that are responsible for fast constraint propagation methods.

Let us consider first the process of projecting IA-networks itself. In theory, there is a great reduction in the amount of information contained in an IA-network, since each of the 8192 types of arcs are projected into a 3x3 truth matrix, which has only $2^9 = 512$ different possibilities. However, if we consider all 8192 different disjunctions of Allen primitives, their projection through Γ covers only 33 of the possible 512 constraints between nodes in a PNF-

Table 3.3 The PNF relations that actually occur in projected IA-networks, and their corresponding inverses.

$\hat{R}$	$\Gamma^{-1}(\hat{R})$
<PNF,PNF,PNF>	*<all>*
<PNF,PNF,NF>	*{e, b, d, id, o, io, m, im, s, is, f, if}*
<PN,PNF,PNF>	*{e, ib, d, id, o, io, m, im, s, is, f, if}*
<PNF,NF,NF>	*{e, b, d, o, m, s, f, if}*
<PNF,PNF,F>	*{e, b, id, o, m, s, is, f}*
<PN,PNF,NF>	*{e, d, id, o, io, m, im, s, is, f, if}*
<P,PNF,PNF>	*{e, ib, id, io, im, s, is, f}*
<PN,PN,PNF>	*{e, ib, d, io, im, s, is, f}*
<PNF,NF,F>	*{e, b, o, m}*
<PN,PNF,F>	*{e, id, o, m, s, is, if}*
<PN,NF,NF>	*{e, d, o, m, s, f, if}*
<PN,PN,NF>	*{e, d, io, im, s, is, f}*
<P,PNF,NF>	*{e, id, io, im, is, f, if}*
<P,PN,PNF>	*{e, ib, io, im, is, f}*
<PN,PN,F>	*{e, s, is}*
<PN,NF,F>	*{e, o, m, s, if}*
<P,PNF,F>	*{e, id, is, if}*
<PN,N,NF>	*{e, d, s, f}*
<P,PN,NF>	*{e, io, im, is, f}*
<P,NF,NF>	*{e, f, if}*
<PN,N,F>	*{e, s}*
<P,PN,F>	*{e,is}*
<P,NF,F>	*{e, if}*
<P,N,NF>	*{e, f}*
<PNF,PF,PNF>	*{b, ib, m, im}*
<PN,PF,PNF>	*{ib, m, im}*
<PNF,PF,NF>	*{b, m, im}*
<PN,PF,NF>	*{m, im}*
<P,P,PNF>	*{ib, im}*
<PNF,F,F>	*{b, m}*
<P,P,NF>	*{im}*
<PN,F,F>	*{m}*
<P,N,F>	*{e}*

network. For instance, there is no set of Allen primitives whose projection yields $\langle N, PF, N \rangle$. In fact, it is always true that, given any set of primitive relations R and its projection $\Gamma(R) = \langle r_P, r_N, r_F \rangle$, then $P \subseteq r_P$ and $F \subseteq r_F$.

Besides that, the image of Γ has a nice nested structure. That is, given a PNF-relation $\hat{R}$, every set of relations R that maps into P, $\Gamma(R) = \hat{R}$, is contained in a "maximal" set of relations in $\overline{\Lambda}$ which we will note as $\Gamma^{-1}(\hat{R})$. In fact, the nested structure allows us to define such "inverse" function for Γ, $\Gamma^{-1}: M \times M \times M \to \overline{\Lambda}$.

Table 3.3 shows the 33 PNF relations that actually occur in projected IA-networks and the maximal sets of Allen relations that project into them. We can also use the table as a way to compute the projection of a set of relations *R*. In this case, to project *R* we just search the column of inverses looking for the smaller set that contains *R*. The nested structure guarantees that there is only one minimal set that corresponds, in fact, to $\Gamma(R)$.

One of the consequences of having only 33 different relations in projected IA-networks, compared to 8192 in IA-networks, is that many basic operations become easily implementable by small look-up tables. Moreover, it becomes a lot faster to examine the space of the networks through enumeration. For instance, in the next section we will show that 3-node and 4-node PNF-networks have some interesting features. These features were determined through exhaustive enumeration of all possible 3-node and 4-node PNF-networks. If we were using the original IA-network space where arcs come in 8192 different types, we would have to examine $8192^3 \cong 5 \times 10^{11}$ 3-node networks, and $8192^4 \cong 4.5 \times 10^{15}$ 4-node networks. Instead, we could examine only $33^3 = 35{,}937$ and $33^4 = 1{,}185{,}921$ PNF-networks, respectively.

Another important property of the projection process is that it preserves path-consistency. To verify this, we start by defining a function $F: (M \times M \times M) \times M \to M$ that, given a PNF-binary constraint $\hat{P} = \langle r_P, r_N, r_F \rangle$, maps a PNF-state W_0 into another which contains the image of W_0 through the binary constraint,

$$F(\langle r_P, r_N, r_F \rangle, W_0) = \bigcup_{s \in W_0} r_s$$

Let us now define the concept of transitive closure in the space of PNF-networks. As explained in [177], the transitive closure between two relations is the least restrictive relation that is permitted by the composition of the two relations. That is, if $\hat{O}$ is the transitive closure of PNF-relations $\hat{P}$ and $\hat{Q}$, we should have that, for every PNF-state W_0,

$$F(\hat{O}, W_0) \subseteq F(\hat{Q}, F(\hat{P}, W_0))$$

and $\hat{O}$ being such that $F(\hat{O}, W_0)$ is maximal. Figure 3.12 depicts a diagram of these relationships.

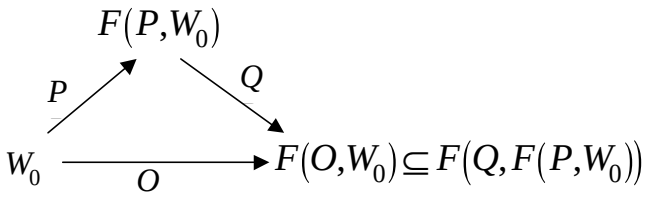


Figure 3.12 Definition of transitive closure.

Let us then define the transitive closure in the PNF realm by simply reverting to the IA-network definition of transitive closure. Given PNF-relations *P* and *Q*, we define the *transitive closure of P and Q* as the result of the function *ptc*: $M^3 \times M^3 \rightarrow M^3$

$$ptc(P,Q) = \Gamma\left(TC(\Gamma^{-1}(P),\Gamma^{-1}(Q)\right)$$

where TC is the transitive closure of two disjunctions of Allen primitives as defined in section 3.2. To show that this is actually the transitive closure, we first observe that the projection function Γ is well behaved. In fact, for any sets of Allen primitive relations $R,S \in \overline{\Lambda}$, and any PNF-state $W_0 \in M$

$$F\left(\Gamma\left(TC(R,S)\right),W_0\right) = F\left(\Gamma(S),F\left(\Gamma(R),W_0\right)\right)$$

This is proved by enumeration, checking all the $8192 \times 8192 \times 8 = 536{,}870{,}912$ possibilities using a computer program. Now, given two PNF-relations *P* and *Q*, and their inverses $\Gamma^{-1}(P)$ and $\Gamma^{-1}(Q)$, it follows immediately that

$$F\left(\Gamma\left(TC(\Gamma^{-1}(P),\Gamma^{-1}(Q))\right),W_0\right) = F\left(\Gamma\left(\Gamma^{-1}(Q)\right),F\left(\Gamma\left(\Gamma^{-1}(P)\right),W_0\right)\right)$$

and therefore, since $\Gamma\left(\Gamma^{-1}(X)\right) = X$ for any PNF relation $X \in M \times M \times M$, we obtain

$$F\left(ptc(P,Q),W\right) = F\left(Q,F(P,W)\right)$$

Since equality holds, it is clear that $ptc(P,Q)$ is maximal and thus the transitive closure function for PNF-networks. This result yields the following proposition that guarantees that the projection of a path-consistent IA-network is path-consistent.

Proposition 3.4 . Let *X* be an IA-network, and X' its PNF projection. If *X* is path consistent then X' is path-consistent.

Proof. Since *X* is path-consistent, for every triad of nodes x_i, x_j, x_k we have that their relations R_{ij}, R_{jk}, R_{ik} satisfy $R_{ik} \subseteq TC\left(R_{ij}, R_{jk}\right)$. Let us consider the projection of those relations,

$P_{ij} = \Gamma(R_{ij})$, $P_{jk} = \Gamma(R_{jk})$, and $P_{ik} = \Gamma(R_{ik})$. Since for any set of relations $R \in \overline{\Lambda}$ it is always true that $R \subseteq \Gamma^{-1}(\Gamma(R))$, we can conclude that

$$R_{ik} \subseteq TC(R_{ij}, R_{jk}) \subseteq TC(\Gamma^{-1}(\Gamma(R_{ij})), \Gamma^{-1}(\Gamma(R_{jk})))$$

Applying the projection in both sides and observing the definition of *ptc*, we obtain

$$\Gamma(R_{ik}) \subseteq \Gamma(TC(\Gamma^{-1}(\Gamma(R_{ij})), \Gamma^{-1}(\Gamma(R_{jk})))) = ptc(\Gamma(R_{ij}), \Gamma(R_{jk}))$$

and therefore that $P_{ik} \subseteq ptc(P_{ij}, P_{jk})$, or simply, X' is also path-consistent.◆

3.5 Computing an Approximation of the PNF-Restriction

Computing the minimal domain in 3-valued constraint satisfaction networks is, in general, an *NP-hard* problem [46, 103, 178]. In our experiments, we have been employing an arc-consistency algorithm (based on [90]) to compute an approximation of the minimal domain in linear time. In this section we define the notion of arc-consistency and provide a linear algorithm (based in Mackworth's arc-consistency algorithm [90]) to compute it. We then proceed examining how good is this approximation.

Following the general definition provided by Dechter in [46], a PNF-network with unary constraints $W = (W_i)_i \in U$ is *arc-consistent under W* if and only if, for each binary constraint $\hat{R}_{ij}$ between two variables w_i and w_j, the image of every component W_i of *W* is contained in W_j,

$$W_j \subseteq F(\hat{R}_{ij}, W_i)$$

The real meaning of arc-consistency is that, in an arc-consistent network, given a value in a variable that satisfies the unary constraint, it is always possible to find a value in any other variable that satisfies its local constraint and the binary constraint between them.

To simplify notation, let us define a function $AC(W): U \to U$ which maps a component domain of a network into the maximal arc-consistent network contained in *W*. It is easy to see that, for any set of unary constraints *W*, the minimal domain is contained in the maximal arc-consistent network contained in *W*.

Proposition 3.5 For any component domain *W* of a PNF-network,

$$R(W) \subseteq AC(W)$$

Proof. Just observe that a node value that belongs to a solution for the whole network satisfies the binary constraints with any other node value in the solution.◆

3.5.1 The Arc-Consistency Algorithm

Procedure 3.3 shows an algorithm that computes the maximal arc-consistent network under a component domain *W*. This is a version of the arc-consistency algorithm *AC-2* proposed by Mackworth in [90] and adapted here to the component domain notation. The algorithm uses the function *F* defined above.

Procedure 3.3

Input: a PNF-network with variables $w_1, w_2, \ldots, w_n$ and binary constraints $\hat{R}_{ij}$; a component domain, $W = (W_i)_i$ representing unary constraints in the variables.

Output: $AC(W)$, the maximal arc-consistent component domain that is contained in *W*

initialize a queue *Q* with all nodes w_j such that $W_j \neq PNF$ (1)
$\overline{W} \leftarrow W$ (2)
while $Q \neq \varnothing$ (3)
    $w_0 \leftarrow first(Q)$ (4)
    for each variable w_i (5)
        $X \leftarrow F(\hat{R}_{i_0 i}, \overline{W_0})$ (6)
        if $\overline{W_i} \neq X \cap \overline{W_i}$ (7)
            $\overline{W_i} \leftarrow X \cap \overline{W_i}$ (8)
            $queue(w_i, Q)$ (9)
return $\overline{W}$ (10)

The first step of the algorithm consists in detecting which nodes of the component domain *W* are different from PNF, and queuing all those nodes for further expansion. Then $\overline{W}$, the component domain to be returned is initialized identically to the input *W*. The core of the algorithm is a loop that ends when the queue *Q* is empty. In each cycle, one node w_0 at state $\overline{W_0}$ is examined. For each node w_i an auxiliary variable *X* is assigned the possible values for the domain $\overline{W_i}$ of w_i given the present domain $\overline{W_0}$ of w_0, computed by the function *F* described above. In steps 7-9, if necessary, the domain $\overline{W_i}$ of w_i is actualized with the intersection of its previous value and *X*, and the modified node w_i is put in the last position of the queue.

Let us verify that procedure 3.3 actually computes the maximal arc-consistent network under the component domain *W*. It is easy to see that the result is contained in *W*, since line 8 sets new values for components as intersections with the previous ones. To prove that the result is

arc-consistent, just consider the invariant that, before the test of the while instruction of line 3, it is always true that $\overline{W_i} \subseteq F\left(\hat{R}_{ji}, \overline{W_j}\right)$ for all nodes w_j which are not in the queue Q (the fact is trivially true before line 3, since all nodes w_j not in the queue have *PNF* as their state). To see that the invariant remains true after lines 4-9 let us first notice that after lines 7-9, it is true that $\overline{W_i} = F\left(\hat{R}_{i_0 i}, \overline{W_0}\right) \cap \overline{W_i}$ for any node w_i. Therefore, after the big loop of lines 4-9, although w_0 is not in the queue any more, it is true that $\overline{W_i} \subseteq F\left(\hat{R}_{i_0 i}, \overline{W_0}\right)$. However, new nodes have been queued but all those that were not queued keep satisfying the invariant since $\overline{W_i} = F\left(\hat{R}_{i_0 i}, \overline{W_0}\right) \cap \overline{W_i}$.

Therefore, upon termination, since the queue is empty, every node satisfies the invariant and therefore the result is arc-consistent. Since only values that do not satisfy the constraints are removed from the solution in line 8, it is easy to show that the output is also maximal.

Also, since the nodes can assume only 3 values and a node is queued only when its domain is reduced, we can have at most *3n* executions of the loop 4-9 and therefore the algorithm is guaranteed to terminate. Moreover, assuming constant time for the computation of the function *F* and the intersection operations (through look-up tables), we conclude that the algorithm is $O\left(n^2\right)$ in the number *n* of variables, or, as more commonly referred in the constraint propagation literature, linear in the number of constraints.

3.5.2 How Good is Arc-Consistency?

Before we go further and demonstrate how PNF-networks can be used to recognize and control interaction, we want to examine how well arc-consistency works as a method to approximate the minimal domain of a PNF-network.

In this analysis, we will assume that the original IA-network was processed using Allen's path-consistency algorithm and therefore, according to proposition 3.4, the PNF-network is path-consistent, that is, any triad of relations $\hat{R}_{ij}, \hat{R}_{jk}, \hat{R}_{ik}$ among 3 nodes w_i, w_j, w_k satisfies $\hat{R}_{ik} \subseteq ptc\left(\hat{R}_{ij}, \hat{R}_{jk}\right)$.

The basis of our arguments is a result by Dechter [46] that demonstrates that any 3-valued binary network that is strong 4-consistent is globally consistent. Global consistency implies that in any 4-consistent 3-valued network, the minimal domain is exactly equal to the result of the arc-consistency algorithm. So, we want to examine how far a path-consistent PNF-network is from being strong 4-consistent.

According to Dechter, a network is *strong i-consistent* when it is j-consistent for every $j = 1,2,\ldots i$. A network is *i-consistent* when every partial solution of the network involving $i-1$ nodes can be expanded to a partial solution with *i* nodes. For instance, an arc-consistent

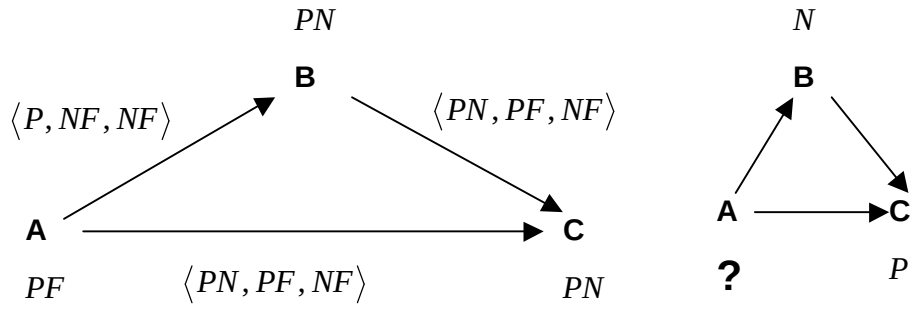


Figure 3.13 An example of a path-consistent, 2-consistent network that is not 3-consistent.

network is 2-consistent, since any value assumed by one individual variable can be extended such as there is another value in each of its neighbors that satisfy the constraint between them.

Unfortunately, path-consistency, in the way we defined it, does not imply 3-consistency in Dechter's sense. In fact, as shown by Meiri [103], it is easy to construct path-consistent 3-node IA-networks where there are 2-node partial solutions that can not be extended.

Figure 3.13 shows an example of a path-consistent PNF-network that is 2-consistent but it is not 3-consistent. To see that the network is path-consistent, let us compute the transitive closure of $\hat{R}_{AB}$ and $\hat{R}_{BC}$. Since, according to table 3.3, $\Gamma^{-1}\left(\hat{R}_{AB}\right) = \{e, f, if\}$ and also that $\Gamma^{-1}\left(\hat{R}_{BC}\right) = \{e, d, io, im, s, is, f\}$, we can compute the transitive closure as

$$\begin{aligned} ptc\left(\hat{R}_{AB}, \hat{R}_{BC}\right) &= \Gamma\left(TC\left(\Gamma^{-1}\left(\langle P, NF, NF\rangle\right), \Gamma^{-1}\left(\langle PN, PF, NF\rangle\right)\right)\right) \\ &= \Gamma\left(TC\left(\{e, f, if\}, \{e, d, im, io, s, is, f\}\right)\right) \\ &= \Gamma\left(\{e, b, im, d, id, o, io, s, is, f, if\}\right) \\ &= \langle PN, PNF, PNF\rangle \end{aligned}$$

And therefore, since $\hat{R}_{AC} = \langle PN, PN, NF\rangle \subset \langle PN, PNF, PNF\rangle$, the network is path-consistent. It is straightforward to see that the network in fig. 3.13 is 2-consistent, that is, it is arc-consistent. However, the network is not 3-consistent as shown in the diagram in the right of fig. 3.13. Given the values *N* for node *B* and *P* for node *C*, the constraint $\hat{R}_{BC}$ between them is satisfied by these values, but there is no value of *A* (inside its initial domain *PF*) that satisfies at the same time the constraints $\hat{R}_{AB}$ and $\hat{R}_{AC}$.

This, however, does not diminish the importance of pre-processing the PNF-network in order to assure that it is path-consistent. In particular for our approach, detailed in the following chapters, we compute path-consistency off-line and only run arc-consistency for each run-time cycle. This produces, for each time cycle, a PNF-network that is path-consistent and 2-consistent. First, notice that we could take the resulting network with its unary constraints and

run procedure 3.2 which computes by brute-force search the minimal domain. As shown by Mackworth and other researchers [82, 90], it is often the case that the occurrence of back-tracking is significantly reduced after arc- and path-consistency.

In our applications we have not even gone that far but instead we have considered the arc-consistent PNF-network our final approximation. In order to estimate how often we are overshooting the minimal domain, we ran some experiments trying to verify how often path-consistent 2-consistent PNF-networks are not 3-consistent and 4-consistent — and therefore globally consistent.

Table 3.4 Occurrence of problems in 3-node networks.

	relations only	*relations and values*
Total cases	35,937	12,326,391
Consistent cases	5,345	1,449,649
% consistent cases	14.87%	11.76%
MD not equal to AC (in consistent cases)	942	2,598
% MD not equal to AC (in consistent cases)	17.62%	0.18%

Table 3.4 shows the results of examining 3-node PNF-networks. The data in table 3.4 was constructed by generating every possible combination of relations and PNF-states for a 3-node network. The first column lists the results grouped considering only the relations among the nodes while the second column consider all the different assignments of PNF-values to each 3-node PNF-network. That is, in the second column we consider individually each of the different $7^3 = 343$ possible assignments for a particular PNF-network.

For each configuration generated, we checked if the configuration was path- and arc-consistent (shown in the second row). As we see, only approximately 15% of the possible 3-node configurations are path-consistent and 12% of the cases are also arc-consistent. Most importantly, we then checked in how many of the consistent cases the minimal domain was different from the result of the arc-consistency algorithm. The results, listed in the last two rows, show that in more than 17% of the configurations there is at least one assignment of PNF-values that causes the minimal domain to be smaller than the result of the arc-consistency algorithm. However, if we look into how many of the individual assignments causes “trouble” to the arc-consistency algorithm, in only 0.2% of the cases the result of arc-consistency was different from the actual minimal domain.

Table 3.5 Occurrence of problems in 4-node networks.

	relations only	*relations and values*
Total cases	1,291,467,969	3,100,814,593,569
Consistent cases	4,175,450	6,938,350,000
% consistent cases	0.32%	0.22%
MD not equal to AC (in consistent cases)	1,806,050	32,535,000
% MD not equal to AC (in consistent cases)	43.25%	0.47%

Table 3.5 shows similar results for the case of 4-node networks. Using a computer program that run for about 5 days, we generated all possible 4-node PNF-networks and for each of them the $7^4 = 2401$ possible assignments for the nodes. Among all of the PNF-networks only 0.32% were consistent, and only 0.22% of the assignments. Among the approximately 4 million consistent PNF-networks, 43% allowed an assignment where the minimal domain was smaller than the result of arc-consistency. However, these assignments are rare, only about 0.47% of the total number of consistent assignments.

We examined a little further the troubled configurations and we noticed two interesting characteristics:

- if the PNF-state of any of the 3 nodes is either *P*, *N*, or *F*, then the minimal domain is equal to the result of the arc-consistency algorithm; that is, if one of the nodes is single-valued, the arc-consistency algorithm propagates values through the constraints to the "local" nodes as well as the minimal domain algorithm;
- if the 3 relations follow exactly the transitive closure, $\hat{R}_{ik} = ptc\left(\hat{R}_{ij}, \hat{R}_{jk}\right)$, then it is also true that the minimal domain is equal to arc-consistency.

Based on this analysis, we believe that it is quite common that in path-consistent, 2-consistent PNF-networks, the minimal domain is equal to the domain computed by the arc-consistency algorithm.

For the rest of this thesis, unless otherwise noted, we employ procedure 3.3 to determine an (approximation) of the minimal domain. We found no need to refine this result, specially considering that, as noted above, the minimal domain of the PNF-network can be bigger than the actual minimal domain of the IA-network. A similar conclusion, but in a different context, was reached by Ladkin and Reinefeld in [82].

3.6 PNF Propagation

To finish this chapter, we want to introduce one last concept that tries to expand into discrete time the ideas discussed so far. From the way the interval *now* is defined, it is clear that PNF-restriction deals exclusively with determining feasible options for the PNF-state of an action **at**

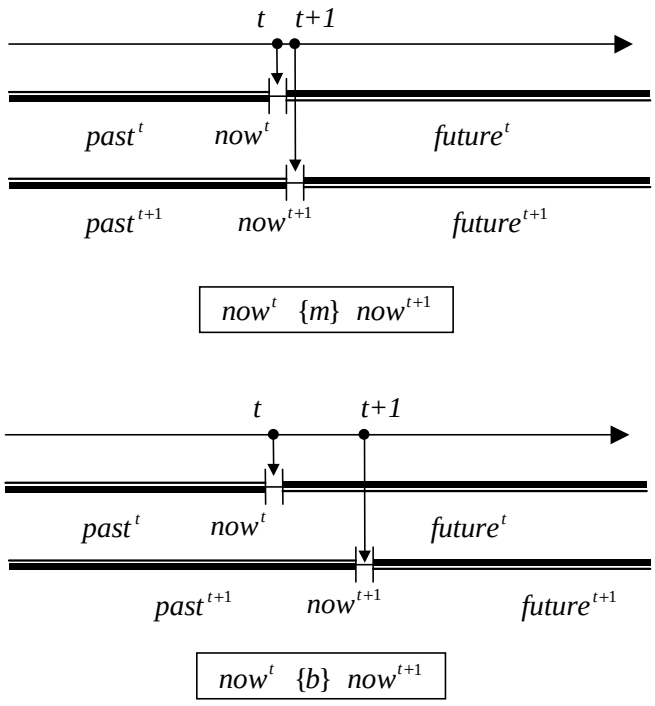


Figure 3.14 Diagram of the two different possibilities for two consecutive *now* intervals.

a given moment of time. The question is how much information from one moment of time can be carried to the next?

In fact, information from the previous time step can be used to constrain the occurrence of intervals in the next instant. For example, after a node is determined to be in the *past*, it should be impossible for it to assume another PNF-value, since, in our semantics, the corresponding action is over. Similarly, if the current value of the node is *now*, in the next instant of time it can still be *now* or the corresponding action might have ended, when the node should be *past*. To capture these ideas we define a function that time-expands a component domain into another that contains all the possible PNF-values that can occur in the next instant of time.

3.6.1 Time Expansion

Given the PNF-state of a variable in time, we want to define a function $\mathrm{T}{:}U \to U$, called the *time expansion function*, that considers a component domain W^t at time *t* and computes another component domain $W^{t+1} = \mathrm{T}\left(W^t\right)$ at time $t+1$ that contains the minimal domain at time $t+1$ if the original component domain W^t also contained it.

First, we have to observe that for each consecutive instant of time *t* there is a different interval now^t being associated. Given the semantics of *now*, that is, an interval never overlaps with *now*, we are left with just two options for the relation between two consecutive *now* intervals corresponding to consecutive instants of time. Either the two *now* intervals meet, $now^t \{m\} now^{t+1}$, or they follow each other, $now^t \{b\} now^{t+1}$. In particular, notice that they can not overlap, otherwise the other intervals could start or finish inside one of the *now* intervals. Figure 3.14 depicts the two different possibilities.

Let us consider first the case where the two intervals meet, $now^t \{m\} now^{t+1}$. Under this condition, we start by defining a time expansion function for each element of $m = \{past, now, future\}$, $\tau_m : m \to M$ such as:

$$\tau_m(past) = P$$
$$\tau_m(now) = PN$$
$$\tau_m(future) = NF$$

Given the function that time-expands elements of m, we define the function that expands the elements of M, $\mathrm{T}_m : M \to M$ as being the union of the results of τ_m,

$$\mathrm{T}_m(\Omega) = \bigcup_{\omega \in \Omega} \tau_m(\omega)$$

and the time expansion of a component domain *W*, $\mathrm{T}_m : U \to U$ (abusing the notation), as the component-wise application of the original T_m on a component domain $W = (W_i)_i$,

$$\mathrm{T}_m(W) = (\mathrm{T}_m(W_1), \mathrm{T}_m(W_2), \ldots, \mathrm{T}_m(W_n))$$

As much as we have a different interval now^t for each instant of time *t*, there are also different functions $s^t(I)$ and $S^t(\sigma)$. Now, consider an interval *I* with PNF-state $s^t(I)$ at time *t*. The next proposition guarantees that the PNF-state in the next instant of time is contained in the time expansion of $s^t(I)$.

Proposition 3.6 Let I be an interval with PNF-state $s^t(I)$ at time *t*. Assume $now^t \{m\} now^{t+1}$. Then the time expansion of the PNF-state contains the next PNF-state,

$$s^{t+1}(I) \subseteq \mathrm{T}_m(s^t(I))$$

Proof. Suppose first that $s^{t+1}(I) = P$, that is, $now^{t+1} \{ib, im\} I$. Since $now^t \{m\} now^{t+1}$, by transitive closure we obtain $now^t \{ib, io, id, im, e, f, if\} I$. Since no interval starts or ends inside a *now* interval, we have in fact $now^t \{ib, im, e, f\} I$. If $now^t \{ib, im\} I$, we obtain $s^t(I) = P$ and

therefore $\mathrm{T}_m(s^t(I)) = P$. Otherwise, $now^t\{e, f\}I$ yields $s^t(I) = N$, but since the time expansion guarantees $\mathrm{T}_m(s^t(I)) = PN$, we obtain $P = s^{t+1}(I) \subseteq \mathrm{T}_m(s^t(I))$.

Now, suppose that $s^{t+1}(I) = N$, that is $now^{t+1}\{s,d,e,f\}I$. In this case, the transitive closure produces $now^t\{m,o,s,d\}I$, from which the relation *o* is eliminated since no interval is overlapped by *now*, yielding $now^t\{m,s,d\}I$. If $now^t\{s,d\}I$ we have $s^t(I) = N$, and therefore $\mathrm{T}_m(s^t(I)) = PN$. In the case that $now^t\{m\}I$ we obtain $s^t(I) = F$, and thus $\mathrm{T}_m(s^t(I)) = NF$. In both cases, we verify that $N = s^{t+1}(I) \subset \mathrm{T}_m(s^t(I))$.

Finally, if $s^{t+1}(I) = F$, a similar reasoning produces $now^t\{b\}I$, and therefore $s^t(I) = F$ and $\mathrm{T}_m(s^t(I)) = NF$.◆

Let us now consider the other possibility for the occurrence of the *now* intervals, that is, when they follow each other but do not meet, $now^t\{b\}now^{t+1}$. Under this condition, an interval can happen between two consecutive *now* intervals, or, in other words, the PNF-state of a variable can go from *future* to *past* between two cycles. Therefore it is necessary to define a new time expansion function, $\tau_b : m \rightarrow M$, that handles those situations:

$$\tau_b(past) = P$$
$$\tau_b(now) = PN$$
$$\tau_b(future) = PNF$$

Notice that the only difference between the two time expansion functions is the expansion of the *future* state, which in the τ_b case also allows the *P* value. Like in the previous case we can define an expansion function for PNF-states and for component domains, $\mathrm{T}_b : U \rightarrow U$. It is straightforward to see that proposition 3.6 also holds for this function. To simplify notation, for the rest of this section we will just refer to the time function T_m. Unless otherwise mentioned, all the results apply to both expansion functions.

It is easy to see that proposition 3.6 can be expanded to sets of unary constraints over multiple nodes. The following proposition asserts that if the projection at time $\bar{t}$ of the unary constraints are contained in component domain $W^{\bar{t}}$, then they are contained in any subsequent time expansion of $W^{\bar{t}}$.

Proposition 3.7 Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ be a collection of unary constraints in an IA-network *X*. Suppose the PNF-projection at time $\bar{t}$ of σ, $S^{\bar{t}}(\sigma) = \left(S^{\bar{t}}(\sigma_1), S^{\bar{t}}(\sigma_2), \ldots, S^{\bar{t}}(\sigma_n)\right)$ is contained in a component domain $W^{\bar{t}}$, $S^{\bar{t}}(\sigma) \subseteq W^{\bar{t}}$. Then, if we define $W^{t+1} = \mathrm{T}_m\left(W^t\right)$, for $t \geq \bar{t}$, it is true that

$$\forall\, t \geq \bar{t} \qquad S^t(\sigma) \subseteq W^t$$

Proof. By induction on *t*. Consider initially the case where $t = \bar{t} + 1$. From proposition 3.6 it is clear that $S^{\bar{t}+1}(\sigma) \subseteq \mathrm{T}_m\left(S^{\bar{t}}(\sigma)\right)$. But since T_m is conservative, $\mathrm{T}_m\left(S^{\bar{t}}(\sigma)\right) \subseteq \mathrm{T}_m\left(W^{\bar{t}}\right)$, that is

$$S^{\bar{t}+1}(\sigma) \subseteq \mathrm{T}_m\left(S^{\bar{t}}(\sigma)\right) \subseteq \mathrm{T}_m\left(W^{\bar{t}}\right) = W^{\bar{t}+1}$$

A similar argument makes clear that the step of induction is true, finishing the proof.◆

Notice that, in particular, if the set of unary constraints $\overline{W}^t$ is equal or contains the minimal domain of the IA-network, proposition 3.7 assures that time expansion always yields a component domain that still contains the minimal domain of the whole network. Given this, we are now in position to put the concept of time expansion together with PNF-restriction, obtaining a general method to propagate and constrain PNF-states through time.

3.6.2 PNF Propagation: Definition

To finish the theoretical structure of our work we will define a method, called *PNF propagation*, that propagates temporal constraints through a sequence of events considering the available information about the current state and the influence of all the past states of the system. Initially, let us show that PNF-restriction behaves well through different events.

Proposition 3.8 Suppose that at an initial time $\bar{t}$ the projection of a collection of unary constraints σ is contained in an initial state $W^{\bar{t}}$, $S^{\bar{t}}(\sigma) \subseteq W^{\bar{t}}$. For any subsequent $t \geq \bar{t}$, we define $W^{t+1} = R\left(\mathrm{T}_m\left(W^t\right)\right)$. Then, the projection of the minimal domain under σ, $\overline{\sigma}$, is contained in W^t, for all $t \geq \bar{t}$,

$$S^t\left(\overline{\sigma}\right) \subseteq W^t$$

Proof. By induction on *t*. First, observe that in the case of $\bar{t}$, $S^{\bar{t}}\left(\overline{\sigma}\right) \subseteq S^{\bar{t}}(\sigma) \subseteq W^{\bar{t}}$, making the base of the induction true. Let us now suppose that the proposition is true at time t, $t \geq \bar{t}$, that is $S^t\left(\overline{\sigma}\right) \subseteq W^t$. Given that $S^{\bar{t}}(\sigma) \subseteq W^{\bar{t}}$ we can apply proposition 3.7, obtaining

$S^{t+1}(\sigma) \subseteq \mathrm{T}_m(W^t)$. From proposition 3.3 we have that $S^{t+1}(\overline{\sigma}) \subseteq R(S^{t+1}(\sigma))$, and combining both statements, given that *R* is conservative, we obtain

$$S^{t+1}(\overline{\sigma}) \subseteq R(S^{t+1}(\sigma)) \subseteq R(\mathrm{T}_m(W^t)) = W^{t+1}$$

finishing the proof.◆

Proposition 3.8 basically states that if the first component domain $W^{\bar{t}}$ contains the projection of the minimal domain of the IA-network, the minimal domain remains contained in the result of the repeated application of time expansion and restriction. However, no information from the current instant of time is added and, in practice, the tendency is that W^t ends up containing only states with value PNF. The next proposition provides a safe mechanism by which new information can be added at every instant of time while keeping the minimal domain inside the result.

Proposition 3.9 Suppose that at an initial time $\bar{t}$ the projection of a collection of unary constraints σ is contained in an initial state $W^{\bar{t}}$, $S^{\bar{t}}(\sigma) \subseteq W^{\bar{t}}$. For any subsequent $t \geq \bar{t}$, we define $W^{t+1} = R(\mathrm{T}_m(W^t) \cap V^{t+1})$ where V^{t+1} is a component domain that satisfies $S^{t+1}(\overline{\sigma}) \subseteq V^{t+1}$, $\overline{\sigma}$ being the minimal domain of σ. Then, the projection of $\overline{\sigma}$, is contained in W^t, for all $t \geq \bar{t}$,

$$S^t(\overline{\sigma}) \subseteq W^t$$

Proof. By induction on *t*. The base of induction follows immediately from proposition 3.8. Also, the same proposition provides that $S^{t+1}(\overline{\sigma}) \subseteq \mathrm{T}_m(W^t)$. By hypothesis, $S^{t+1}(\overline{\sigma}) \subseteq V^{t+1}$, and therefore $S^{t+1}(\overline{\sigma}) \subseteq \mathrm{T}_m(W^t) \cap V^{t+1}$. If we apply restriction on both sides, $R(S^{t+1}(\overline{\sigma})) \subseteq R(\mathrm{T}_m(W^t) \cap V^{t+1})$, and noticing that the minimal domain is invariant under restriction, $R(S^{\bar{t}}(\overline{\sigma})) = S^{\bar{t}}(\overline{\sigma})$, we obtain

$$S^{t+1}(\overline{\sigma}) = R(S^{t+1}(\overline{\sigma})) \subseteq R(\mathrm{T}_m(W^t) \cap V^{t+1}) = W^{t+1}$$

completing the proof.◆

Let us examine the basic mechanism that proposition 3.9 allows us to use. Typically, we set the initial component domain W^0 to be composed only of *PNF* states, $W^0 = (PNF)_i$, thus trivially satisfying the initial condition of the proposition. Then, for each instant of time *t*, we can determine, through sensor information or external sources, the PNF-state of some of the nodes.

If we create the component domain V^t containing all those known values and the *PNF* value for the other nodes, it is guaranteed that the minimal domain of the IA-network is contained in V^t, $S^t(\overline{\sigma}) \subseteq V^t$, if we assume that the sensor information is right. Then, given the previous component domain W^{t-1}, we can compute an upper bound of the current minimal domain of the PNF-network by making

$$W^t = R\left(\mathrm{T}_m\left(W^{t-1}\right) \cap V^t\right)$$

We call this process *PNF propagation.* Notice that the more information is contained in V^t, the smaller is W^t. In the extreme case, if V^t is the minimal domain, then W^t is also the minimal domain. In most cases, however, we will have V^t providing new information which is filtered through the intersection with the past information (safely provided by $\mathrm{T}_m\left(W^{t-1}\right)$). Then, information incompatible with the structure of the problem is removed by constraint propagation, through the computation of the minimal domain. Here, we also want to point out the two propositions above are also true if we use arc-consistency instead of the full minimal domain.

Like PNF-restriction, PNF propagation is also a conservative approximation of the true PNF-states of the original IA-network. Therefore, there can be situations where W^t contains values that do no correspond to possible solutions in the original network. However, as we highlighted before, if the current state W^t is inconsistent, that is, it has no solutions, then the original IA-network has also no solutions. Typically, such situations are caused either by an error in the current value of one of the sensors (making V^t incorrect) or in the value of a sensor in the past, in a situation where an incompatibility with other values is detected only in the current moment.

In the following chapters we will show examples of use of PNF propagation in different problems. In most cases we have used, in fact, the arc-consistency algorithm to compute an approximation of the minimal domain. As it will be seen in the examples, the combination of restriction, time expansion, and new information provides powerful enough constraints to actually determine the PNF-state of most of the nodes.

3.7 Future Directions

We have shown throughout this chapter that the PNF approach is a conservative approximation of solving an IA-network, that is, that projecting an IA-network and computing the minimal domain in the corresponding PNF-network always contains all the solutions of the original network, although the converse is not true. We would like to investigate in which conditions the minimal domain of an IA-network and its PNF projection coincide. In particular, we would like to find situations where if the original domains respect some determined structure or property then the converse of proposition 3.3 is true.

Similarly, we want to determine conditions in which the minimal domain of a PNF-network is equal to the result of the arc-consistency algorithm. In subsection 3.5.2 we listed some preliminary results in this direction where we identified how often the two algorithms provide different results in 3- and 4-node networks. It would be extremely useful to identify properties of the PNF-network that guarantee the equality, especially if the properties are based only on the temporal constraints (and not in the unary constraints) and therefore, possible to be checked before run-time.

Recently we have also got interested in comprehending better the space of generic PNF-networks. As mentioned in subsection 3.4.5, there are constraints that are representable in PNF-networks that do not correspond to any set of primitive Allen relationships. For instance, consider the constraint $\langle N, PF, N \rangle$. This constraint requires that when one of the intervals is not occurring, the other interval is happening, similar to the not ($\neg$) operator in logic. The question is to determine what kind of constraints — besides temporal constraints — can be represented in the PNF space (or in another of its subsets) without corrupting the semantics of temporal relationships.

Another fundamental issue is the reduction of the brittleness of PNF-networks, especially by allowing uncertainty measures and probabilities of occurrence. In the next chapter we propose some ideas for the special case of action recognition in the presence of sensor error. However, we are also interested in the more general problem of how to introduce uncertainty in constraint propagation without exploding the complexity of the algorithms.

3.8 Summary

In this chapter we discussed the advantages of using IA-networks for the representation of temporal structures. In particular, IA-networks meet the criteria we set in the beginning of the chapter for a good representation for time. IA-networks can represent imprecise and uncertain information about time; persistence is a natural consequence of the use of intervals instead of events; and they can represent mutually exclusive actions.

However, constraint propagation algorithms in IA-networks are, in most interesting cases, of exponential complexity. To overcome this limitation of IA-networks we propose the use of PNF-networks, a simplification of IA-networks where each node only assumes three values — *past*, *now*, or *future*.

We showed how IA-networks can be projected into PNF-networks and that all solutions in the IA space are also present in the PNF space, although the converse is not true. To perform constraint propagation in PNF-networks and, in particular, to compute the minimal domain, we suggest the use of an arc-consistency algorithm. Although arc-consistency is just a conservative approximation of the minimal domain, we presented reasons by which we expect that, in practice, both computations will yield the same results most of the time.

Finally, we provided the theoretical ground for PNF propagation, a method to compute the PNF-state of the nodes of an IA-network in consecutive instants of time that considers past occurrence of actions. In particular, we have determined conditions in which the use of PNF

propagation provides a conservative approximation of the true state of the network, that is, it produces a component domain that contains all the solutions of the original network. Therefore, if the result of the PNF propagation does not have solutions, there are also no solutions in the original network. In the next chapters we exploit the findings of this chapter when examining the actual use of PNF propagation in methods to handle two kinds of situations: human action recognition and interaction control.

4. Action Recognition using PNF-Networks

The goal of this chapter is to show how IA-networks can be employed to represent the temporal structure of human actions and how this representation is used to detect the occurrence of actions. The fundamental assumption of our approach is that actions can be decomposed into sub-actions having among them a variety of temporal constraints. Given that some of the sub-actions can be directly detected by perceptual methods, the task is to determine whether the action — and its component sub-actions — is currently happening or not.

We propose the use of Allen's temporal primitives (as discussed in the previous chapter) to describe the rich temporal structure of human action. Most previous action recognition schemes [67, 111, 118, 159] use strict sequential definitions of actions that do not reflect the simultaneous way actions happen in everyday life. In our case we were not only able to represent parallel actions and states but also allowed the definition of mutual exclusive relations. The construction of such representations is examined in section 4.2.

For a given action, we collect all the temporal relations between its sub-actions in an IA-network (see chapter 3), pre-process it using Allen's path consistency algorithm, and then project it into a PNF-network. To detect an action we apply the PNF propagation method described in the previous chapter using as unary constraints the information gathered by perceptual routines, as detailed in section 4.3. In each instant of time we combine the previous state of the PNF-network with the sensor-derived constraints and make this the input for the PNF restriction algorithm. The algorithm then removes PNF-states that are incompatible with the temporal constraints.

We show using examples that if the temporal constraints are tight enough, the constraint satisfaction algorithm propagates the sensor-derived constraints into the higher-level sub-actions and actions and recognizes their occurrence, as described in section 0. In practice we have used the arc-consistency algorithm (described in the previous chapter and based on [90, 107]) which computes an approximation of the minimal domain in a time linearly proportional to the number of constraints. In section 4.5 we demonstrate this approach on two other examples of action detection taken from the cooking show script described in chapter 2. The examples show the impact of different temporal constraints on which actions and sub-actions are detected given information from simple sensor routines.

A shortcoming of straightforward PNF propagation is that it assumes that actions and sensor information always proceed from *future* to *now* and then to *past*. Although conceptually correct, this model makes it hard to recover from erroneous sensor information. We address the issue in section 4.7, where we propose a method for recovery from errors based on the PNF propagation of multiple threads of states. Finally, section 4.8 presents results for the recognition of actions in an interactive environment, the installation *"It"* (described in detail in chapter 7).

Traditionally action *detection* is distinguished from action *categorization*. Action detection concerns the problem whether in a given instant of time an action is happening or not. Action categorization is the problem of determining which action of a set of actions is happening in a given moment of time — given that one of the actions is happening. We will use the term action *recognition* to refer to the simultaneous problem of detecting and categorizing actions. Throughout this chapter we basically examine action detection scenarios where we must recognize which of the component sub-actions is happening. As we will see, the framework we have developed can be used both for detection and for recognition.

The work described in this chapter first appeared in [134] in a different formulation. The results, except for the recovery methodology and its results, have appeared before in [135].

4.1 Limitations of Current Methods for Action Recognition

The majority of work on the machine understanding of video sequences has focused on the recovery of the two-dimensional optic flow or the three-dimensional motion of objects or the camera. Recently, however, emphasis has shifted to the interpretation or classification of the observed motion. In chapter 2 we presented our definition for action as movement in a context.

Recent years have witnessed a great deal of work in activity recognition and, in particular, gesture recognition [40, 165, 181]. Most commonly, those systems fully exploit the fact that gestures are linear sequences of recognizable states and the real problem is how to consider the variations of the length of time spent in each state. For instance, Darrel and Pentland [40] used dynamic programming to determine the most likely path of states in a gesture model.

The success of *hidden Markov models* (HMMs) in speech recognition (see, for instance [61]) spawned a wave of systems employing HMMs for visual action recognition. HMMS first appeared in the pioneering works of Yamato et al. [187] and Schlenzig et al. [152]. More recently they have been applied in domains like American sign language [165], performance animation [71], and T'ai Chi gestures [35]. Using HMMs enables the learning of the models using the Baum-Welch algorithm (see [140] for a good introduction). However, HMMs are probabilistic finite-state machines that grow exponentially in size with the number of parallel events and simultaneous sub-actions.

When it comes to the recognition of actions — that is, requiring contextual information — the body of work is much smaller. As discussed in chapter 2, most attempts at the logical formalization of the semantics of action (e.g. [65, 69, 149]) are based on either philosophy or formal logic. These systems are typically not grounded in perceptual primitives and have not been used in real action recognition systems.

Mann et al. [97] developed a system to recognize simple visual actions without any temporal information, mostly based in contact information. Although interesting, and as noted by Bobick [20], this is a limited approach for action recognition. Other works in the vision community [81, 111, 159] have attempted to incorporate logical definitions of time into perceptual mechanisms. However, these systems are unable to cope with most of the complex time patterns of everyday actions that include external events, simultaneous activities, multiple sequencing possibilities, and mutually exclusive intervals [6]. For example, Kuniyoshi and Inoue [81] used *finite automata* to represent actions when performing simple actions to teach a robot. Implicit in the model is the assumption of strictly sequential sub-actions which, although adequate for the mapping into robot primitives, is a strong restriction for the representation of generic human actions.

Similarly, Nagel [111] used *transition diagrams* to represent driver's maneuvers in a highway but provided no means to represent overlapping activities unless they are generalizations or specializations of each other. Siskind's approach [158, 159] used an *event logic* to represent basic actions that are temporarily connected using Allen's primitive relationships. However, in addition to being a fully exponential modal logic, Siskind's temporal propagation method using *spanning intervals* is computationally inefficient.

Probabilistic methods for visual action recognition have been proposed in the computer vision community [27, 67, 118]. The work of Brand et al. with *coupled HMMS* (CHMMs) [27], later applied to the problem of surveillance [118], tried to overcome part of the problem by providing a model in which two HMMs can be trained and run in parallel with mutual influences. The problem here is that some of the improvements do not scale for situations with three or more parallel actions. Further, we believe that it is important to exploit the fact that logical impossibilities prevent the occurrence of some sequences of sub-actions and that incorporating such constraints into action recognition systems can significantly improve their performance. As shown by our results, the addition of a single temporal constraint between two intervals can dramatically increase the recognition capability of the detection system.

Bobick and Ivanov [22] used *stochastic grammars* to represent and recognize human actions. The idea has been applied to recognition of a conductor's gestures [22] and, more recently, to detect abnormal actions in a parking lot [67]. Although context-free grammars are more expressive than HMMs — since context-free grammars can represent recursive structures — they still lack the ability to handle parallel actions. Like their finite-state models, probabilistic grammars can only recognize **sequences** of sub-actions.

4.2 Representing the Temporal Structure

To represent the temporal structure of an action we use an *interval algebra network* [4], or simply an *IA-network*. As detailed in the previous chapter, an IA-network is a constraint satisfaction network where the nodes correspond to time intervals and the arcs correspond to binary temporal constraints between the intervals. The temporal constraints are expressed using Allen's interval algebra [4], which employs disjunctions of the 13 possible primitive relationships between two time intervals. The primitive relations are equal (*e*), before (*b*),

meet (*m*), overlap (*o*), during (*d*), start (*s*), finish (*f*), and their inverses, *ib*, *im*, *io*, *id*, *is*, and *if* (see chapter 3 for the definition of these relationships).

In our methodology, we start by identifying the different sub-actions and states of an action following the general approach described in chapter 2. These sub-actions and states are identified by the nodes of an IA-network. Next, we examine the definition of the action and determine temporal constraints that exist between the different components. For instance, grasping a bowl always precedes the act of lifting the bowl. Such identified temporal constraints are translated into disjunctions of Allen's primitive relationships and the resulting constraints are associated to the corresponding nodes of the IA-network.

After determining the temporal constraints between the basic sub-actions and states of an action, we then define another set of nodes in the IA-network corresponding to the state of the available sensors. Nodes associated with perceptual routines are referred in this chapter as *detectors*. The detectors are then connected to the other nodes of the network by temporal constraints representing the situations when they occur.

The basic structure of an action can be used by systems with different sets of sensors. A new set of sensor routines can be added by simply establishing the appropriate temporal relations between the detectors and the nodes representing the sub-actions and states of the action. All the temporal relationships are determined by considering normal occurrences of the actions and not considering sensor malfunction. Later in this chapter we propose a mechanism for handling a limited amount of wrong information coming from sensor routines.

It is important to consider the adequacy of the initial specification of the action, i.e., is the specification sufficiently constrained to recognize the action given the sensors? This problem is discussed later, but a basic result of the method developed here is the ability to determine when a description is insufficient. In order to clarify our approach let us examine a concrete example involving the description of an action of *"picking up a bowl from a table"*.

4.2.1 Picking Up a Bowl

Figure 4.1 shows the representation for the temporal structure of a *"pick-up bowl"* action from the cooking show script described in chapter 2. In the figure, the node *pick-up-bowl* corresponds to the temporal interval where the action of picking up the bowl is occurring. This action is decomposed into two sub-actions corresponding to the nodes *reach-for-bowl* and *grasp-bowl*.

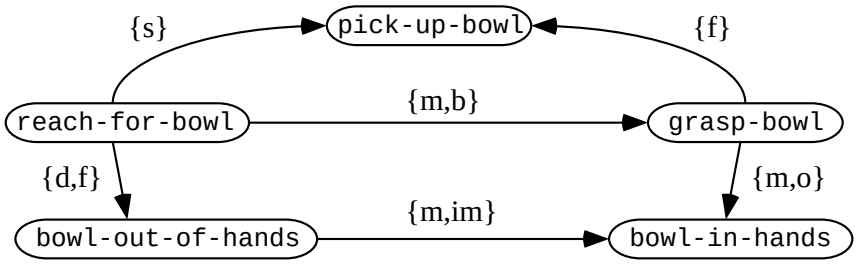


Figure 4.1 IA-network corresponding to the temporal structure of a *"pick-up bowl"* action.

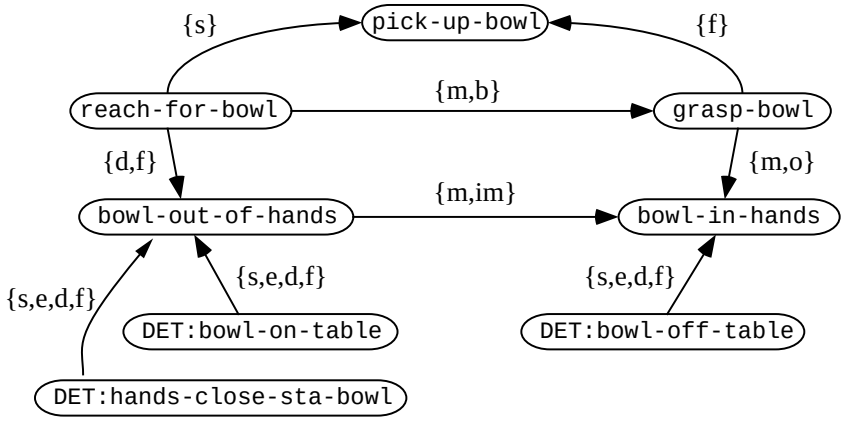


Figure 4.2 IA-network of fig. 4.1 connected to detectors (marked by the prefix *"DET:"*).

The relations between these three nodes are defined by the arcs connecting them. The sub-action *reach-for-bowl* is declared to be a time interval which has the same beginning as *pick-up-bowl*, but finishes first — the *{s}* relationship. Similarly, *grasp-bowl* finishes at the same time as *pick-up-bowl*, *{f}*. The relationship between *reach-for-bowl* and *grasp-bowl* is defined considering two possibilities: they either immediately follow each other or they happen in a sequence, though they are always disjoint as represented by the constraint *{m, b}*.

The next level of decomposition involves two complementarily predicates (written as *{m, im}*) encoding the physical relation between the bowl and the hands: either the bowl is in the hands of the agent, *bowl-in-hands*, or not, *bowl-out-of-hands*. Notice that the constraint *{m, im}* states that the end of one action meets the beginning of the other or, in other words, that they are mutually exclusive. Allen's algebra was chosen as the underlying temporal formalism because we consider the expressing of mutually exclusive actions and states a fundamental capability of an action representation paradigm. For this reason, we can not use temporal algebras based on relations between endpoints (like [48, 178]), despite the good performance of the reasoning methods for point-algebras as discussed in the previous chapter.

Finally, the fact that reaching for the bowl must happen while the bowl is not in contact with the hands is expressed by the *{d, f}* relationship between *reach-for-bowl* and *bowl-out-of-hands*. Similarly, *bow-in-hands* starts during *grasp-bowl* or immediately after its end (*{m,o}*).

4.2.2 Connecting to Perceptual Routines

So far the structure of our *"pick up bowl"* example contains only sub-actions and states that are intrinsically part of the action. To have a perceptually-based system recognizing the action, it is necessary to connect the nodes of the action's network to detectors. Detectors, in our notation, are always marked by the prefix *"DET:"*.

For example, suppose in a vision-based system routines there are routines that detect when the user's hands are close to the bowl and also whether the bowl is on the table or not. Let us associate to the node *DET:hands-close-sta-bowl* the result of the first perceptual routine that detects if the hands are close to the bowl (only in the cases that the bowl is static and on the table). Similarly, we associate to the routine that identifies the presence of the bowl on the table two other detector nodes, *DET:bowl-on-table* and *DET:bowl-off-table*.

Figure 4.2 displays the resulting network. From the definition of the action and of the detectors, there are clear temporal constraints connecting the detectors to some sub-actions and states of the *"pick up bowl"* action. In this case, *DET:hands-close-sta-bowl* and *DET:bowl-on-table* can fire only when the bowl is out of the hands. This corresponds to the temporal constraint *{s,d,e,f}* between each detector and *bowl-out-of-hands*, i.e., the interval of time corresponding to the occurrence of the detectors must be contained (maybe including the endpoints) in the interval of time of *bowl-out-of-hands*. On the other hand, *DET:bowl-off-tabl*e can only happen while the bowl is being held, as expressed by the constraint *{s,d,e,f}* with *bowl-in-hands*.

4.2.3 Strengthening the Constraints

A clear benefit of using IA-networks for representation is the possibility of computing the transitive closure to detect temporal constraints that are implied by the stated constraints. As discussed in chapter 3, in practice we employ Allen's path consistency algorithm that, although not removing all inconsistencies, computes a reasonable approximation of the transitive closure in time proportional to the cube of the number of constraints.

Figure 4.3 shows part of the IA-network of *"pick up bowl"* after computing path-consistency, where the nodes corresponding to detectors are omitted for clarity. As seen in the figure, path-consistency tightened the temporal relationships of the network. For instance, the original constraint that *reach-for-bowl* is followed by *grasp-bowl* (*{m,b}*) is propagated and force *bowl-out-of-hands* to accept just a meet (*{m}*) relationship with *bowl-in-hands* — instead of the original *{m,im}*.

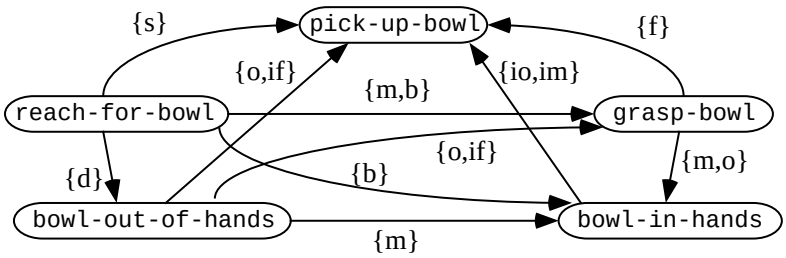


Figure 4.3 Part of the IA-network of the *"pick up bowl"* action as generated by Allen's path consistency algorithm. The nodes corresponding to detectors are omitted for clarity.

Allen's path consistency algorithm allows the system designer to avoid the laborious and tedious task of manually encoding all the possible relationships. Moreover, in our use of the system, we found that many times the result of the algorithm makes visible for the system developer mistakes and errors in the definition of the temporal structure of the action. Typically, mistakes are detected either by the detection of inconsistencies or by the unexpected removal of some primitive relation from the disjunctive set of relations between two nodes.

4.2.4 Automatic Inference of Temporal Constraints

Describing the temporal structure of an action using IA-networks is not a trivial task and involves some training of the designer/programmer. However, in our experience, we found that after some basic rules were understood, specification of temporal relations became fairly straightforward.

Further, we also believe that some of the temporal relations could be automatically derived by simple inference systems if enough information about the action itself was available. For instance, many of the relationships defined in the *"pick up bowl"* example do not involve "deep" common-sense reasoning. For example, *bowl-in-hands* and *bowl-out-of-hands* are temporally mutually exclusive because, by definition, they represent logically opposite states of the bowl. As described in chapter 2, it is possible to generate the decomposition of actions into sub-actions using "shallow" common-sense reasoning. We believe that temporal constraints similar to the ones used in the representation of the *"pick-up bowl"* action can also be automatically generated by a system composed of a dictionary of basic actions and simple reasoning.

4.3 Recognition Using PNF-Networks

In this section we describe the basic structure of our recognition method that is based on the *PNF propagation* method (formally defined in the end of chapter 3). PNF propagation is a method of detecting the occurrence of actions based on intersecting the information from the sensors with the time expansion of the component domain of PNF-states representing all the past information.

When using PNF propagation for action detection, we consider the computed PNF-state of each node to determine whether the action is happening. If the PNF-state of the node is *now*, we can

say that the action is happening; if it is *past*, *future*, or either of them (*PF*), the action can be said to be not happening. Also, if the PNF-state of a node is NF, we conclude that the action has not finished yet and if it is PN, that it has not started. Otherwise (that is, *PNF*), we assign an indeterminate label to the node.

Although PNF propagation has been described in the end of chapter 3, we summarize below the basic steps as they are implemented in our test set-up:

I. The IA-network representing the action is projected into a PNF-network.

II. For each instant of time *t*, a component domain W^t is computed that corresponds to the PNF-state of each node at time *t*. Based on the PNF-state of each node, it is possible to determine the occurrence of an action as described above.

   i. The initial state W^0 is defined according to the information available about the nodes or simply with every state equal to *PNF*, $W^0 = (PNF)_i$.

   ii At each subsequent instant of time, $t > 0$, W^t is computed by the following sequence of steps:

   - Information from the detectors (perceptual routines) is gathered in a component domain V^t such as

$$V_i^t = \begin{cases} \text{PNF - state of the detector } I_i \text{ , if } I_i \text{ is a detector} \\ PNF \text{ , otherwise} \end{cases}$$

   - The current state of the nodes is determined by simple PNF propagation,

$$W^t = AC\left(\mathrm{T}_m\left(W^{t-1}\right) \cap V^t\right)$$

   where T_m is the time expansion function (based on meet relations) and AC is the arc-consistency algorithm as defined in chapter 3.

Typically, the information coming from a detector is either *N* or *PF*. In other words, the PNF-state corresponds to the detector being either on or off. Also, it is necessary to time expand the component W^{t-1} before intersecting it with the perceptual information V^t, since between instant *t-1* and *t* actions may have ended or begun. In fact, the experimental results of the next section demonstrate that using past information is a fundamental component of the power of PNF propagation.

Finally, if we compare the above definition of W^t to the hypothesis of proposition 3.9 of chapter 3 we can see that they are identical. Therefore, if we assume that the detectors are corrected all the time, then the proposition holds and thus it is true that for every instant of time *t*, W^t contains the true PNF-state of each node of the IA-networks.

Figure 4.4 Images from the video used in the experiments.

4.4 Temporal Constraints vs. Detection Power

By representing actions by IA-networks we provide a framework where the representation is independent of particular perceptual routines. For each perceptual system, the different available detectors can be linked to the representation through temporal constraints. However, in practical situations there is no guarantee that the perceptual routines have enough discriminatory power to actually detect the action.

Similarly, a common problem in action recognition and especially in vision-based systems is that most of the information gathered by the perceptual routines is indirect. For instance, current camera resolutions prevent the detection of contact between two objects forcing the use of the weaker notion of proximity instead. Also, it is not clear in our formulation how many and how strong the temporal constraints need to be and how many different sub-actions and states are needed to fully describe an action.

There are no unique answers to these questions. Our experiments have convinced us that IA-networks provide an excellent framework in which representations are easily experimentally defined and refined. Moreover, unlike learning-based methods like HMMs, the programmer of the application has a clear representation of what the system knows, what can be detected, and the assumptions used to enhance detection.

This section describes how representing the temporal structure with IA-networks and detecting actions using PNF propagation facilitate the handling of these issues. We discuss them by considering different IA-networks to detect the action *"pick up bowl"*. To test the expressive power of the different IA-networks, we tested them with manually extracted values for the sensors. We obtained these values by watching a video of the action being performed. We also determine the interval where every action and sub-action has actually happened (the *"TRUE:"* state of the sub-action) and use the information to evaluate the performance of the action detection method.

Typical images of our video sequences are shown in fig. 4.4. We did not implement the perceptual routines described above because using manually extracted information allows the simulation of sensors with different levels of correctness and a more careful analysis of

coincidences. Later in this chapter we will present examples of action detection with machine generated perceptual data.

4.4.1 Visual Representation of PNF States

For the display of the results we employ a visual representation for PNF states based on different symbols for each possible state. As shown in the legend of fig. 4.5 we use the convention that the bottom line represents the *future* state, the middle represents *now*, and the top, *past*. For instance, the *PF* state is represented by lining the bottom and the top row simultaneously, and *PNF* is displayed as three parallel lines. Also, we highlight the unitary PNF-states *P*, *N*, and *F* to clearly mark moments when the detection system reaches only one possible situation.

4.4.2 Weaknesses of the Original Representation

Let us consider the detection of the action *"pick up bowl"* using the PNF propagation method as described above. In this section we presume perfect sensors; a fault-tolerant extension of these ideas is described later.

The top part of fig. 4.5 displays the temporal diagrams for the PNF states of the detectors (marked as *"DET:"*) and the true state of all other sub-actions and states (marked as *"TRUE:"*) for a particular instance of the action of picking up a bowl. As explained above, the data were obtained manually from a video depicting the action.

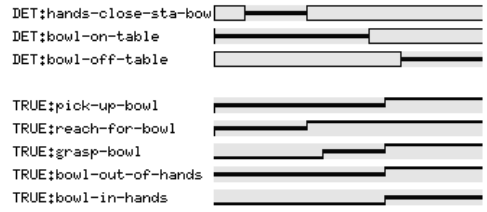
Item (a) of fig. 4.5 shows the results when the detection process uses the description of the *"pick-up bowl"* action exactly as given in fig. 4.2. Only the physical states *bowl-in-hands* and *bowl-out-of-hands* are close to being detected. The main action, *pick-up-bowl*, is never detected. However, notice that in the initial period the method determines that the action may have started but it is not finished (by the value *NF*). This is followed by a period of complete indeterminacy (*PNF*), but after *DET:bowl-off-table* becomes *N*, it is detected that the action *pick-up-bowl* is happening or has already happened (*PN*).

The problem is that the definition of *"pick-up bowl"* of fig. 4.2 has constraints that are too weak, although always true. We intentionally constructed a weak example because it illustrates clearly some of the extra assumptions that are needed to link states detected by simple sensors to actions and sub-actions.

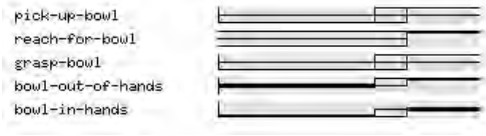
4.4.3 Causal Links

One of the problems is that the original definition of "pick-up bowl" lacks any causal link between detecting that the bowl is not on the table and the result of the act of grasping. If we assume that movements of the bowl can not be caused by other agents, objects (the bowl could be pushed by a spoon), or forces (gravity), we can include a constraint imposing that the only cause for movement of the bowl is a grasping action by the agent of grasping. This can be accomplished by setting the relationship between *grasp-bowl* and *DET:bowl-off-table* to be sequential and non overlapping *{b,m}*. Item (b) of fig. 4.5 shows that with the addition of such relation the end of *pick-up-bowl* is detected. This happens as the result of the detection of the

Detectors (DET:) and true state (TRUE:)

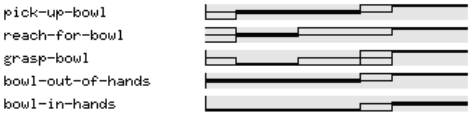


a) Original pick-up bowl representation (as in fig. 4.2)

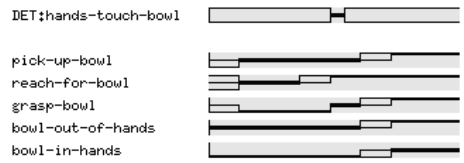


b) Addition of a new relation:
grasp-bowl {b,m} DET:bowl-off-table

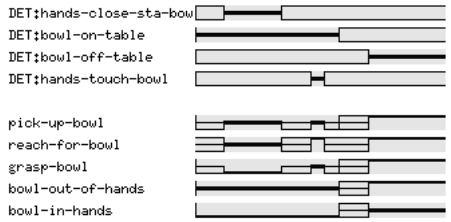
c) Addition of a new relation:
DET:hands-close-sta-bowl {s,e,d,f} reach-for-bowl



d) Addition of a new detector:
DET:hands-touch-bowl {s,e} grasp-bowl



e) Using sensor information without time propagation:



LEGEND:

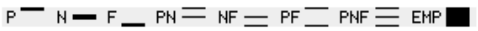


Figure 4.5 Influence of the temporal structure on the detection of the action *"pick-up bowl".*

end of *grasp-bowl*, that is now possible due to the association with the detector of the bowl being off the table.

To detect the beginning of *pick-up-bowl*, it is necessary that the action description includes some causal relationship about the beginning of the sub-action *reach-for-bowl*. One way to do this is to indicate that the proximity between hands and bowl (as detected by *DET:hands-close-sta-bowl*) is an indicator for the occurrence of *reach-for-bowl*. In this situation, we impose that whenever the hands are detected close to the bowl, the sub-action *reach-for-bowl* is supposed to be under way. By doing this, we are assigning a relationship that may not be always true. However, given the simplicity of our sensors (and of most state-of-art vision-based algorithms), such "intentional" links are necessary to detect higher level actions. The results, shown in item (c) of fig. 4.5, display the almost complete detection of *pick-up-bowl* and *reach-for-bowl*.

4.4.4 Adding an Extra Detector

Finally, if we also want to detect the beginning of the occurrence of *grasp-bowl*, a new detector is necessary. This is shown in itemt (d) of fig. 4.5, which displays the temporal diagram of a new sensor, *DET:hands-touch-bowl*. This sensor fires precisely when the hand touches the bowl. This detector, coupled with the assumption that touching the bowl always starts the action of grasping the bowl, is sufficient to assure the detection *of grasp-bowl*. As we can see in this last case, the states of the sub-actions are known in the majority of the time and are correct (compare to the *TRUE:* diagram at the top of figure 4.5).

4.4.5 The Influence of Past States

In the beginning of chapter 3 we claimed that the ability to represent time with three states (i.e., *past*, *now*, or *future*) significantly improves detection — thus justifying some computational overhead. Item (e) of fig. 4.5 shows the importance of the information from the previous instant of time on the power of PNF propagation. In this case, W^t is computed solely based on the information from the sensors, with no reference to past states, that is, $W^t = AC\left(V^t\right)$. Comparing item (e) of fig. 4.5 with item (d), we can see a distinct degradation in the results. The main reason is that after a cause for a sub-action or state being in the *now* state ceases to exist, the system still considers that the sub-action can still happen later in the *future* (compare, for instance, the detection of *pick-up-bowl* in both cases).

4.5 Detector Instances and Detectors

Before further examining results, it is necessary to understand and solve a technical difficulty. The basic problem is that during the occurrence of an action a perceptual routine may detect the occurrence of a state several times. For instance, in the "*mixing ingredients*" action described in the next section, we use a vision routine that fires when the hands of the chef get close to the mixing bowl. This detector fires both when the chef picks up the bowl and when he puts it down.

Suppose that we associate the same detector — called in this case *DET:hands-close-bowl* — to both sub-actions by imposing, as we did in the previous section, that the detector must happen

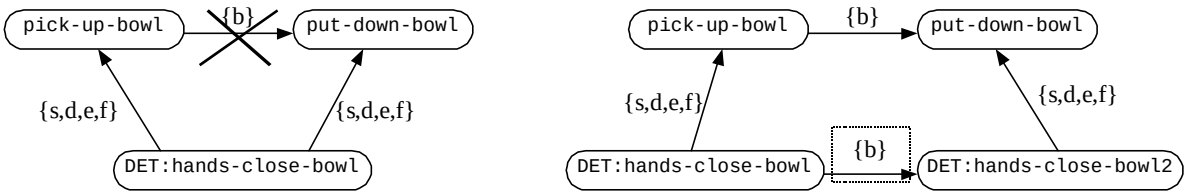


Figure 4.6 An inconsistent association of a detector to two intervals solved by using two different instances of the same detector.

during both sub-actions using the temporal constraint *{s, d, e, f}*. However, *"picking up bowl"* and *"putting down bowl"* are sub-actions that must occur one after the other, that is, there is a temporal constraint *{b}* between them. As fig. 4.6 shows, the imposition of these three temporal constraints lead to an inconsistent IA-network. In fact, imposing these constraints require *DET:hands-close-bowl* to happen in two intervals of time that are distinct.

The issue here is that the on-off behavior of the perceptual routine is not adequately modeled by a single node in an IA-network. As discussed in chapter 3, a node in an IA-network corresponds to a single interval of time, while the occurrence of a detector corresponds to multiple segments of time.

A solution for this problem is to associate multiple nodes in the IA-network to the same detector and consider those nodes to be different *instances* of the same detector. The right side of fig. 4.6 displays an IA-network for the case described above where we employ two instances of the same detector to avoid the inconsistency, named *DET:hands-close-bowl* and *DET:hands-close-bowl2*. Notice that since the nodes *pick-up-bowl* and *put-down-bowl* have the *{b}* temporal constraint between them, it is possible to infer (using Allen's path-consistency algorithm) that the two instances of the detector must also follow one another. That is, there is also a temporal constraint *{b}* between them (shown inside the dotted rectangle in fig.4.6).

With this example we also want to make clear that in an IA-network each node represents a single occurrence of the sub-action associated to it. Notice that multiple instances of the same detector are not required because of detector's misfiring, but only in situations where the same detector is supposed to occur more than one time during the occurrence of the same action. The issue of sensor malfunction is discussed later in this chapter.

All the reasoning presented in this section also applies to sub-actions and states that have multiple occurrences within the same action. As with the detector case, it is necessary to rely on defining multiple instances of the sub-actions and to constrain them appropriately. In the next chapter we present — in the context of representing interaction — a mechanism by which an arbitrary number of repetitions of the same sub-action or detector can be handled.

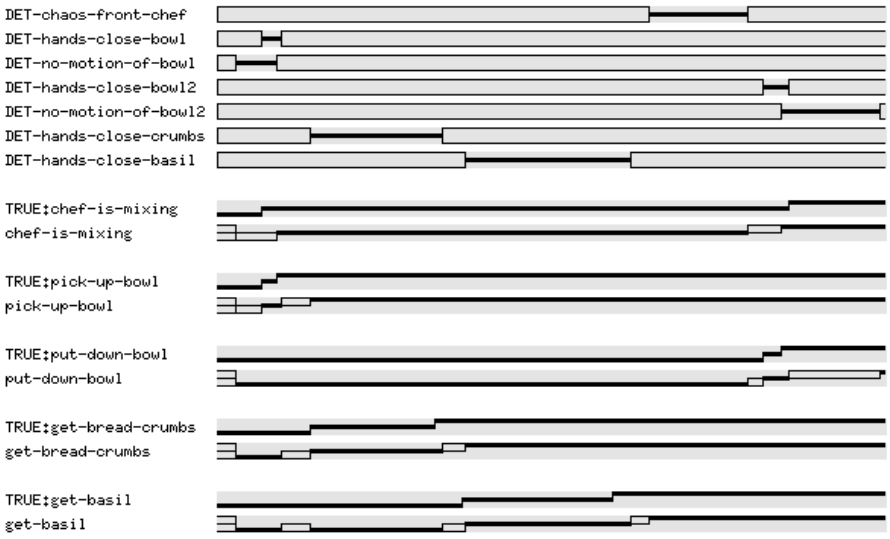


Figure 4.7 Detection of the action “mixing ingredients”.

4.6 Other Examples of Results

Having discussed the influence of the temporal constraints on the detection power, it is useful to examine more complex actions. In our experiments (described in [135]) we tested the PNF propagation methodology applied to the detection of two other actions from the cooking show domain, *“mixing ingredients”* and *“wrapping chicken”*. The temporal structure of each action was codified manually into an IA-network and the values of the detectors were manually gathered from actual video sequences and provided to the detection system.

With these two examples we want to experimentally show that both the IA-network representation and PNF propagation scale well. The action *“pick up bowl”* is, in fact, a sub-action of the *“mixing ingredients”* action. In general, we have observed that the detection of actions is enhanced as the complexity of the action increases. This is due to the power of constraint propagation to spread quickly information through an entire network. In this scheme, small pieces of evidence lump together and constrain the possible states to very constrained component domains, especially when past information is available.

4.6.1 Mixing Ingredients

The complete description of the IA-network for the action *“mixing ingredients”* is found in appendix C, codified in the *action frames* format discussed in chapter 2. The action *“mixing ingredients”* is composed of four sub-actions, corresponding to different phases: first, the bowl is picked up, then basil and bread crumbs are added to the bowl (in any order), and, after the mixing is done, the bowl is put back on the table. For the sake of compatibility with the original description, the node corresponding to the action *“mixing ingredients”* is called *chef-is-mixing*.

Seven detectors are employed in this example. The detector *DET:chaos-front-chef* is a vision routine that looks for fast-paced change in the area in front of the chef's chest that corresponds to the actual mixing of the ingredients in the bowl. Whenever the chef's hands get close to the stationery bowl, *DET:hands-close-bowl* fires. The second time this happens, corresponding to the end of putting down the bowl, a new instance of the detector *DET:hands-close-bowl2* fires. Similary, *DET:no-motion-of-bowl* and *DET:no-motion-of-bowl2* are two instances of a perceptual routine that determines when the bowl is observed to be static on the table. Finally, two other detectors, *DET:hands-close-crumbs* and *DET:hands-close-basil*, detect when the hands get close to the cups containing bread crumbs and basil, respectively.

The description of *chef-is-mixing* comprises a total of 17 sub-actions plus the seven detectors. Figure 4.7 shows the results for the detection of the action using data manually extracted from a video sequence. The figure displays the detector values on the top and, for each of the five main sub-actions, the ground truth data (marked as *"TRUE:"*) and the result of the detection. As we see in fig. 4.7, the main action is detected most of the time and similar results are obtained for the four main sub-actions.

Some of the non-resolved periods are caused by the lack of appropriate sensors. For instance, in the case of *get-bread-crumbs*, there is a period just before the action starts to happen when the PNF-state is *NF*. This is caused by the fact that the detection of this sub-action relies on the proximity of the hands to the bread crumbs (*DET:hands-close-crumbs*). Before proximity is detected, the only way for the system to know that the action is not happening is the fact that *pick-up-bowl* has not finished, as determined by the IA-network describing *"mixing ingredients"* listed in appendix C. In the action definitions, there is a temporal constraint enforcing that *get-bread-crumbs* must happen after (*ib*) *pick-up-bowl*. When *pick-up-bowl* finishes, freing the constraint, there is no information to determine if *get-bread-crumbs* has started or not. Of course, when *DET:hands-close-crumbs* is detected, the sub-action *get-bread-crumbs* is immediately set to *N*. Similar situations explain most of the other cases of non-perfect detection.

4.6.2 Wrapping Chicken with a Plastic Bag

Figure 4.8 illustrates the detection of another complex action, *"wrapping chicken with a plastic bag"*, which involves 25 sub-actions and 6 detectors. The corresponding IA-network is defined in appendix C in the *action frames* format. Figure 4.8 displays the true and the recognized PNF-state of the main action and of five sub-actions, each of them with a level of complexity similar to the *"pick-up bowl"* shown above. All the sensors are very simple: proximity between hands and the box containing plastic bags (*DET:hand-close-pbag-box*) and the plate containing chicken (*DET:hand-close-chix-co*), chaotic movement in front of the subject's trunk (*DET:chaos-front-trunk*), and absence of motion in the area of the wrapped chicken (*DET:no-motion-wrap-chix*).

Notice that the main action and the five sub-actions in the example are correctly detected most of the time. The only real error happens in the detection of *put-down-wrap-chix*, which remains in the *N* state even after the action has finished. The problem is caused by the detector *DET:no-motion-wrap-chix*, which is responsible for determining the end of that sub-action. As seen by

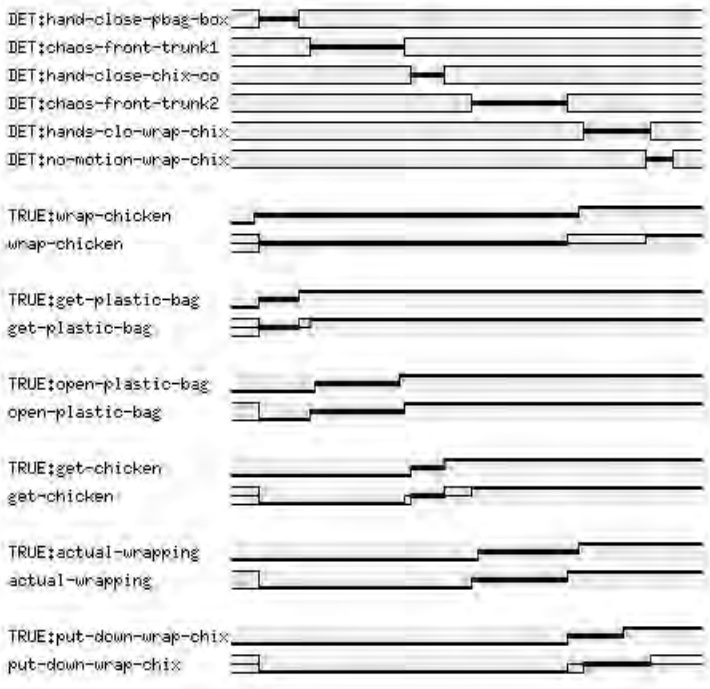


Figure 4.8 Detection of the action "wrapping chicken".

examining the other detector, *DET:hands-clo-wrap-chix*, the subject's hands remained close to the chicken after it was put down (as evidenced by *TRUE:put-down-wrap-chix*), preventing the detector *DET:no-motion-wrap-chix* from actually determining its state.

In theory, the detector *DET:no-motion-wrap-chix* should have signaled this by outputting *PNF* instead of *PF*. However, here we simulated the normal situation where a perceptual routine is not able to determine when it is failing. As fig. 4.8 shows, that does not prevent the detection process from being correct most of the time, although it could have been the case that a contradiction would completely prevent the recognition of the action. How to overcome such problems is the subject of the following section.

4.7 Recovery From Errors

Implicit in the PNF propagation method as described above is the assumption that the sensors are always correct. When computing the current component domain W^t using the previous domain W^{t-1} (time-expanded), only the possible domains allowed by the previous values of the sensors, $V^0, V^1, \ldots, V^{t-1}$, are considered. For instance, if a sensor erroneously set up a node to *P*, the proposed method has no way to recover from the error.

To overcome this limitation, we have been using a scheme where the detection system keeps track of multiple hypotheses. We first describe a method that assumes that at most one sensor is wrong at each instant of time and guarantees the recovery in such cases. The price for this ability to tolerate errors is an increase in space and time requirements and a small decrease in

detection power. Next we explore heuristic methods to deal with multiple sensor failures.We start by defining some useful notation. We note as V_0^t the values of the l sensors at instant t, $V_0^t = V^t$ and $V_1^t, V_2^t, \ldots, V_l^t$ as the component domains where V_j^t, $j = 1,2,\ldots,l$, is obtained by "inverting" the value of the j-th sensor in V^t. In other words, V_j^t is the true state of the sensors if only the sensor j had failed by reporting the inverted value. Therefore, the set $V_0^t, V_1^t, \ldots, V_l^t$ represents all the possible combinations of sensor states if at most one of the sensors fails at one instant of time.

4.7.1 Detection with At Most One Sensor Error

Our approach for error recovery relies on the construction, for each instant of time t, of a set of *error-tolerant component-domains*, Ω^t. The basic idea is to define Ω^t such as it includes all possible threads of component domains, considering that at most one sensor was wrong at any past instant of time. This is accomplished by

$$\Omega^t = \left\{ R\left(\mathrm{T}(\omega) \cap V_j^t\right) \neq \varnothing \mid \text{for all } j = 0,1,2,\ldots,l \text{ and } \omega \in \Omega^{t-1} \right\}$$

In other words, the set of error-tolerant component domains contains all non-empty restrictions of the combinations between the previous error-tolerant domains and current values of the sensors with at most one error allowed. Typically, there is total ignorance at time $t=0$, $\Omega^0 = \{(PNF)_i\}$.A potential problem is that Ω^t can increase exponentially by this method. In our tests, however, we found that the large number of repetitions and empty component domains kept the size of Ω^t almost constant. A possible explanation can be related to the observation (from [82]) that the number of consistent IA-networks is very small if the number of nodes is greater than 15. The method described above guarantees that the progression of sets of error-tolerant domains $\Omega^0, \Omega^1, \ldots, \Omega^t$ always contains the correct component domain **if at most one sensor is wrong each instant of time**. Therefore, under this constraint, we can assure that if we declare the current values of the nodes to be

$$W^t = \bigcup_{\omega \in \Omega^t} \omega$$

then W^t contains the right answer.

However, computing W^t in this manner is too conservative: many times, most of the nodes are assigned the *PNF* value. To address this problem, we propose to de-couple the building of the set of error-tolerant domains from the choice of the current values for the nodes. It is important to keep the set of all error-tolerant domains as complete as possible. However, we determined that a good policy is to make the current value W^t based on the actual readings of the sensors, considering all the different possibilities of past errors. Basically, we define W^t to be the union of restriction of all previous domains intersected only with the current value of the sensors,

$$W^t = \bigcup_{\omega \in \Omega^{t-1}} R\left(\mathrm{T}(\omega) \cap V_0^t\right)$$

Of course, if this quantity ends up being empty, then we use the overly-conservative method for computing W^t. In summary, the set Ω^t is computed considering always that one of the sensors might be wrong (plus the previous set Ω^{t-1}) but the current state W^t considers just that the sensors could be wrong in the past, but not in the present. This was found to be a good compromise between accuracy and ability to recover from errors.

4.7.2 A Heuristic for General Error-Tolerant Detection

In practice we need to be able to recover, at least partially, even in the case of more than one simultaneous error. We focused here on the situations where all the current sensor readings allow no consistent interpretation with the past information, that is, $\Omega^t = \varnothing$. In this case, we use an heuristic for recovery which completely disregards past information, considering only the possible implications of current sensor information, but still assuming that only one of the sensors is wrong,

$$\Omega^t = \left\{ R\left(V_j^t\right) \neq \varnothing \mid \text{ for all } j = 0,1,2,\ldots,l \right\}$$

If this is still an empty set, we define the set of error-tolerant domains to be the one with the least information possible, $\Omega^t = \left\{ (PNF)_i \right\}$.

4.7.3 Examples of Recovery from Errors

Figure 4.9 shows examples of the results obtained by using the method described above. As a measure of comparison, we provide in item (a) of fig. 4.9 the results obtained assuming perfect sensors (the same as in item (d) of fig. 4.5). Using the method described above, the processing of the same data produces the results in item (b) of fig. 4.9, where there are intervals of time when the system is unable to decide whether the actions are happening or not.

Surprisingly, when we introduce errors in the detectors *DET:hands-close-sta-bowl* and *DET:bowl-off-table* (but never at the same time) there is an increase in the detection accuracy, as show in fig. 4.9.c. This can be explained by considering that when a sensor has a false positive, its corresponding value of *now* in the component-domain has the effect of producing many contradictions, pruning the set of error-tolerant domains Ω^t at an early stage.

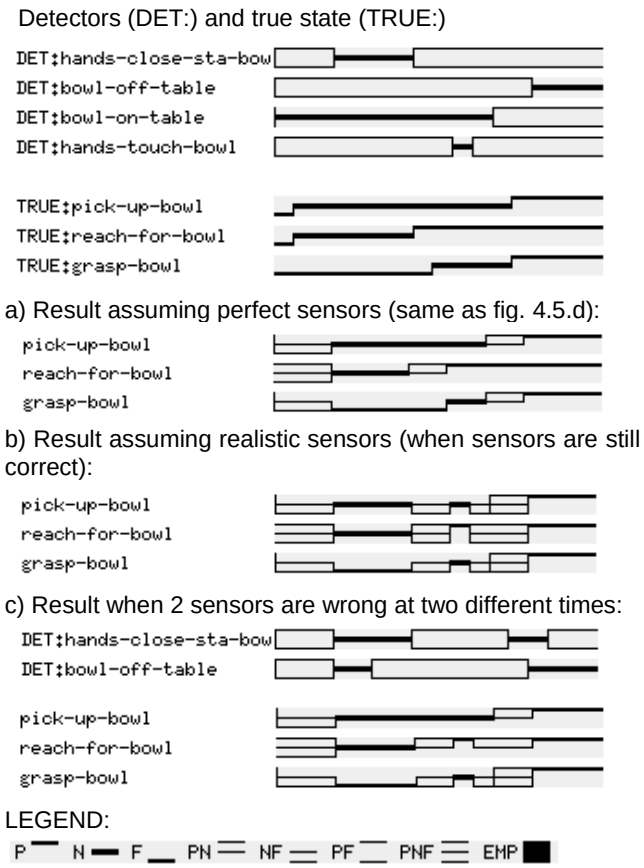


Figure 4.9 Influence of sensor errors on the detection of the action *"pick-up bowl"*.

Figure 4.10 shows other results obtained by adding erroneous information into the sensors of the *"wrapping chicken"* action of fig. 4.8. In item (a) of fig. 4.10 we repeat the result of fig. 4.8 for comparison. Item (b) shows the result when all the information from the sensors is correct but employing the error recovery system described above. Basically, the result is the same except when all the sensors are off (*PF*).

This degradation of performance when sensors are wrong is examined further in items (c) and (d) of fig. 4.10, where we observe the effects of false positives and false negatives of the same sensor, *DET:chaos-front-trunk1*, on the detection of the action *wrap-chicken*. Notice that a false *N* value improves the detection but a false *PF* value gives rise to segments of complete ignorance (*PNF*). In itens (e) and (f) of fig. 4.10 we examine two other cases showing the robustness of the method to repeated errors and to confusion between similar sensors.

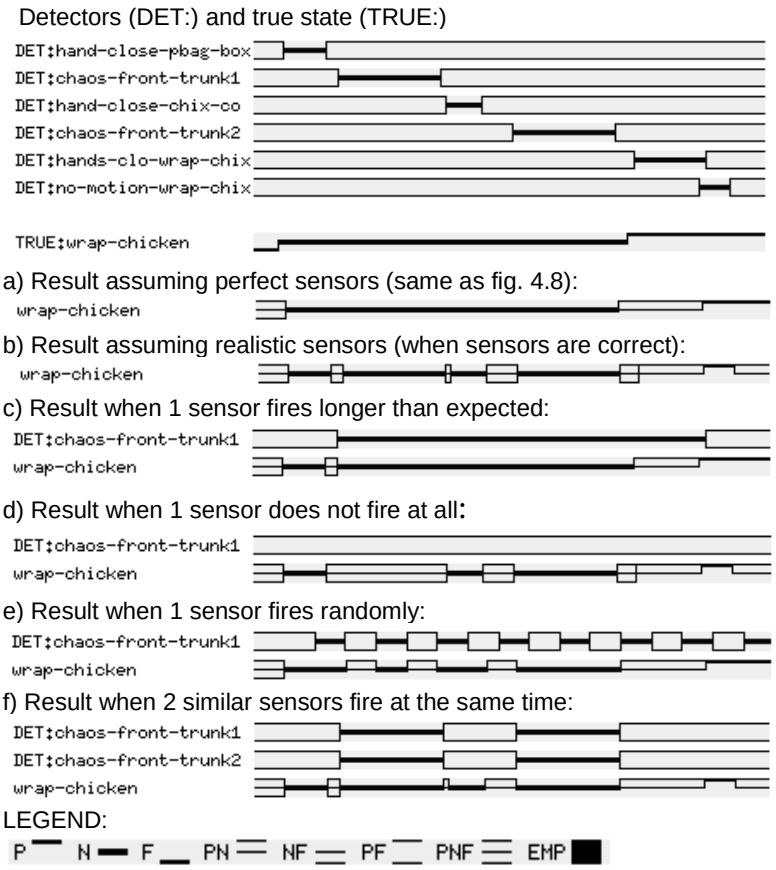


Figure 4.10 Influence of sensor errors on the detection of the action *"wrapping chicken".*

Until now all the examples have relied on manually extracted data from a single video sequence depicting the action and on hand-generated errors. The next section examines these methods running on data from perceptual routines employed in one of interactive spaces we build, the art installation *"It"* (described in chapter 7).

4.8 Experiments in the *"It"* Scenario

As part of the art installation *"It"* that we design and built in 1999 (described in chapter 7), we wanted to detect actions involving the user and wood cubes in the space. Among other actions, we examine here the case of detecting the occurrence of situations where the user picks up a cube from the floor and move it to other position in the space.

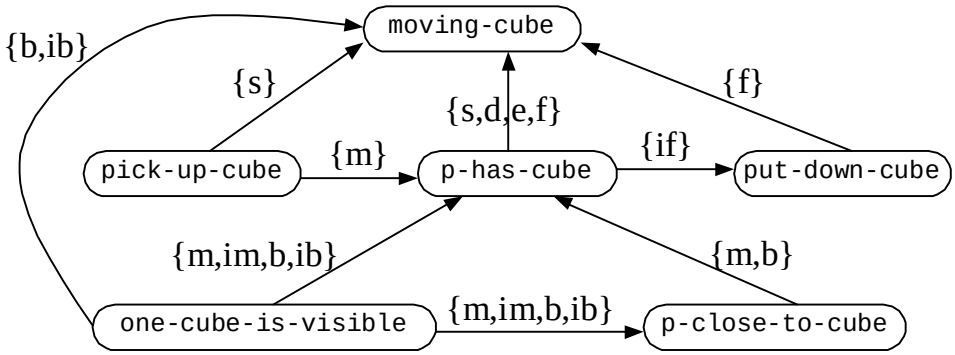


Figure 4.11 IA-network for the *"moving cube"* action.

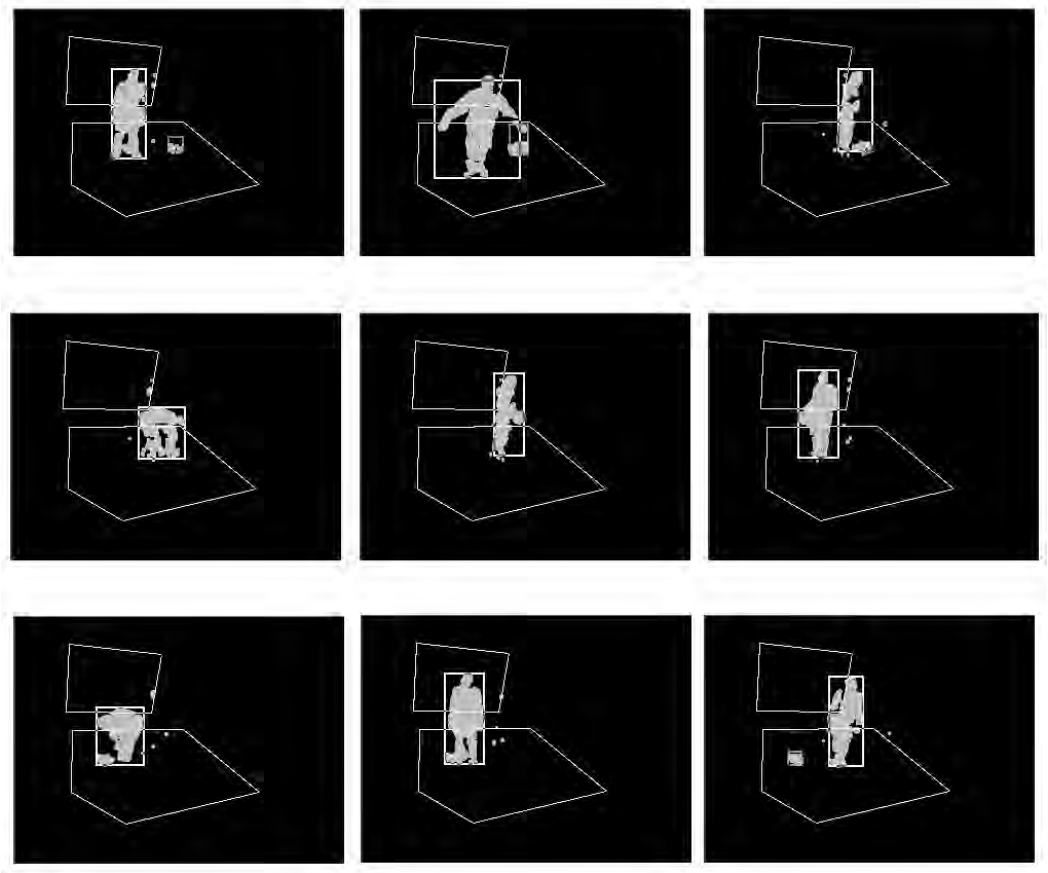


Figure 4.12 Silhouette input used in the experiments of the detection of a "moving cube" action.

The IA-network that represents the action *"moving cube"* is shown in fig. 4.11. The network is composed of six nodes, where the main action is decomposed first into three sub-actions: *"pick-up cube"*, *"person has cube"*, and *"put-down cube"*. We also associate two states that are affected by the action, the first corresponding to the situation where the cube seen can be seen as an isolated component from the person's body; and the second corresponding to the state where the person is known to be close to the cube. As we will see below, this is motivated by the fact that we are employing a silhouette-based vision system that can not distinguish whether a person is carrying a cube or not.

Unlike the examples previously used in this chapter, this experiment has been performed with sensor data obtained automatically from the vision system of *"It"*. Based on silhouette images similar to the ones shown in fig. 4.12, we implemented simple detectors for visual properties.

The implemented detectors are:

- `DET:one-cube-is-detect` : determines if an isolated cube has been detected by the low-level visual routines. This detector is linked to the state `one-cube-is-visible` of fig. 4.11 by a constraint *{s,d,e,f}*.
- `DET:p-is-shorter` : determines if the height of the person has decreased. This detector is used to detect the beginning of both `pick-up-cube` and `put-down-cube` actions, and therefore we employ two instances of it. The first, `DET:p-is-shorter` has a *{o,s,e,is}* relation with `pick-up-cube`, and the second, `DET:p-is-shorter2` is constrained by a *{s,e,is}* arc to `put-down-cube`.
- `DET:p-is-higher` : similar to `DET:p-is-shorter`, detects if the height of the person has increased. Two instances are used, the first, `DET:p-is-higher` has a constraint *{im,io}* to `pick-up-cube`, and the second, `DET:p-is-higher2`, is *{im,io}* to `put-down-cube`.
- `DET:p-sa-as-one-cube` : detects if the person is at the same area as the cube was seen for the last time. This detector is associated to the state `p-close-to-cube` through the constraint *{s,e,d,f}*.
- `DET:and-not-sa-not-det`: a detector corresponding to the AND composition of the negation of the states of two other detectors, `DET:p-sa-as-one-cube` and `DET:one-cube-is-detect`. It basically fires when the person is seen leaving the area where a cube has been seen for the last time and the cube is not seen anymore, that is, typically when the person is carrying the cube. This detector is associated to the state `p-has-cube` through a *{s,d,e,f}* constraint.
- `DET:I-is-inside` : determines if the person is inside the area. This is, in fact, a necessary condition for the action to be seen, expressed by a constraint *{is,e,id,if}* with the main action `moving-cube`.

The low-level visual routines associated to these detectors are, unfortunately, quite noisy, often producing errors. Therefore, we employ the method to recover from errors described above. To

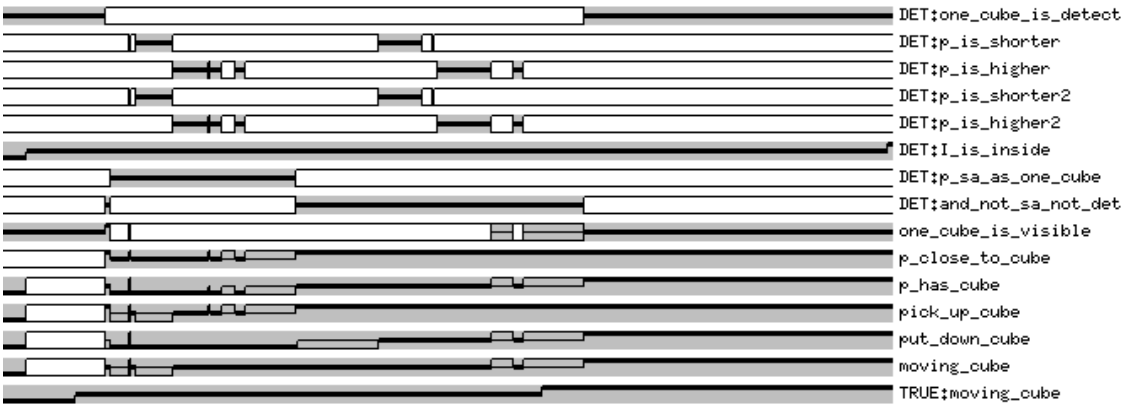


Figure 4.13 Example of the detection of *"moving cube"* using the method for recovery from errors.

verify the performance of the detection of the action, we designed an experiment where subjects entered the space and where asked to perform the action whenever they liked. They were instructed to remain inside the space for about 25 seconds, during which they should perform the action only once, and do whatever they wanted in the rest of the time. The experiment was recorded in video and the actual occurrence of the action *"moving cube"* was determined manually, providing ground truth to compare the results of the algorithm.

We run the experiment with 4 different subjects, for a total of 18 analyzed runs. Figure 4.13 display the PNF diagram of one of these runs (for subject #1). The top part of the figure contains the sensor data that are clearly noisy. The last line displays the ground truth, that is, the actual occurrence of the action as determined by manual inspection of the video. Just above it, we see the result of the action detection algorithm with recovery from errors. As it can be seen, the algorithm successfully determines whether the action is happening most of the time.

Table 4.1 Distribution of the four categories of answers according to the actual and the computed PNF-state.

		computed PNF-state						
		F	N	P	PF	PN	NF	PNF
actual state	F	right	wrong	wrong	right	wrong	not wrong	undetermined
	N	wrong	right	wrong	wrong	not wrong	not wrong	undetermined
	P	wrong	wrong	right	right	not wrong	wrong	undetermined

To evaluate numerically the performance of the algorithm, we classify the computed PNF-state at each instant of time according to four categories: **right**, if the algorithm provides an answer that is correct and specific; **wrong** if the computed PNF-state does not contain the true state of the action; **not wrong** if the PNF-state contains the true state of the action, but it is not specific; or **undetermined**, for the cases where the PNF-state is computed to be *PNF*. Table 4.1 displays the assignments of these categories for all possible combinations of actual and computed PNF-states. Notice that we consider that the algorithm is right when the actual state is *P* or *F* and the

Table 4.2 Results for the detection of different subjects performing the *"moving cube"* action.

	subject #1	subject #2	subject #3	subject #4	average
right	81.2%	84.4%	71.8%	68.1%	76.4%
not wrong	7.5%	6.8%	4.7%	14.7%	8.4%
wrong	9.4%	6.3%	8.1%	8.8%	8.1%
undetermined	1.9%	2.5%	15.5%	8.4%	7.0%
length of action	29.4%	33.0%	19.5%	34.5%	29.1%

computed PNF-state is *PF*, so it is possible to compare the performance of our method with traditional algorithms that do not differentiate between past and future.

Table 4.20 displays the results of detecting the action *"moving cube"* using PNF propagation augmented by the recovery from error method described in section 4.7. The table contains the results for each of the four subjects and the average detection rates. The last line displays the length of action compared to the total time where the subject was inside the space, in average 30% of the total time.

The method, in average, provided the correct answer 76% of the time and the wrong answer only 8% of the time. Also, if we also consider the time that the algorithm was not wrong, we obtain 85% of time where the system could provide an answer that was not wrong. We consider this results quite good given the amount of noise coming from the sensors and that we have not employed any kind of "smoothing" techniques on top of the results. We observed that in the situations where the sensor routines performed better the overall result of the algorithm was also greatly improved. Based on these observations we believe it is possible to increase the detection rates by at least 10%, to the levels achieved in the case of the best subject (#2).

4.9 Future Directions

The PNF-network model for action recognition proposed in this chapter needs further development. The major advantage is its ability to express complex temporal patterns, including mutual exclusion of actions and states. This is possible only because we have developed the PNF framework where approximate temporal constraint propagation is fast in a strong temporal algebra.

There are two major shortcomings in the model. First, PNF networks and algorithms can not handle confidence measures computed by perceptual routines. This introduces an undesirable degree of brittleness that does not match well with the reality of computer vision low-level routines. Second, the current methodology requires completely manual definition of the actions and their temporal structures. Some ideas to overcome these difficulties are discussed below.

4.9.1 Including Confidence Measures

To introduce confidence measures into the PNF propagation structure the first challenge is to determine what the subject of the confidence measure should be. In one scenario, each node would have different confidence values for each of the three basic states *past*, *now*, and *future*. Although this seems straightforward for the detector nodes, it is not clear which mechanisms

could be applied to propagate those values to non-detector nodes and how to compute this in a reasonable amount of time.

Instead, we have been considering a different approach where, given confidence measures for the detectors, we first associate a confidence measure to a component domain, and then to an individual temporal thread of component domains. Suppose that at instant of time *t*, a detector I_j has a confidence level $c_j^t(v)$ that its value is *v*. Let us consider a particular collection of values $\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_l$, and its associated component domain $\bar{V}^t$. Then we can compute the confidence level of $\bar{V}^t$ as

$$c^t\left(\bar{V}^t\right) = \prod_{j=1}^{l} c_j^t\left(\bar{v}_j\right)$$

In this model, since the confidence is associated to the detector nodes' confidences and not with the other nodes, computing the PNF restriction or the arc-consistency on a component domain does not alter its confidence, $c^t(W) = c^t\left(R(W)\right) = c^t\left(AC(W)\right)$, except if an inconsistency is detected. If $R(W) = \varnothing$, then this is a logically impossible situation, and therefore the confidence on the component domain must be 0,

$$c(\varnothing) = 0$$

Similarly, time expansion does not alter the confidence in a component domain, $c\left(\mathrm{T}(W)\right) = c(W)$.

Given this model, let us consider the set of error-tolerant component-domains Ω^t,

$$\Omega^t = \left\{ \; R\left(\mathrm{T}(\omega) \cap V_j^t\right) \neq \varnothing \;\middle|\; \text{for all } j = 0,1,2,\ldots,l \text{ and } \omega \in \Omega^{t-1} \right\}$$

Notice that an element ω of Ω^{t-1} represents a particular thread of component domains and detector values from the instant 0 to the current time *t-1*. If we assume that the occurrence of the sensor values at time *t-1* is independent of their occurrence at time *t*, we can compute the confidence of each element of Ω^t by taking the product of the confidence of the time expanded component thread, $c^{t-1}\left(\mathrm{T}(\omega)\right) = c^{t-1}(\omega)$, and the confidence of a particular set of sensor values, $c^t\left(\bar{V}^t\right)$

$$c^t\left(R\left(\mathrm{T}(\omega) \cap \bar{V}^t\right)\right) = c^t\left(\mathrm{T}(\omega) \cap \bar{V}^t\right) = c^{t-1}(\omega)\, c^t\left(\bar{V}^t\right)$$

Of course such a formula is possible only if independence between sensor values in different instants of time is assumed. However, there is one more technical difficulty. Since Ω^t is defined as the union of component domains, we have to determine what the confidence value is in the case of a component domain that is obtained by the combination of different threads and

sensor values. For instance, suppose two component domains ω_1 and ω_2 and two sets of sensor values $\overline{V}^t$ and $\overline{\overline{V}}^t$ such as

$$\omega' = R\left(\mathrm{T}(\omega_1) \cap \overline{V}^t\right) = R\left(T(\omega_2) \cap \overline{\overline{V}}^t\right)$$

A possible approach in such cases is to take as the confidence of the component domain ω' to be the maximum of the confidence of the two "threads" that led to it; that is,

$$c^t(\omega') = \max\left\{c^t\left(R\left(\mathrm{T}(\omega_1) \cap \overline{V}^t\right)\right), c^t\left(R\left(T(\omega_2) \cap \overline{\overline{V}}^t\right)\right)\right\}$$

Given this formulation, we can determine the most likely current state simply by considering the component domain with highest confidence in Ω^t,

$$W^t = \arg\max_{\omega \in \Omega^t} c^t(\omega)$$

As before, the set of error-tolerant component-domains can expand exponentially. A simple heuristics to prevent this is, at each cycle, to keep in Ω^t just the component-domains with highest confidence. Finally, it is necessary to normalize the confidence measures at each instant of time to avoid numeric underflow in the values of the confidence. We plan to test this formulation in real action recognition situations in the near future.

4.9.2 Learning Temporal Structures

In our methodology, the temporal structure of each action has to be manually defined by the designer of the application. First, notice that since the nodes in our representation are symbolic, it is possible to simplify the task by collecting basic representations into action dictionaries and, as discussed in chapter 2, pre-process the representations with simple reasoning algorithms in order to infer low-level nodes and states. Also, as explained before, we can separate the basic elements of the action from the specific detectors of the task at hand, increasing the reusability of previously defined actions.

However, it is important to consider the possibility of automatically learning the temporal structure of the actions. Particularly, we want to briefly discuss the possibility of learning the temporal constraints of a given network by observing multiple occurrences of the action. A simple mechanism would be to observe which pairs of states between two actions occur in the examples and to construct PNF temporal constraints (not Allen-type constraints) allowing only the occurrence of the observed pairs of states. For instance, if the only observed states between two nodes *A* and *B* are (*past*, *now*), (*now*, *now*), and (*now*, *future*), then the relation between then can be defined by $\langle PN, NF, F \rangle$. Notice that we have to include the relations (*past*, *past*) and (*future*, *future*) since it is always possible to have both nodes in the *past* or in the *future*.

The computed relation — or a close approximation — can be projected back using the function Γ^{-1} defined in chapter 3 into a set of Allen's primitive relations. Given the learned relations for

the whole IA-network, we can apply Allen's path-consistency algorithm, remove unnecessary relations, and project back into a PNF-network.

The problem with this approach is that it requires a set of observations that span all the possible occurrences of each pair which, especially in the case of parallel actions, can become quite a large set. This is, in fact, a problem that is faced by any learning algorithm for actions. For instance, in [118] Olivier et al. had to resort to synthesized data in order to train their HMMs to recognize simple actions like *"following"* and *"meeting"*. Notwithstanding, this is a direction that can potentially simplify the construction of the IA-networks.

4.10 Summary

In this chapter we showed how IA-networks can be used to represent the temporal structure of actions. In particular, IA-networks allow the definition of simultaneous sub-actions and states without the overhead that is typical of finite-state machines such as HMMs. Another feature of IA-networks is the symbolic specification of the temporal structure that allows the programmer/designer to refine the definition of an action and to easily visualize the different assumptions used to detect the action.

We demonstrated our ideas by showing examples of real-time detection of actions represented by IA-networks. During the detection of those actions, we also performed recognition of sub-actions. In particular, we highlighted the role of different constraints in the process and how causal links can be inserted to strengthen the recognition power.

The original PNF propagation formulation assumes perfect sensing. In this chapter we developed a set of heuristics to recover from errors that are based on tracking possible states through time. The results showed that being more resilient to errors slightly degrades the recognition but produces a recognition system able to handle many failure situations. We also tested the action detection method with real data in a concrete problem of recognizing an action and obtained good detection rates. Finally, in the future direction section we drafted a method to incorporate confidence measures into the framework based on the assignment of confidence values not to nodes but to entire component domains. In the next chapter we return to some of the issues discussed in this chapter when dealing with a different kind of application for PNF-networks: the control of interactive spaces.

5. Interval Scripts

In the previous chapter we described how the temporal structure of human actions can be represented and recognized using PNF-networks. The research described in this chapter extends the ideas previously developed by considering a different, but similar, problem: the scripting of computer characters and interaction. As before, we propose the use of temporal constraints to represent the temporal structure of the interaction and use the PNF framework to recognize and, in this case, control the interactive environment.

The proposed paradigm — called *interval scripts* — is based on the encapsulation of actions and states in nodes of an IA-network constrained by the temporal primitives of Allen's algebra [3]. The objective is to provide the designer of an interactive system or complex computer graphics scene with a tool that combines expressiveness and simplicity. Among other desired features, we included in interval scripts: facilities for the easy abstraction of actions, the ability to impose restrictions on what can happen (negative scripting), mechanisms to infer indirectly the occurrence of users' actions (in a similar manner to the previous chapter), and the capacity of recover from errors and unexpected situations.

To accomplish this, we added constraint-based scripting mechanisms to the traditional procedural methods of designing interaction. Basically, an interval script contains descriptions of how to activate and stop the actions of the characters. The activation of the actions is determined by verifying the current state of the system, comparing it with the desired state according to the constraints, and issuing commands to start and stop action as needed. Another innovation in interval scripts is the de-coupling of the actual state from the desired state which allows for the introduction of methods for recovery from errors and increases the robustness of the system.

Similar to the action recognition problem described in chapter 4, a key issue when scripting interaction is the management of complex temporal patterns. The interval script paradigm builds upon the work presented in the previous chapters first by employing temporal constraints to represent the structure of the interaction as we did for human action; and second by using the same PNF-propagation method (chapter 3) to allow fast constraint propagation. However, unlike the action recognition case, we have developed the paradigm into a fully functional prototype where the ideas could be thoroughly tested.

The idea of interval scripts has been implemented in an *interval script language* that is described in this chapter and fully specified in appendix D. Although the concept of interval scripts could be variously realized (e.g. as a graphical programming language), in this chapter we present all the basic concepts of our paradigm using the interval scripts language.

In the human-machine interaction community there has been much work on languages to describe interaction in multimedia interfaces and computer graphics environments. However, few attempts have been made to create methods to handle specific difficulties of scripting interactive characters in immersive, interactive worlds. Interval scripts have proved to be essential to manage the complexity of the large-scale interactive, story-driven spaces that we have been developing. Three of those projects will be examined in detail in chapter 7. It is our opinion that it would take far longer to implement these systems in any other language or system. In fact, we believe that the fact that it was possible to build these complex systems at all is good evidence of the usefulness and appropriateness of the interval scripts paradigm.

Although our research has concentrated on immersive environments and, in particular, in interactive theatrical performances, we argue that the design of scenes with semi-autonomous characters in animated movies face similar problems. Current industrial techniques for animation rely on the painstaking definition of the changes between two consecutive frames. Along with issues of movement and body control and facial animation, research on how to tell a computerized character what to do is becoming more common. Most of the research is, however, concentrated on how to describe coordinated body activities like walking, running, or Newtonian physical interaction.

When computer graphics is used to generate crowd scenes in animated features, it is often the case that simply applying flock-behavior like programming elements do not achieve the necessary result. For instance, in a scene with a crowd in a marketplace, an animator might want autonomous characters to wander around the place looking for merchandise and bargaining with the shoppers. It is conceivable with today's technology to create such a scene using behavior-based characters. However, in a real animated movie, it might necessary to impose story-related constraints to achieve the desired level of drama or comedy. For example, it is desirable that no big action among the crowd characters happens during a particular intense moment of the main characters; or perhaps that suddenly, the whole crowd pays attention to some event. As we will show, the interval script paradigm is suited to control high-level story-related issues similar to those.

The initial formulation of the idea of interval scripts was developed by Pinhanez, Mase and Bobick [137]. The methodology was greatly refined when the interval script language was developed for the production of the computer theater play *"It / I"* (see chapter 7 and [136] for more details). For the installation *"It"* (see chapter 7) we re-designed the run-time engine and considerably improved the interval script language. The examples discussed in this chapter refer to this version of the language.

This chapter starts by reviewing the current scripting languages and identifying their shortcomings. We then introduce the interval script language through some simple examples. The core of the chapter describes the run-time engine architecture based on the PNF

propagation method described in chapter 3. We then explore more complex constructions allowed by the interval script language. In chapter 7 we describe our experiences on using the paradigm to build interactive spaces.

5.1 Problems with current Scripting Techniques

The idea of interval scripts was spawned by our dissatisfaction with the tools for the integration of characters, stories, and I/O devices in interactive environments. As described by Cohen, systems to control interaction tend easily to become *"big messy C program(s)"* ([39], fig. 2). Also, from our experience with *"The KidsRoom"* [18], it became clear that one of the major hurdles to the development of interesting and engaging interactive environments is that **the complexity of the design of the control system grows faster than the complexity of the interaction**.

5.1.1 Finite-State Machines and Event Loops

The most common technique used to script and control interactive applications is to describe the interaction through finite-state machines. This is the case of the most popular language for the developing of multimedia software, *Macromedia Director's Lingo* [1]. In *Lingo* the interaction is described through the handling of events whose context is associated with specific parts of the animation. There are no provisions to remember and reason about the history of the interaction and the management of story lines.

Videogames are traditionally implemented through similar event-loop techniques. To represent history, the only resort is to use state descriptors whose maintenance tends to become a burden as the complexity increases. Also, this model lacks appropriate ways to represent the duration and complexity of human action: hidden in the structure is an assumption that actions are pinpoint-like events in time (coming from the typical point-and-click interfaces for which those languages are designed) or a simple sequence of basic commands.

In the *"The KidsRoom"* [18] an interactive, immersive environment for children built at the MIT Media Laboratory, the interaction was controlled by a system composed of a finite-state machines where each node has a timer and associated events. A problem with this model is that it forces the designer to break the flow of the narrative into manageable states with clear boundaries. In particular, actions and triggered events that can happen in multiple scenarios normally have to be re-scripted for each node of the state machine, making incremental development quite difficult.

5.1.2 Constraint-Based Scripting Languages

The difficulties involved in the finite-state based model have sparkled a debate in the multimedia research community concerning the applicability of constraint-based programming (starting with the works of Buchanan and Zelllweger [34] and Hamakawa and Rekimoto [59]) versus procedural descriptions, including state machines (for example, [175]). In general, it is believed that constraint-based languages are harder to learn but more robust and expressive.

Bailey et al. [11] defined a constraint-based toolkit, *Nsync*, for constraint-based programming of multimedia interfaces that uses a run-time scheduler based on a very simple temporal algebra. The simplicity of the temporal model, in particular due to its inability to represent non-acyclic structures, is also the major shortcoming of *Madeus* [74], *CHIMP* [154], *ISIS* [2], and *TIEMPO* [184].

André and Rist [9] have built a system, called *PPP*, that designs multimedia presentations based on descriptions of the pre-conditions of the media actions, their consequences and on the definition of temporal constraints using a strong; hybrid temporal algebra that combines intervals and points as its primitives (based on Kautz and Ladkin's work [78]). The system uses the descriptions to plan a multimedia presentation that achieves a given goal. The multimedia presentation is represented as a constraint network. During run-time, the scheduler has to build and choose among all the possible instances of the plans. However, since Kautz and Ladkin's propagation techniques are exponential, and the number of possible plans can be exponentially large, the applicability of André and Rist's *PPP* [9] is restricted to simple situations, and especially, to cases with limited interaction (otherwise the number of possible plans increases too fast).

In computer graphics, Borning [25] has been the strongest proponent of constraint-based languages for animation. Other examples are *TBAG* [49] and Kakizaki's work on deriving animation from text [76].

5.1.3 Scripting of Computer Graphics Characters

The scripting of computer graphics characters and, in particular, of humanoids has attracted considerable attention in the computer graphics field. A group of researchers has worked with languages for scripting movements and reactions of characters, like Perlin [124], Kalita [77], and Thalman [170]. In particular, Perlin [125] has developed a language, *Improv*, that nicely integrates low-level control of geometry with middle-level primitives such as *"move"*, *"eat"*, etc. The major shortcoming of *Improv* is the lack of simple methods to synchronize and to temporally relate parallel actions. On the other hand, Cassel et al. [36] examines directly the contents of a dialogue between two characters and automatically generates gestures, facial expressions, and voice intonations; however, it is hard to imagine a way in their system to impose on the characters dramatic constraints that could alter their behavior.

Bates et al. [14] created an environment composed of several modules that encompass emotions, goals *(Hap)*, sensing, language analysis, and generation. Character actions are represented as AND/OR plans to achieve goals. To represent the development of the interaction among the characters, *Hap* employs an adaptation of AND/OR trees where AND nodes contain conjunctions of goals and OR nodes represent disjunctions of plans [87]. During run-time, the system grows an *active plan tree* considering the goals that are achieved and excluded by the current state of the world, and selects appropriate sub-goals and plans to be executed. Although *Hap* provides easy forecast of the future effect of actions, it is hard to express complex temporal structures in the formalism. By the own nature of AND/OR trees, *Hap* imposes many restrictions in the ways parallel actions can be synchronized.

Another line of research stems directly from Rodney Brooks' works with autonomous robots [33]. The objective is to create characters with their own behaviors and desires. Significant examples are Blumberg's behavior control system [17], Tosa's model for emotional control [172], and Terzopoulos work on the interaction of sensory and behavior mechanisms [169]. Sims [157] went beyond the creation of behaviors and devised methods to evolve them in conjunction with the creature's physiology. The problem with purely behavior-based characters is that they are unable to follow stories. In chapter 6 we will argue that there is a significant difference between computer creatures and computer actors, the former being unable to accept high-level commands that are needed for the precise timing and developing of a stories. Integration of high-level thought into behavior-based control still defies the research in the area.

Finally, Strassman's *Divadlo* [167] uses natural language-like statements to interactively create animation. The system uses a shallow rule-base reasoning system that compares the users' statements to a world database. However, as a foundation for a run-time system, the issue of robustness poses a main problem.

5.1.4 Scripting Interactive Environments

Starting with the pioneer work of Myron Krueger [79] the interest of building interactive environments, especially for entertainment, has grown in the recent years (see [15] for a good review). In the case of virtual reality (VR), it seems that the user-centered and exploratory nature of most interfaces facilitates the scripting of the interface by using state machines and event loops. There are, however, few references of scripting systems for VR (for example [155]). A recent example is *Alice* [120] used in Disney's *"Alladin"* ride (described in [121]) a language that allows rapid prototyping but has very few possibilities in terms of temporal structures.

In many interactive environments the control of the interaction is left to the characters themselves (seen as creatures living in a world). This is the case of *ALIVE* [94], where the mood of the CG-dog *Silus* commands the interaction; and also in *Swamped* [71] where a user-controlled chicken plays cat-and-mouse with a raccoon. In both cases, however, there is no easy way to incorporate dramatic structure into the control system as discussed above. In most of these cases, as well as in Galyean's work [52], it is necessary to "hack" the behavior structure in order to make a story flow. For example, in a scene of where a dog is supposed to follow a human character, the behavior is achieved by making the leg of the human being associated with food [16].

In [39] Cohen proposes a distributed control structure, the *Scatterbrain*, based on Marvin Minsky's concepts of *society of mind* [106] and Rod Brooks' *subsumption architecture* [33]. In this paradigm, used in the control of a sensor-loaded room, the dispersion of knowledge and behavior is even greater. Although an appealing model, the use of multiple agents without centralized control makes story control and, to some extent, authoring, extremely difficult.

5.2 A Proposal: Interval Scripts

Using *interval scripts* to describe interaction was first proposed by Pinhanez, Mase, and Bobick in [137]. The basic goal was to overcome the limitations of traditional scripting languages as described above. The work drew from the technique for representation of temporal structure of human actions (described in chapter 4) which was in its initial stages at that time.

In interval scripts, all actions and states of both the users and the computer characters are associated with the temporal interval where they occur. However, the actual beginning and ending of the intervals are not part of the script. Instead, the script contains a description of the temporal constraints that the intervals obey during run-time. For example, suppose we want to describe the situation where a CG character appears on the screen whenever the user makes a gesture. To do this with interval scripts, we associate the user gesture to an interval *A* and the appearance of a CG character to another interval *B*. We then put a temporal constraint stating that the end of the gesture interval *A* should be immediately followed by the entrance of the character *B*. During run-time, the interval script engine monitors the occurrence of the time interval corresponding to the gesture. When it finishes, the engine solicits the entrance of the CG character in an attempt to satisfy the temporal constraint.

In this chapter we will use sometimes the term *interval* to refer to nodes of the IA-network corresponding to an interval script. Although, as defined in chapter 3, the nodes of an IA-network are variables on the space of temporal intervals, we found that it is conceptually simpler to refer to the representation of the possible temporal occurrence of an action (that is, a node in an IA-network) as the *interval of the action*.

Developing an interface or interactive system with interval scripts is a classical case of programming by constraints as discussed above. Notice that programming by constraints always requires a run-time engine that examines the current information and assigns new values for the different variables of a problem to satisfy the constraints.

To model the time relationships between two actions we employ again the interval algebra proposed by Allen [3] and described in chapter 3. For instance, in the above example, the relation *A* meet *B* describes the temporal constraint between the gesture interval *A* and the interval associated with the entrance of the CG character *B*. If the entrance of the character could start before the end of the gesture, the relation between the two intervals would be described by the disjunction overlap OR meet. Of course, in any real occurrence of the intervals, only one of the relationships actually happens.

We see several reasons to use Allen's algebra to describe relationships between actions in an interactive script. First, no explicit mention of the interval duration or specification of relations between the intervals' starting and endings are required. Second, the existence of a time constraint propagation algorithm allows the designer to declare only the relevant relations, leading to a cleaner script. *Allen's path consistency algorithm* (described in chapter 3) is able to process the definitions and to generate a constrained version that contains only the scripts that possibly satisfy those relations and are consistent in time. Third, the notion of disjunction of interval relationships can be used to declare multiple paths and interactions in a story. Fourth, it

is possible to determine whether an action is or should be happening by properly propagating occurrence information from one node of the IA-network to the others in linear time (as described later in chapter 3). This property is the basis of the interval script run-time engine which takes temporal relationships between intervals as a description of the interaction to occur and determines which parts of the script are occurring, which are past, and which are going to happen in the future by considering the input from sensing routines.

Finally Allen's algebra allows the expression of mutually exclusive intervals. For instance, to state that a CG character does not perform actions *C* and *D* at the same time we simply constrain the relation between the intervals *C* and *D* to be before OR i-before. That is, in every occurrence of the intervals, either *C* comes before *D* or after *D*, but never at the same time. As discussed in detail in chapter 3, the ability of expressing mutually exclusive intervals defines different classes of temporal algebras [102]. In general, algebras without that property allow fast constraint satisfaction but are not expressive [177]. However, Allen's algebra is both expressive and can be used in real time because we have developed a fast method to compute approximations of the values that satisfy the constraints — PNF propagation.

André and Rits [9] have also used a strong temporal algebra to script multimedia presentations. Their work employs a more expressive temporal algebra that combines Allen's [3] and Villain's [178] proposals as suggested by Kautz and Ladkin [78]. However, constraint propagation in this temporal algebra is basically $O\left(n^2\left(e+n^3\right)\right)$, where *n* is the number of nodes and *e* is the time required to compute the minimal network of the IA-network; typically, *e* is exponential in the number of nodes, $O(e)=O\left(2^n\right)$. Moreover, their system requires the enumeration of all possible temporal orders of actions implied by the constrained network, which can grow exponentially, especially in the case of multiple and frequent user interaction.

We can see an interval script as the declaration of a graph structure where each node is associated with actions, events, and states of characters and users in an environment. The arcs between nodes represent the constraints imposed on the structure of the interaction. Basically, an interval script describes a space of stories and interactions. Notice that, unlike in other formalisms like [14], interval scripts represent and control plans of action and interaction but can not create them from a set of goals like in traditional planning [116]. In the next section we introduce the basic structures used in interval scripts. Following that, we detail the basic inference mechanism of the run-time engine.

5.3 Basic structures of Interval Scripts

In this section we describe the basic capabilities of a scripting language based on the paradigm of interval scripts. There are multiple ways to implement the concepts described in this chapter. We chose to develop a compiler that takes a text file containing the descriptions of the intervals and the temporal constraints between them and produces a C++ file. The C++ file can be compiled and, through especially-defined functions, the script can be executed at run-time.

It is arguable whether a text file is an adequate format for scripting interaction and stories in comparison with a graphical programming language. We will return to this discussion later. For

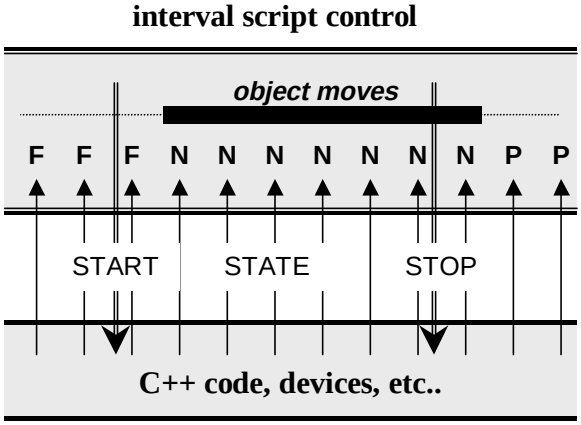


Figure 5.1 The structure of an interval.

now, we will describe the fundamental structures of interval scripts assuming the syntax and format of our language. The grammar of the language is described in appendix D. Notice that in most of the cases the constructions we examine are completely implementation-independent.

5.3.1 Start, Stop, and State Functions

Each action or state in an interval script is described by three functions:

- **START:** a function that determines what must be done to start the action; however, after its execution, it is not guaranteed that the action has actually started.
- **STOP:** a function defining what must be done to stop the action; similar to START, the interval may continue happening after the STOP function is executed.
- **STATE:** a function that returns the actual current state of the interval as a combination of three values, *past*, *now*, or *future*, corresponding respectively to the situation where the interval has already happened, is happening in that moment of the time, or it has not happened yet.

STATE functions are oblivious to START and STOP functions and are designed to provide the best assessment of the actual situation. Also, if they are not declared, the state of an interval is automatically set to *PNF*. They are used both to define intervals attached to sensors and to provide the actual state of execution of an action.

As we can see, in interval scripts we de-coupled the goal of starting or stopping an interval (represented by the START and STOP functions) from the actual state of the interval. That is, an interval script describes how an interaction or story should happen and does not assume that response is perfect. That is simply because it is impossible to predict how characters or devices actually react in run-time due to delays, device constraints, or unexpected interactions. This follows the notion of *grounding* as proposed by Rodney Brooks for autonomous robots [33]. According to this principle, a system that interacts with other complex systems should as much

as possible sense directly and not predict or model the common environment to avoid losing contact with the reality.

We explicitly distinguish between wish and occurrence exactly because interactive environments are as hard to model as the real world. Even in the case of computer graphics scenes where all the information is potentially available, it is still possible that a situation satisfying all the different agents is unattainable.

Therefore, we adopted a design in interval scripts where the request that an action happens is independent of the actual occurrence of the action. During run-time, the interval scripts engine examines the current state of the intervals (through the results of STATE functions) and tries to achieve a set of states compatible with the temporal constraints by executing appropriately START and STOP functions.

Figure 5.1 shows a typical execution of an action in interval scripts. This action corresponds to a movement of a CG character. In this diagram time is running from left to right, and the START function is executed some time before the STATE functions actually detects the occurrence (*N* for now state) of the action. Similarly, it takes time for the effects of the STOP call to be sensed. Of course the response time is different for different devices and systems. In the case of interactive environments, it is common that some physical devices have significant delays; while in computer graphics scenes, it can take time for a character to start an action because pre-conditions must be have to be achieved first. For instance, a command for moving might be delayed because the character first must stand up.

Figure 5.1 depicts the best case where the action actually happens after the START call. It is easy to see that physical or spatial constraints can prevent an object from moving. In the same way, a character might start to move because an object bumped into him. In both cases, we would expect that the STATE function report the actual state of the character.

5.3.2 Encapsulating Code in Intervals

It is important that a scripting language communicates with low-level programming languages that provide basic functions and device access. In our current implementation of interval scripts, we allow the inclusion of C++ code directly into the script.

Let us examine an example from the script of one of our experiments, the art installation called *"It"* described in chapter 6. In that installation a camera-like object appears on the screen and interacts with the user ("camera", in this example, is not the computer graphics virtual camera, but a CG object that looks like a photographic camera).

```
"camera moves" =
  {
     START:   execute [> camera.Move(posA,posB); <];
     STOP:    execute [> camera.Stop(); <];
     STATE:   set-state pnf-expression
                [> (camera.isMoving() ? _N_ : P_F) <];
  }.
```

Figure 5.2 Interval describing the movement of the "camera" character.

Figure 5.2 shows the definition of the interval *"camera moves"* in our language for interval scripts. The definition is comprised between curly brackets containing the definition of the basic functions of the interval. To include C++ code we use the command execute followed by the symbols "[>" and "<]". For instance, when the START function is called, it executes the C++ code between those symbols, that is, a C++ method called "Move" for the object "camera" with parameters "posA" and "posB". These variables are defined in separated C++ files that are linked together with the code generated by the interval script compiler.

In our model, a START function is not called if the current PNF-state of the corresponding interval does not contain the *F* state. That is, an interval is started only if there is a possibility that interval is in the *future* state. Similarly, a STOP function is not called unless the PNF-state of the interval contains the *now* state. We also employ a mechanism by which if a STOP function is called and the PNF-state is exactly *F*, then the state of the interval is set to *P*, but the function is not called.

The definition of the STATE function is slightly different. In this case, the function is defined to set the state of the interval to be equal to the PNF-state returned by the C++-code inside the special symbols. In the case depicted in fig. 5.2, a method of the object "camera" determines if the computer graphics camera is moving or not. If true, the state of the interval is set to *now*, referred in the C++-code by the special constant "_N_"; otherwise, the state is set to be *past* OR *future*, symbolized by "P_F". We similarly define the constants "P__", "__F", "PN_", "_NF", and "PNF". The last constant stands, as before, for *past* OR *now* OR *future*, that is, there is no information available about the interval.

5.3.3 Putting Constraints between Intervals

Let us examine now how temporal constraints are defined. Continuing the example of fig. 5.2, let us consider that after the camera moves, it should zoom, that is, move forward towards the user. Also, we would like the camera movement to be accompanied by the sound of a pre-recorded file.

```
"camera moves" ={ . . . }.
"camera zooms" ={ . . . }.
"camera moving sound" = { . . . }.

better-if  "camera moves"  meet  "camera zooms".
better-if  "camera moving sound"  start OR equal  "camera moves".
```

Figure 5.3 Script with temporal constraints.

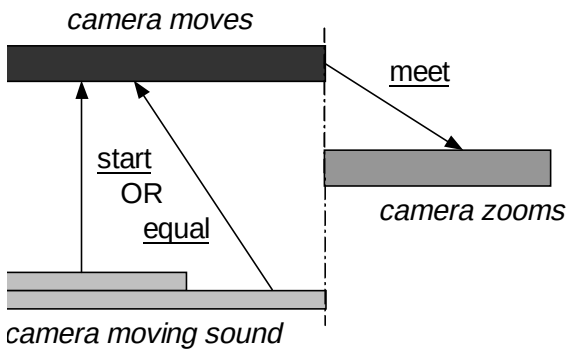


Figure 5.4 Diagram of temporal constraints in the script of fig. 5.3.

Figure 5.3 shows the script corresponding to the situation. It contains three intervals (whose definitions, using C++ inline code, are omitted for clarity). The interval declarations are followed by two statements establishing temporal constraints between intervals. In the first, it is declared the *"camera moves"* should meet with the interval *"camera zooms"*. Notice the syntax better-if which was chosen to imply that this is a constraint that will be tried to be enforced but that is not guaranteed to occur. Figure 5.4 shows the desired relation between the two intervals.

The last line of the script in fig. 5.3 establishes a constraint between the intervals *"camera moving sound"* and *"camera moves"*. They should always start together and *"camera moving sound"* can end before or at the same time as *"camera moves"*. Figure 5.4 renders the two possible occurrences of *"camera moving sound"* in order to respect the constraint start OR equal.

```
"camera moves" ={ . . . }.
"camera zooms" ={ . . . }.
"camera moving sound" = { . . . }.
better-if  "camera moves"  meet  "camera zooms".
better-if  "camera moving sound"  start OR equal  "camera moves".

"user is posing" =
   {  STATE:    set-state pnf-expression
                [> (user.isMoving() ? _N_ : P_F) <];
   }.

"ready to click" =
   {  STATE:    if  "camera zooms"  is past
                    AND "user is posing"  is now
                set-state now
                endif.
   }.

"camera clicks" = { . . . }.

better-if  "ready to click"  start OR equal OR i-start  "camera clicks".
```

Figure 5.5 Scripting based on previously defined intervals.

5.3.4 Defining on Previous Intervals

Although the ability to include references to external code is very important, a key feature that we want to introduce with interval scripts is the possibility of defining a new action or state solely based on other intervals. With this we can create hierarchies, abstract concepts, and develop complex, high-level scripts.

Continuing with our example, we want to define the interaction between the user and the camera-like object as follows. When the user makes a pose (as detected by a computer vision system) and the camera has finished moving and zooming, the camera can "click" and "take" a picture. Figure 5.5 shows the script corresponding to this interaction.

Initially the interval "*user is posing*" is defined as before, by a reference to C++ code that communicates with the vision system. Then we define the interval "*ready to click*" that has only a STATE function. The state of the interval is determined by checking the state of two previously defined intervals: if "*camera zooms*" is in the *past* state, and "*user is posing*" is happening, than the state is set to *now*, otherwise it is the default, that is, undetermined (*PNF*).

In our model we assume that the state of the intervals is evaluated following the order of their definition. In the case of nested intervals, the state of the contained intervals is determined before the state of the container. By doing that, we assure that the PNF-state of all intervals refer to the same instant of time.

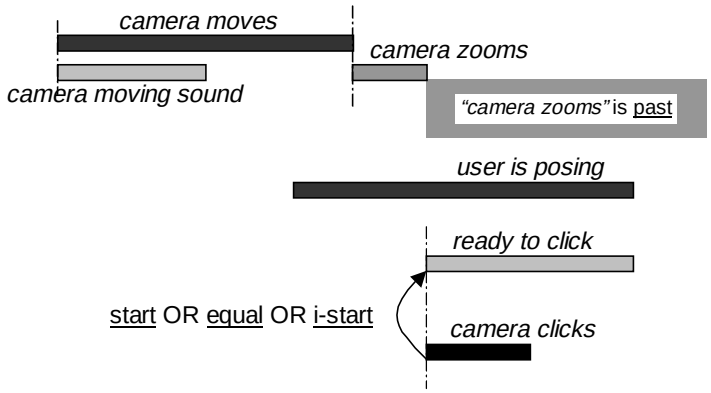


Figure 5.6 A possible occurrence of the script described in fig. 5.5.

To accomplish the clicking of the camera and the "taking of the picture", the interval *"camera clicks"* is defined, again through invocation of C++ code. Finally, we want that whenever *"ready to click"* starts, the camera takes the picture. This is established by the constraint start OR equal OR i-start that forces the two intervals to start together. That is, when the interval *"ready to click"* assumes the state *now* (by examining the state of its two defining intervals), the propagation of temporal constraints will request that *"camera clicks"* is also running. In response to that the run-time engine calls the START function *of "camera clicks"* as many cycles as needed until the state of the interval also becomes *now*. This process is described in detail in the next section.

A typical occurrence (the case where the actual relation is start) is depicted in the diagram of fig. 5.6. Notice that *"ready to click"* is now in the intersection between *"user is posing"* and the time after *"camera zooms"* is finished. Also, the figure shows one of the possible occurrences of *"camera clicks"*, when the interval starts *"ready to click"*, that is, starts together but finishes first.

5.3.5 Mutually Exclusive Actions

We want to improve the scene described above by not allowing the taking of pictures if the user is moving. This is the classical case of mutually exclusive intervals mentioned above. Moreover, in our vision system the detection of movement is independent of detection of posing. Using interval scripts the expression of such constraints is easily accomplished by adding the following lines to the script of fig. 5.5:

```
"user is moving" = { . . . }.
better-if  "camera clicks"  before OR i-before  "user is moving".
```

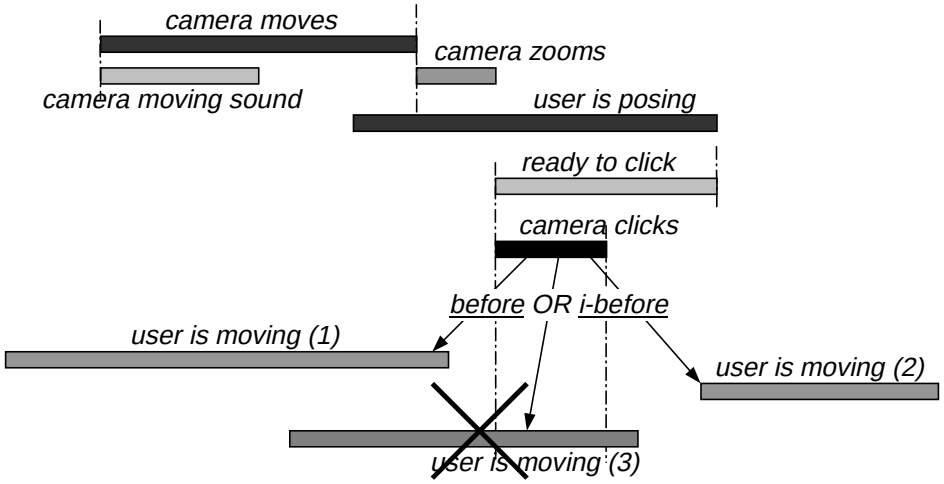


Figure 5.7 Possible occurrences of "user is moving".

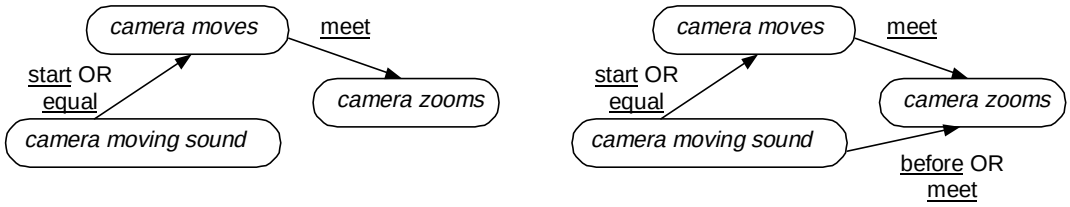
The interval *"user is moving"* communicates with the vision system and assumes the state *now* if the user is perceived as moving around. Then, a constraint is established determining that moving and clicking never happen at the same time.

Figure 5.7 shows three possible occurrences of the interval *"user is moving"*. The first two, before or after *"camera clicks"*, are compatible *with "camera clicks"*. The third occurrence overlaps with *"camera clicks"* and is not compatible. In this last case, since by definition *"user is moving"* is only a STATE function and can not be stopped, the run-time system would never call the START function of *"camera clicks"*, preventing the undesirable situation from happening. To understand better how this is accomplished, it is necessary to examine how the run-time engine actually works.

5.4 The Interval Script Engine

As we see from above, an interval script associates actions and states of an interactive environment to a collection of nodes in an IA-network with temporal constraints between them. In this section we explain how the state of the intervals is determined during run-time and the process that triggers the call of START and STOP functions. At each time step the goal is to select which of those functions to call in order to make the state of the world, as represented by the PNF-state intervals, consistent with the temporal constraints of the PNF-network associated to the script.

The basis of this process is the PNF propagation method described in chapter 3. We start by observing that, given an interval script, we have two different kinds of information: how to compute the state, and how to start, and stop each interval as represented in the basic functions; and the temporal constraints between them. The collection of temporal constraints constitutes an interval algebra network. Item (a) of fig. 5.8 displays the interval algebra network associated with the script of fig. 5.3.



a) Original network. b) After Allen's path consistency.

Figure 5.8 Interval algebra network associated with the script of fig. 5.3 before and after path consistency.

As in the case of action recognition, it is possible to remove most of the superfluous temporal relations by applying *Allen's path consistency* algorithm (described in chapter 3). As noted before, computing the minimal network that contains only satisfiable constraints is *NP-hard* (as shown in [177]). In all of our experiments we have always used only Allen's path consistency algorithm and have found no case where a stronger algorithm seemed to be needed. Item (b) of fig. 5.8 shows the result of the application of the path consistency algorithm on the IA-network corresponding to the script of fig. 5.3.

5.4.1 Using Prediction to Avoid Conflicts

We have defined a framework where wish and reality are de-coupled and, at the same time, temporal constraints are imposed between the different actions and sensing states of the IA-network corresponding to an interaction script. The problem is how to decide when to call the START or the STOP function of each interval, especially in the situation where the current state of the system is inconsistent.

Let us consider a very simple case of a network of two intervals, *A* and *B*, temporarily constrained to be equal. Figure 5.9 shows a situation where the PNF-state of interval *A* is *PF* and *B* is *N*, and therefore their values do not satisfy the constraint. In that situation the system should take two actions: try to start interval *A* (so it PNF-state becomes *N*) or to stop interval *B* (becoming *P*). Figure 5.9 shows the four options for the actions to be taken, considering that the no action can be done, one of the two, or both. In particular, notice that if the system decide both to start *A* and to stop *B*, a new inconsistent state is reached.

The goal of this example is to show that taking action involves looking for actions whose outcome lead to the global consistency of the network. If we look only locally in each node for a manner to make the network consistent, it might happen that both nodes *A* and *B* change their state but keep the network inconsistent (as it is the case when both *A* is started and *B* is stopped). The correct way to choose which intervals to start or to stop is to look for global solutions for the IA-network corresponding to the achievable PNF-states.

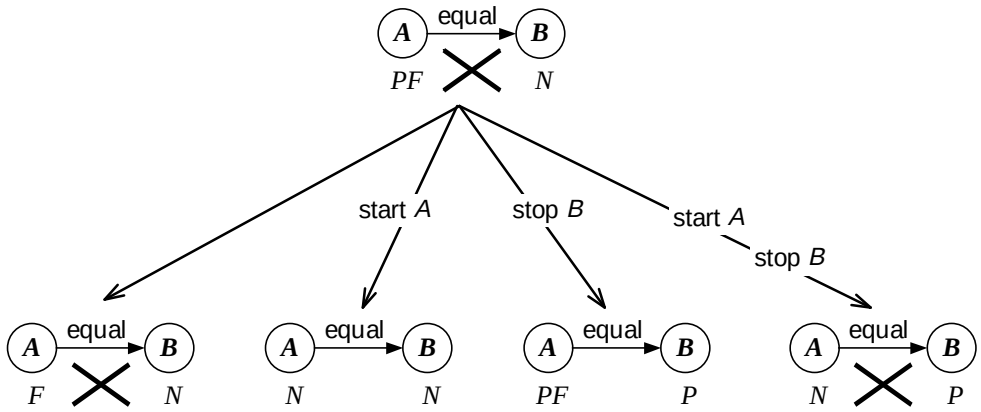


Figure 5.9 Example of an inconsistent state and the effect of possible actions.

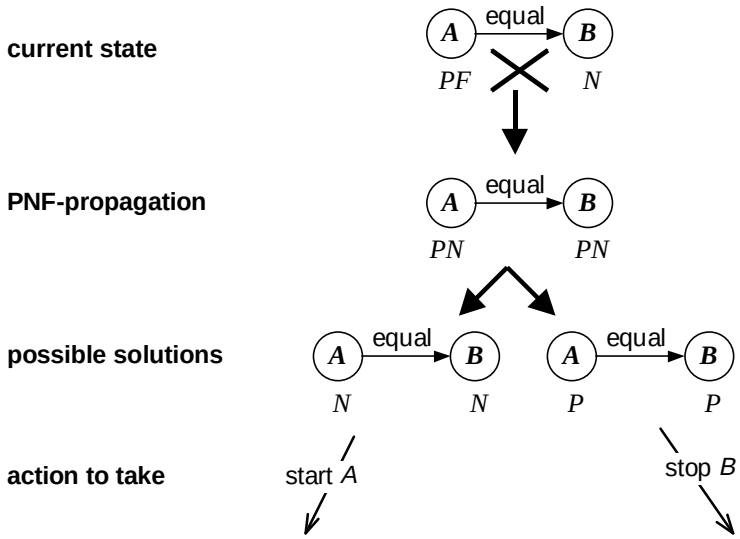


Figure 5.10 Finding a globally consistent state for the situation depicted in fig. 5.9

To quickly determine possible globally consistent solutions, we propose starting the process by PNF propagating the current state of the network, that is, time-expanding and computing the minimal domain (or an approximation). By doing so we obtain a compact representation of the space of consistent solutions for the next time step. Figure 5.10 displays this process in the example of fig. 5.9. As displayed in the figure, the result of the PNF propagation is that both intervals can be in the *PN* state in the next time step. In fact, the time expansion of PF gives *PNF*, while for interval B the result of the time expansion is *PN*. As defined in chapter 3, the

second step of PNF propagation involves taking the PNF restriction, which rules out *future* as a possible state for interval *A*.

Now it becomes easier to search for globally consistent solutions. In this case, the globally consistent solutions are *(N, N)* and *(P,P)*. In the first case, the appropriate action to be taken is to start *A*, while in the second is to stop *B*. The decision of which line of action to take is arbitrary.

5.4.2 Run-Time Engine Architecture

The method described above has the problem that it is necessary to search for solutions in the PNF-network, what can be exponential in the number of intervals. To avoid this, we have designed a run-time engine for interval scripts that has heuristics that decide which action to take without resorting to the full search.

There are three reasons for doing so. First, we have found in our experiments that those heuristics handled quite well the decision of which actions to take. Second, many times the inconsistency in the original global state is caused by faulty sensing and therefore the delay caused by performing a full search is not justifiable. In fact, in the latter case, it is often more useful if the system just proceeds as fast as it can to the next step, causing the minimal amount of change that can be irreversible. Third, as mentioned above, it may take time for intervals do actually start and stop; while waiting for this to happen, the run-time system might detect states with no solutions. For instance, if in the example above the decision is to start the interval *A*, some cycles may be necessary until the interval actually starts, and while that does not happen the constraint is violated (and, again, the search is useless). Notice that, by design, an interval script describes the situations that should happen but not the intermediate, unexpected states. The handling of these is left for the run-time engine.

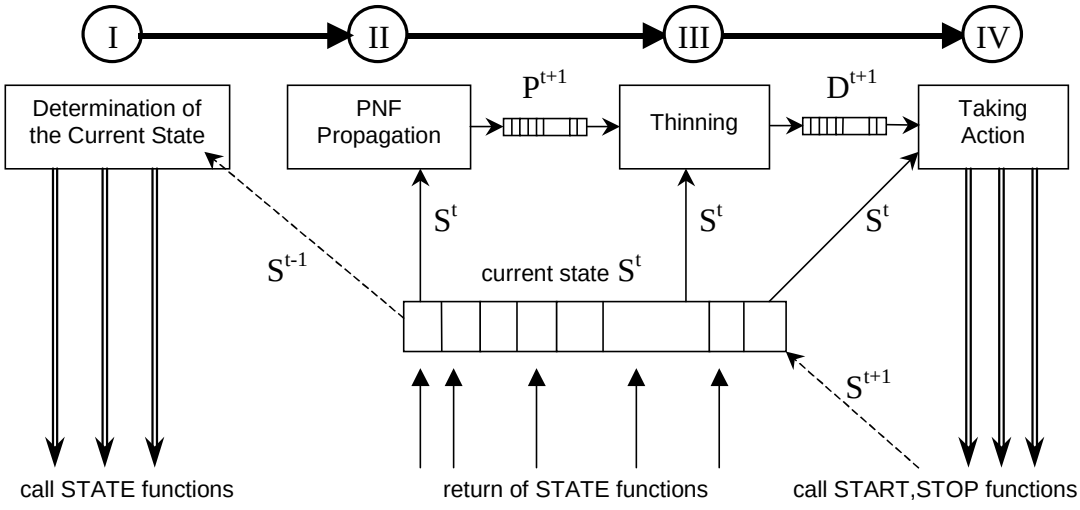


Figure 5.11 A cycle of the interval script engine.

Figure 5.11 shows a diagram of one cycle of the run-time engine. The main component is the current state S^t at time *t* that contains the current state of all intervals in the associated PNF-network. It is important to notice that, unlike PNF propagation, the current state is not updated by time expansion or PNF restriction at each cycle.

Each cycle *t* is composed of the following four stages:

I. Determination of the Current State: all the STATE functions are called and the engine waits until all the results are reported back and stored in the component domain S^t. For this computation, the STATE function is allowed to use the previous state of the intervals, available in the previous component domain S^{t-1}, and the current states of all the intervals that had already been computed. After this stage, the current state remains unchanged for the rest of the cycle.

II. PNF Propagation: in this stage the engine tries to determine what changes can be made to satisfy the constraints in the next cycle $t+1$, noted as P^{t+1}. Unlike in the original time expansion, we consider for expansion only those intervals that can be started or stopped, since no action can be taken to alter the state otherwise,

$$\overline{\mathrm{T}_m}\left(S_i^t\right) = \begin{cases} \mathrm{T}_m\left(S_i^t\right) & \text{if the interval } i \text{ has STARTor STOP functions} \\ S_i^t & \text{otherwise} \end{cases}$$

Using this definition, this stage then PNF-propagates the current state,

$$P^{t+1} = R\left(\overline{\mathrm{T}_m}\left(S^t\right)\right)$$

If the computation of $P^{t+1} = R\left(\overline{\mathrm{T}_m}\left(S^t\right)\right)$ detects a conflict, $P^{t+1} = \varnothing$, the engine tries to enlarge the space of possibilities by using simple PNF propagation, that is, expanding all states regardless of the existence of START or STOP functions,

$$P^{t+1} = R\left(\mathrm{T}_m\left(S^t\right)\right)$$

Although this normally handles most problems, there are situations where sensors report incorrectly and produce a state with no solutions. In those extreme cases, the engine simply time-expands the current state without computing the restriction,

$$P^{t+1} = \mathrm{T}_m\left(S^t\right)$$

III. Thinning: the result of stage II is normally too big and undetermined, although it contains all the globally consistent solutions. To have a more specific forecast of the next stage without doing search, we apply an heuristic where the current state of an interval should remain the same unless it contradicts the result of the PNF-propagation, but as long as the final result is still feasible. This is accomplished by taking a special

intersection operation between the forecast of the next state P^{t+1} and the current state S^t. For each node the special intersection is computed by

$$S_i^t \overline{\bigcap} P_i^{t+1} = \begin{cases} S_i^t \bigcap P_i^{t+1} \text{ if } \quad S_i^t \bigcap P_i^{t+1} \neq \varnothing \\ P_i^{t+1} \quad \text{otherwise} \end{cases}$$

The result is then passed through PNF restriction to assure that there are solutions and to remove impossible states,

$$D^{t+1} = R\left(S^t \overline{\bigcap} P^{t+1}\right)$$

If the computation of $D^{t+1} = R\left(S^t \overline{\bigcap} P^{t+1}\right)$ yields a state with no solutions we simply ignore the intersection

$$D^{t+1} = P^{t+1}$$

In practice this normally prevents any action to be taken since the states of P^{t+1} tend to be not as simple as required to call start and stop functions.

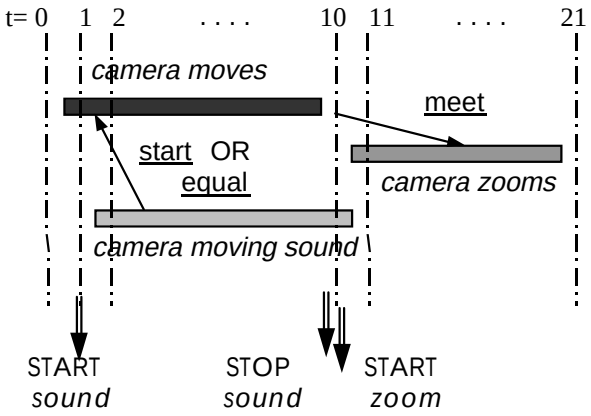
IV. Taking Action: The result of thinning, D^{t+1}, is compared to the current state S^t, and START and STOP functions are called if a need to change the state of an interval is detected. This is done according to the following table:

$x \subseteq S_i^t$	D_i^{t+1}	action
F	*N, PN*	START
N	*P*	STOP
F	*P*	STOP

For example, if the current state of interval *i* can be future, $F \subseteq S_i^t$, and the desired state is *now*, $N = D_i^{t+1}$, then the interval should start and the START function of the interval is called.

In the case that we want to perform the search for globally consistent solutions, stage III (thinning) would be substituted by the search of all solutions and the arbitrary choice for one among them (or simply, by considering the first solution found). As detailed in chapter 3, we can use arc-consistency to speed up considerably such search. In practice, the run-time engine with the heuristics described above has been able to run all our applications without halting, deadlocking, or flip-flopping.

Let us examine the behavior of the run-time engine in the example of fig. 5.9 where intervals *A* and *B* are constrained by an equal relation. Let us assume that $t = 0$ and the current state determined at stage I is $S^0 = (PF, N)$. The result of PNF propagation is $P^1 = (PN, PN)$. The first attempt of thinning the result of PNF propagation yields $S^0 \overline{\bigcap} P^1 = (P, N)$, which is not



interval	t=0			t=1			t=2			t=10			t=11			t=21		
	S^0	P^1	D^1	S^1	P^2	D^2	S^2	P^3	D^3	S^{10}	P^{11}	D^{11}	S^{11}	P^{12}	D^{12}	S^{21}	P^{22}	D^{22}
camera moves	F	NF	F	N	PN	N	N	PN	N	P	P	P	P	P	P	P	P	P
camera moving sound	F	NF	F	F	N	N	N	PN	N	N	P	P	P	P	P	P	P	P
camera zooms	F	F	F	F	F	F	F	NF	F	F	N	N	N	PN	N	P	P	P

Figure 5.12 Example of a run of the interval script of fig. 5.3.

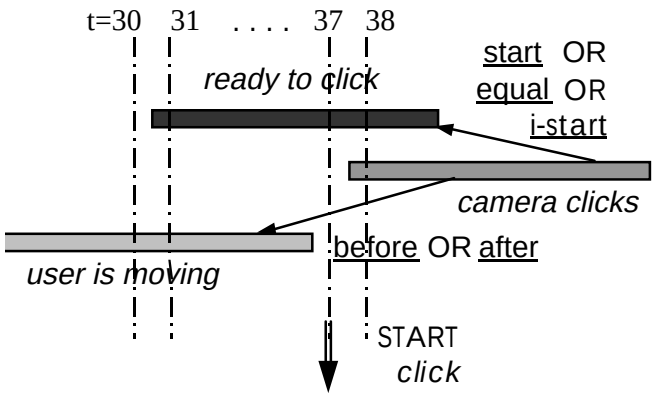
consistent, $D^1 = R\left(S^0 \overline{\cap} P^1\right) = R(P, N) = \varnothing$. By the recovery heuristics, we define the desired state to be $D^1 = (PN, PN)$. According to the stage IV of the run-time engine, only the start function of *A* would be called.

5.4.3 An Example of a Basic Run Without Conflicts

Figure 5.12 shows an example of run of the interval script of fig. 5.3. In the first instant of the run *t=0*, the state of all intervals is *F*. Although the result of PNF propagation allows the first two intervals to be either in the *now* or in the *future* states *(NF)*, the result of the thinning process, D^1 keeps the state as it is and no action is taken.

In the next instant of time, *t=1*, the interval *"camera moves"* has started, as detected by its STATE function. We ignore here the cause of the start of *"camera moves"*. When PNF propagation is run, the fact that *"camera moves"* and *"camera moving sound"* should start together constrains the desired PNF-state of the latter to be *N*, and provokes a call for its START function. In the next instant, *t=2*, we assume that *"camera moving sound"* is already running and therefore no further action is taken. The system remains in this state up to *t=9*.

When *t=10*, *"camera moves"* finishes and assumes the *P* state. Because of the constraints, *"camera moving sound"* should stop and *"camera zooms"* should start. Notice that the result of PNF propagation, P^{11}, shows exactly that configuration and the appropriate actions are taken. In *t=11* the desired state is reached for both intervals; if it was not the case, we would



interval	t=30			t=31				t=37			t=38		
	S^{30}	P^{31}	D^{32}	S^{31}	P^{32}	P^{32}	D^{32}	S^{37}	P^{38}	D^{38}	S^{38}	P^{39}	D^{39}
ready to click	F	F	F	N	N	**N**	N	N	N	N	N	N	N
camera clicks	F	F	F	F	∅	**F**	F	F	N	N	N	PN	N
user is moving	N	N	N	N	N	**N**	N	P	P	P	P	P	P

Figure 5.13 Example of a run with conflict.

have a state similar to *t=10*, and the START and STOP functions would be called again. Finally, *"camera zooms"* ends at *t=21* and all intervals assume the *past* state.

5.4.4 A Run With Conflicts

An example of a conflict situation is show in fig. 5.13, for the interval script depicted in fig. 5.5. Here, we consider the case where the intervals *"camera clicks"* and *"user is moving"* are defined as mutually exclusive and a conflict situation arises where both are expected to happen.

At *t=31*, a conflict is detected by the PNF propagation stage since there is no way to satisfy the requirements that *"camera clicks"* should start together with *"ready to click"* and that *"camera clicks"* can not happen while *"user is moving"* is occurring. In this situation the first level of recovery succeeds and finds a set of states that is compatible with the constraints as shown in fig. 5.13. However, P^{32} is quite non-specific and the thinning process basically keeps the current situation as it is without taking any action. Later, when *t=37*, "*user is* moving" finishes, and since *"ready to click"* is still happening, *"camera clicks"* is started.

As we see, the run-time engine of interval scripts was designed to avoid halts by searching for feasible states with the least amount of change. This strategy is not guaranteed to succeed always but has proved to be robust enough to run quite complex structures. We are currently working on better methods to overcome contradictions such as keeping a history of previous states for backtracking purposes (such as those employed in chapter 4) and devising mechanisms to detect and isolate the source of conflicts.

5.4.5 Compiling Interval Scripts

The design of the mechanisms of the run-time engine is independent of the actual implementation of the interval script structure. However, we think it is important to provide some clarification on specific implementation issues so future implementations can benefit from some of our ideas.

In our implementation the interval script file is translated into a C++ file where every START, STOP, and STATE function is encapsulated into a C++ function call. Those calls become methods in the definition of each interval. After compilation of the C++ code, the run-time engine can call any of the functions simply by calling the interval's method.

In parallel, the translator program builds an IA-network with all the declared intervals. Before generating the C++ code the translator program runs Allen's path consistency algorithm. The tightened network is then projected into a PNF-network as described in chapter 3. The resulting structure is then codified as a C++ function that, when executed, rebuilds the PNF-network for real-time use. Notice that it is not necessary to run Allen's algorithm before an execution of the script but only at compilation time.

Finally, the interval script language provides mechanisms by which a START or a STOP function can set the state of an interval for the next cycle $t+1$. This was included to facilitate some constructions. In the diagram of fig. 5.11, this is show as a dashed arrow bringing results from the run of the START and STOP functions to S^{t+1}.

5.5 Handling More Complex Structures

After examining the internal structure of the run-time engine for interval scripts we want to go back to the issue of expressiveness. In this section we introduce, through two examples, more advanced constructions that can be used to handle more complex scripting structures. Some of those constructions are not intrinsically part of the basic concept of interval scripts but were included in the definition of the interval script language because they were often needed in practice.

5.5.1 Nested Intervals

The first example refers to a segment of a scene to be incorporated to the installation *"It"* in which if the user makes a waving gesture and the computer character reacts by pretending to throw computer graphics rocks towards him/her. Before generating the animation, the computer waits about 3 seconds to create some suspense about its reaction.

Figure 5.14 shows the interval script of this scene, while fig. 5.15 displays a diagram of the temporal structure. It basically consists of one interval corresponding to the reaction of the system called *"annoy user"* and another that detects the *"waving gesture"*.

```
"annoy user" =                                                              (1)
{                                                                           (2)
"wait" = timer (2,4).                                                       (3)
"throw CG rocks at user" = { .... }.                                        (4)
better-if "wait" meet "throw CG rocks at user".                             (5)
        DURATION (5,7).                                                     (6)
        START:   start "wait".                                              (7)
        STOP:    stop "wait" ; stop "throw CG rocks at user" .              (8)
        STATE:   if "wait" is now OR "throw CG rocks at user" is now        (9)
                     set-state now                                          (10)
                 else if "wait" is past AND "throw CG rocks at user" is past (11)
                     set-state past                                         (12)
                 else if "wait" is future AND "throw CG rocks at user" is future (13)
                     set-state future                                       (14)
                 endif endif endif.                                         (15)
        FORGET: forget "wait" ; forget "throw CG rocks at user".            (16)
}.                                                                          (17)
                                                                            (18)
"waving gesture" = { .... }.                                                (19)
                                                                            (20)
when "waving gesture" is now try-to start "annoy user".                     (21)
```

Figure 5.14 A script including nested intervals, a timer, functions defined on intervals, and an example of a sequence.

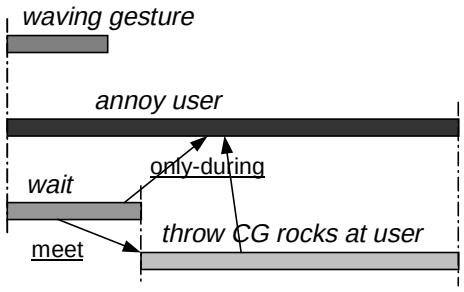


Figure 5.15 Diagram of the temporal structure of the script in fig. 5.14.

We have seen before that an interval can refer to other intervals for its definition. It is possible to go beyond that and define intervals inside other intervals as it is the case of *"wait"* and *"throw CG rocks at user"* (lines 3 and 4 of fig. 5.14). There is no limit in the number of levels of these *nested intervals*. The effect of doing so is that the nested intervals become restricted to occur only when the major interval is happening, that is, in this example they are constrained to happen in the context of *"annoy user"*. In practice, this is accomplished by the computer automatically setting a constraint only-during (equivalent to start OR equal OR i-start OR during) between each nested interval and the container interval.

Notice that unlike most programming languages, the nested intervals can be referred to outside the scope of the container interval; they are just prohibited to **occur** during run-time outside of it. Moreover, the engine computes the current state of those intervals before the state of the container interval; in the case that their state is used in the definition of the STATE function of the container (as it is the case of 5.14, lines, 9-15), the engine employs the current state of the nested intervals in the computation. This avoids the waste of propagation cycles that would be needed otherwise.

We are also considering using the nested structure as a way to recover from errors and contradictions. As mentioned before, it is hard to determine the source of contradiction in constraint satisfaction. Our plan is to run PNF restriction separately in each group of nested intervals, from the most inside intervals up to the higher level. If a contradiction is detected, all the intervals in the group are assigned PNF and restriction is run again. In this way we can explore a structure that is naturally provided by the designer to exclude potential causes of conflict.

5.5.2 Timers and Duration

Controlling duration is often important when scripting. We incorporate two simple mechanisms to facilitate the handling of the length of the intervals. First, it is possible to declare an interval to be a timer; that is the case of *"wait"* (line 3 of fig. 5.14). Inside parenthesis the programmer specifies the minimum and maximum duration of the timer. When a timer is started, the corresponding interval is set to the *N* state and a duration value is randomly selected between the minimum and maximum value. When the time expires the interval assumes the *P* state.

A similar construction can be used to specify the duration of an interval. By using the declaration DURATION in fig. 5.14, line 6, the interval *"annoy user"* is defined to last between 5 and 7 seconds. That is, after the time allotted to run expires, the interval is set to *P*, and due to constraint propagation, all the nested intervals will start to call their STOP functions in order to satisfy the implicit only-during constraint (as explained above). Notice that the duration of the interval is not used in any way in the process of propagating temporal constraints. For that, it would be necessary to base the run-time engine in a more powerful — and computationally expensive — temporal reasoning than PNF propagation.

5.5.3 START and STOP Functions Based on Intervals

As much as STATE functions can be based on previously defined intervals, the interval script language also allows START and STOP functions to be purely described in terms of other

intervals. The interval *"annoy user"* is a typical case. Its START function simply calls for the execution of the interval *"wait"* (see line 7 of fig. 5.14); similarly the STOP function asks for the two nested intervals to stop (line 8).

In our initial designs we have distinguished between the kind of commands that could be used in STATE functions and in START and STOP functions. Currently there is no such distinction but commonly a STATE function includes a command set-state that actually sets the current state of the interval. We commonly found it necessary to set the state of an interval directly by the START functions because many times there is no easy way to create a STATE function that actually senses the state of the world.

5.5.4 WHEN Statements

The last lines of fig. 5.14 define the *"waving gesture"* interval (based on C++ code not shown here) and establish its connection with the interval *"annoy user"*. According to the scene, *"annoy user"* should be started whenever *"waving gesture"* happens. The statement shown in line 21 accomplishes this.

Why not simply put a temporal constraint (start OR equal OR i-start) between the two intervals? The problem is that in many cases the condition is a Boolean expression of states of intervals that can not be translated into a simple temporal constraint. Also, we wanted a simple mechanism to describe a series of start and stop actions. Our solution is to make the translator automatically generate a new interval with a STATE function equivalent to the condition of the when command.

If the when statement triggers the start of an interval, the system sets up the temporal constraint start OR equal OR i-start between the interval corresponding to the condition of the when statement and the interval to be started. If the statement asks for the stop of an interval, the imposed constraint is meet OR before. Notice that when statements may only command the start and the stop of intervals; in particular, execute commands can not appear inside when statements (refer to the complete syntax of the interval script language in appendix D).

Going back to the example of fig. 5.5 , we could have avoided the definition of *"ready to click"* entirely and simply written

```
when  "camera zooms" is past AND  "user is posing"  is now
   try-to start  "camera clicks".
```

If declared inside an interval, a when statement is affected by only-during constraints, that is, its condition is not evaluated if the container interval is either in *F* or *P* state. That makes the control of the context of when statements extremely easy, unlike in many event-loop-based languages.

5.5.5 Sequences

In fig. 5.14 the interval *"annoy user"* is defined as composed of two consecutive, nested intervals. Since sequences of two or more intervals are fairly common, we defined a special

kind of interval to automatically create and manage sequences. The whole definition of *"annoy user"* (lines 5 to 16 of fig. 5.14) can be created simply by writing:

> ***"annoy user"*** = sequence *"wait"*, *"throw CG rocks at user"*.

This definition automatically creates a new interval containing the members of the sequence, defines the corresponding STATE, START, and STOP functions, and establishes the only-during constraints. To assure the sequential run of the intervals composing the sequence, the system automatically sets up a constraint meet between each pair of consecutive intervals.

5.5.6 Forgetting Intervals

Let us now examine another example that is a continuation of the script of fig. 5.14. The corresponding interval script is shown in fig. 5.16 and its temporal diagram is displayed in fig. 5.17. This script deals primarily with the recognition of an action of the human user: picking up a block from the floor. In chapter 4 we have already discussed the typical problems and difficulties in detecting human action. This example is, in fact, very similar to those examined in chapter 4. However, it is interesting to see how the interval script run-time engine handles such situations.

The main cause of complexity in this script is the limitations of the vision system used in the actual system of the installation *"It"*. Described in detail in chapter 7, the vision system works solely based on rough silhouettes of the users and of the wood blocks used in the scenario. We will discuss this in further detail soon. But before going ahead, it is important to examine first the concept of *forgetting intervals* that occurs in this script.

Until now, all the examined intervals were supposed to happen only once, going from the *F* state, to *N*, and finally becoming *P*. Although this is the correct model of what happens in the world, conceptually it is simpler for the designers to consider that an action or sensory state can happen again. Figure 5.16 shows two of such intervals, *"block is seen"* and *"block is not seen*" (lines 1 to 6).

The interval *"block is seen"* corresponds to the situation where the silhouette of a small object is detected isolated from the silhouette of a person. The definition of the interval uses a C++ call to the vision system as shown in line 2. There is no equivalent function for *"block is not seen"* whose definition in fig. 5.16 is empty. However, through the mutually exclusive constraint of line 3, we assure that the end of *"block is seen"* triggers the beginning of *"block is not seen"* and vice-versa (see also fig. 5.17).

The natural meaning of a statement like *"block is seen"* suggests that this is a state that becomes true and false as time goes by. Strictly speaking, there is no semantic provision in the PNF structure to support this: in formal terms, a new occurrence of an interval is a new interval. However, due to the simplicity afforded by having just one interval to name a sequence of states, we implemented a mechanism by which an interval can be forgotten, that is, can go from the *past* state back to the *future* state. Notice that this mechanism is intrinsically different from the definition of multiple sensor instances that we used in chapter 4.

```
"block is seen" =                                                    (1)
    { STATE:    set-state pnf-expression [>(block.isSeen()?_N_:P_F)<]; }.   (2)
"block is not seen" = { }.                                           (3)
better-if  "block is seen"  meet OR i-meet  "block is not seen".     (4)
when  "block is seen"  is past  try-to forget  "block is seen".      (5)
when  "block is not seen"  is past try-to forget  "block is not seen".  (6)
                                                                     (7)
"user same position as block" =                                      (8)
    { STATE:    set-state pnf-expression                             (9)
              [> (block.isSameAsUser() ? _N_ : P_F) <];              (10)
    better-if "block is seen" before OR meet OR overlap "user same position as block".  (11)
    }.                                                               (12)
"playing with blocks" =                                              (13)
{                                                                    (14)
"user picks up block" = { }.                                         (15)
"user has block" =                                                   (16)
    { STATE:    if  "user same position as block"  is past           (17)
                    AND  "block is not seen"  is now                 (18)
                  set-state now                                      (19)
    .             endif }.                                           (20)
better-if  "user picks up block"  only-during  "user same position as block".  (21)
better-if  "user picks up block"  meet  "user has block".            (22)
                                                                     (23)
STATE:  if  "user has block"  is past  set-state past.               (24)
}.                                                                   (25)
better-if  "waving gesture"  only-during  "user has block".          (26)
```

Figure 5.16 Example of a script with recycling intervals and user action recognition.

Forgetting mechanisms are defined by the FORGET function, which is syntactically similar to the basic functions STATE, START, and STOP. In our language there is no difference in the kind of commands allowed in the definition of any of those functions. Also, each interval has a default FORGET function that simply moves the current state of the interval to *F*. If a FORGET function is defined in the interval script, the default function is ignored.

Lines 5 and 6 show a simple method to forget an interval that is simply using the FORGET functions to call the corresponding function whenever the interval goes to *P*. According to lines 5 and 6, if either "*block is seen*" or "*block is not seen*" become *P*, it immediately returns to *F* unless conflict is found (that is handled like the conflict of fig. 5.13). As we see in

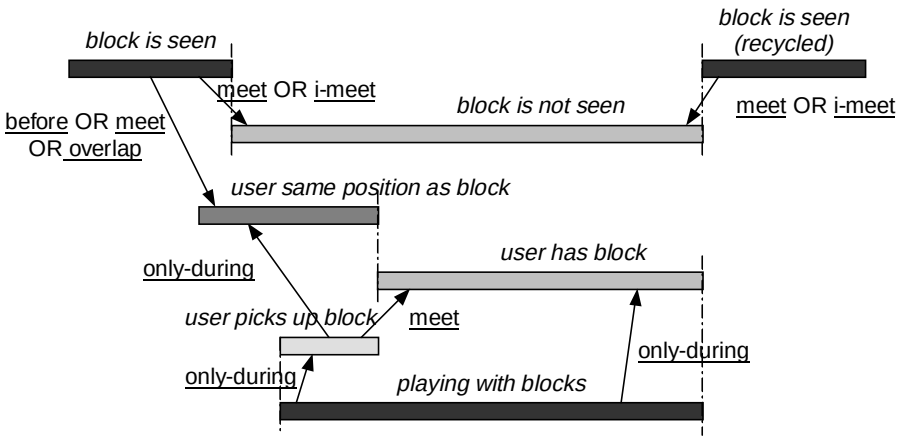


Figure 5.17 Diagram of the temporal structure of the script in fig. 5.16.

fig. 5.17, the interval *"block is seen"* happens twice, the second time after being forgotten once.

A more elegant solution for the problem would be to generate, during run time, new instances of the interval to be forgotten as needed. This would require the running of Allen's path consistency algorithm after each new instance was incorporated to the main network what would slow down the engine. Also, the size of the network would grow as time went by. In practice, given that the architecture of the run-time engine is robust to conflicts, we have not found the need to implement this version of forgetting, and have been successfully using the simple method of moving an interval from *P* to *F*.

5.5.7 User Action Recognition

Let us now examine how interval scripts can facilitate the recognition of user actions. Although these issues are addressed in detail in chapter 4, we present here a simple example that shows how the interval script run-time engine deals with the same situation of recognizing what a user is doing given incomplete information from the sensors. The task in question is to determine that a user has picked up a block from the floor given just two bits of information: whether the block is currently seen by the vision system, which is possible only if the block is far from the user; and whether the user is in the same position that the block used to occupy. In particular, after the user has picked up the block, there is no way to determine, from the human silhouette, that the user actually has the block.

These constraints imply that, the pick-up action is detected only when the following sequence of events happens: (i) the block is seen isolated; (ii) the user gets close to the block; (iii) the block is not seen and the user has left the area where the block used to be; (iv) the user is supposed to be carrying the block; (v) since the only way for the user to have the block is that he/she has picked it up, the action of picking up is recognized to have happened in the past. The reader may be surprised by how indirect and prone to failures such a method is. However, far

from being an exception, this situation exemplifies what typically happens in computer vision systems.

In the script of fig. 5.16 we initially define in lines 8-11 the interval *"user same position as block"*. Following that, an interval called *"playing with blocks"* is defined, containing two intervals: *"user picks up block"* which has no direct method to identify its occurrence; and *"user has block"* which has a STATE function that checks if the condition described above in (iii) is occurring. To determine the state of the interval *"user picks up block"* two temporal constraints, lines 21 and 22, impose that picking up the block occurs while the user is in the same position as the block; and that the action is finished when it is determined that the user has the block.

Notice that both constraints are "natural", in the sense that they arise as part of the structure of the action itself. Although the detection occurs in a very indirect way, the definition of the intervals themselves does not need to involve any complicated reasoning. This is a major feature of non-procedural, constraint propagation-based systems such as the one employed in interval scripts.

5.5.8 Contextualizing Action

Finishing the script of fig. 5.16, we show how simple it is to enforce that an action or sensory state is only detected in a given context. In the scene, the waving gesture (referred in the previous script) can only occur if the user has the block in his/her hands. This is guaranteed simply by establishing the only-during constraint between the intervals as show in line 26.

5.6 Future directions

Although the interval script language has been fully implemented and intensively tested, there is still work to be done in many different directions. First, we would like to get the system to a state where it could be released to other researchers and designers interested in using interval scripts to build interactive systems. We expect to reach this level soon. In this context it becomes possible to evaluate the difficulty in learning the language and which features different types of designers would like to have added to the language.

Designer's Interface to Interval Scripts

We also want to investigate what is the ideal interface for the paradigm: text-based or graphical. In the case of a graphical interface, a possible approach is to allow the user to construct temporal diagrams similar to fig. 5.16. However, since we allow disjunction of temporal constraints, there are always multiple possibilities for the graphical arrangement of the intervals and therefore the visualization might prove to create more confusion than to provide help. Nevertheless, we have found it is always useful to draw the diagrams while thinking about the temporal constraints.

Interval Scripts and Plans

We would like to integrate into the interval script language mechanisms that represent actions in forms that the machine can reason about. Currently, the run-time engine can only control the execution of scripts of action plans but has no mechanisms to create them. In fact, the use of the PNF formalism does not allow the system to perform almost any kind of planning or look ahead.

Comparing, for instance, with the *Hap* system developed by Loyall and Bates [87], the interval script paradigm can not predict the execution of actions in a situation, for instance, in which to achieve a goal in the future, a determined sub-action is started in the current instant of time. In other words, although interval scripts can run and control interactive plans with very complicated temporal structure (certainly more complex than most planners can handle), the formalism can not construct plans from goals. This limitation is partially due to the ideas employed to reduce the complexity of temporal constraint propagation in Allen's algebra (as described in chapter 3) which compacts all the interval occurrences inside three intervals, *past*, *now*, and *future*.

We have considered different mechanisms to overcome this limitation. In chapter 3 we show that the bookkeeping of multiple threads of sensor values can be handled because most of the threads are inconsistent after contradictory sensor values occur. In that chapter, this was done in the context of keeping multiple possibilities for the events in the past. Similarly, we could use multiple threads of sensor values to search the future for an activation pattern that achieve a determined goal expressed as a combination of PNF-states for the different intervals of a script.

To accomplish this, it is necessary to incorporate mechanisms that forecast the likely duration of actions and the likely delays in starting and stopping them, so the step-by-step PNF-style exploration of the future becomes feasible. Traditional plan mechanisms [87, 116] avoid dealing with these issues by assuming strictly sequential sub-actions and states, and therefore they can search the future by ignoring the actual duration of the actions. In fact, those systems examine the state of the world exactly before or after the performance of an action, ignoring the possibility of other state changes while they are occurring.

The research question is how simplified a scheme of estimating the duration of actions in interval scripts must be so that the estimation is at the same time realistic and computationally feasible. Although it is likely that many of the threads representing sequences of future events result in inconsistent states, there is no reason to expect the same degree of pruning in the prediction of the future that we observed in the action recognition methods described in chapter 4. Therefore, to allow a reasonable depth in the search, it would be necessary to assume simplifying conditions for the sensors and actuators. The nature and actual pruning power of such conditions is clearly an open and difficult research topic.

Using Interval Scripts to Describe Scenes in CG Movies

Finally, we want to apply the interval script paradigm to computer graphics animation problems, especially the coordination of computer graphics scenes with multiple, behavior-based characters. We see here a major chance to improve the current techniques used in the

movie industry and to allow the creation of scenes with vivid, interesting crowds that are able to follow directorial directions at the scene level by modifying or suppressing their behaviors when requested.

5.7 Summary

In this chapter we propose the interval script paradigm for scripting interaction that is based on declarative programming of temporal constraints. Through the examples written in the interval script language we have shown that the paradigm is expressive and significantly facilitates the management of context, story, and history in an interactive environment. We also argue that the use of PNF propagation as the basic propagation technique can make feasible real-time temporal constraint satisfaction in a strong temporal algebra with mutually exclusive constraints.

From the experience acquired in the use of the language for the implementation of three different projects, we believe that scripting systems incorporating the interval script paradigm can significantly ease the design and building of interactive environments. The projects are described in full detail in chapter 7.

6. An Architecture and a New Domain for Story-Driven Interactive Spaces

One defining characteristic of our work is an interest in interactive spaces that encompass a narrative structure. The goal of this chapter is to address such story-driven interactive spaces and to present a new idea concerning their control architecture —the *story-character-device architecture* — and to introduce a new domain for experimentation — *computer theater*. In this way, we can regard this chapter as a contribution for the science — and art — of building interactive spaces.

But what we mean by a story-driven interactive space? In section 6.1 we discuss the more general idea of interactive (computerized) spaces, reviewing some prototypes that have been developed in recent years and presenting relevant technology. Instead of giving an exhaustive review, the intention is to identify the critical aspects of a physical environment that condition the development of interactive spaces. We then examine in more detail interactive spaces that immerse their users in a story and the challenges in their design and implementation.

From our experience in developing story-driven interactive systems for entertainment (*"The KidsRoom"* [18], and the three projects described in the next chapter, *"SingSong"*, *"It / I"*, and *"It"*) we have concluded that stories require the interactive space to have a much more complex control structure than those of traditional, exploratory, user-driven interactive systems. Section 6.2 discusses control architectures for systems with multiple characters and narrative structure. There, we propose a three-level architecture, the *story-character-device* architecture, as the underlying structure for the control of story-based interactive systems. The main novelty in our proposal is the differentiation between the control of the characters and the control of the story. In particular, we argue that story-based interactive experiences need centralized story control and the simplest way to achieve it is to have a specific module to manage the story.

We then identify a new domain for experimentation with interactive spaces: the theatrical stage. In section 6.3 we examine this domain in the large context of the use of computers in theater. Although there have been some experiences coupling theater and computers in the past, we believe we were the first to examine systematically experiences aiming to integrate computers into theater in [128]. This investigation resulted in the proposal of the term *computer theater* to

refer to these experiences. Computer theater is discussed in section 6.3 which also provides a brief review of the history of computers in theater, proposes a categorization of experiences in computer theater, and elaborates on why there have been significantly fewer experiments with computers in theater than in other performing arts (music, dance). According to our view, the main reason is the lack of good methods for computer representation of human action — the ultimate element of theater.

We also see a convergence between computerized performances and traditional, user-based interactive spaces. This is discussed in section 6.3 where we present the idea of *immersive stages*, that is, interactive story-based spaces that double as stages for performances and as narrative art installations for non-performers.

6.1 Interactive Spaces

In this thesis we restrict the use of the term *interactive space* to situations where human beings interact with a computer in a physical space through digital input/output devices such as cameras, microphones, video screens, and speakers. The classic example of interactive space is the control room of spaceships as portrayed in science-fiction series and movies like *"Star Trek"* and *"2001:A Space Odyssey"*. In these two cases, the computer that controls the spaceship is omnipresent, interacting with the ship's crew as an invisible, God-like figure that can either access the most vital data or entertain the occupants with a game of chess.

Perhaps the most characteristic feature of an interactive space is precisely its omnipresence. It is hard to identify the locus of agency in a physical space that speaks from different directions and that observes from multiple cameras. In fact, there is something frightening about the idea, especially if the interface is anthropomorphized like in *"Star Trek"* or *"2001"*. We explored this fear as part of the theme of our computer theater play *"It / I"*.

We can see many different applications for systems that encompass the multiple devices and uses of a room. For example, the concept of a house that is aware and responsive to its inhabitants has been a constant in industry showrooms. Another application is specialized physical spaces where critical activities take place such as control rooms of factories and surgery rooms. More recently, the idea of spaces that immerse people in fantastic worlds and adventures have become popular, heralded by the myth of the *Holodeck* in the TV series *"Star Trek"*.

6.1.1 Characteristics of Interactive Spaces

Let us examine some common characteristics of interactive spaces. First, it is a **physical space**. For instance, we do not consider either virtual reality or videogames as interactive spaces in our definition. In other words, we are interested in systems that are required to assume the existence of atoms, the physical interaction among them, and the difficulties in sensing real matter (as opposed to checking the state of virtual entities).

We also assume the existence of a **computer system** controlling the interaction, discarding human-controlled environments like a disco club (where the DJ manages the interaction between the patrons and the dance area). Moreover, interactive spaces must be **responsive**, that

is, they must acknowledge the attempt of interaction by the users either by answering directly or by producing a detectable transformation in the space.

Another aspect is that **agency** should be attributed not to particular objects in the space, but to the space itself. Notice that in the case of entertainment spaces inhabited by virtual or physically represented characters, the agency of the characters has clear locus, but the space is still the non-localized source of the story, game, or narrative. Part of the appeal of the idea of interactive spaces is exactly that, perhaps for the first time in history, we can see a space as dialoguing entity. Although architecture theory sees spaces affecting people and vice-versa, such interaction occurs in a time scale of months and years (see, for instance, Stewart Brand's work on how buildings "learn" with their occupants [28]). In contrast, computers and sensing devices create spaces that are machines, that is, perceived as animated objects.

Given this context, it is important to examine the particular issues involved in the construction of interactive spaces. The most relevant is the issue of sensing that, unlike in the virtual domain, becomes extremely complicated. As it will become clear from the projects we built, the most severe constraint on the design of an interactive space is which and how well the activities, intentions, and wishes of its users can be detected. The four systems we built have been carefully designed to make the detection of all the human actions possible using current technology.

Another characteristic is that the **activities happening in a physical world take time**. Unlike interaction in computers that is event-based, it takes time to move an arm, to walk around, or to utter a sentence. A lot of research in multimedia systems has dealt with the problem that producing output takes time (including here issues like delays, bandwidth, etc.), but the fact that input can also extends in time has been mostly ignored. As we have already presented, a central point of this thesis is the study of methods to represent, recognize, and control activities with complex temporal structure, and that grew largely as a response the perception that the currently-used simple models for time are insufficient to handle typical situations in complex environments.

We can roughly divide sensing systems in interactive systems according to the modality being sensed: vision, audition, touch, and presence. Cameras are very popular to track the presence and the position of the occupants of interactive systems [18, 39, 79, 94], sometimes in the infra-red spectrum [44]. Microphones have been used mostly associated with speech recognition systems [39], but also to detect simple sound events like screams [18]. Pressure-sensitive floors have been developed and employed in the context of interactive spaces [41, 119]. Besides that, it is possible to make machines detect presence of objects and people by monitoring disturbances in electric, optical, and magnetic fields [88, 119]. A good review of various interactive spaces can be found in [15].

6.1.2 User-Driven Interactive Spaces

We employ the term *user-driven* to characterize interactive spaces where the user directs the direction and goal of the overall interaction. This category, in fact, comprises the majority of the work done in terms of creating actual interactive spaces. One of the pioneers of the idea is Mark Weiser, who has been proposing the idea of *ubiquitous computing* for more than a

decade [179]. In its simplest form, ubiquitous computing advocates the disappearance of a visible entity associated to computational power and its spread through space and everyday objects. His view is clearly echoed in our previous discussion of how agency in interactive spaces is associated to the space itself.

Alex Pentland has promoted the idea of *Smart Rooms* as a domain of perceptual computing [122]. The most successful result was the *ALIVE* experience [94], where a user could interact with a CG-generated dog (build by Blumberg [17]) by watching herself in a *virtual mirror* — a large video screen showing a mirror-like image of the room with the user and superimposed computer graphics objects. The *Intelligent Room*, being developed at the MIT AI Laboratory [39], aims to create a room for crisis management that interacts with its occupants and provides them critical information. It incorporates simple camera-based tracking, gesture detection, and reacts chiefly by answering verbal commands. The control is distributed according to Minky's concept of *society of mind* [106]. Raskar et al. [141] suggested a design for the *Office of the Future* where ceiling lights are substituted by computer controlled cameras and projectors that, under the user command, display high-resolution images on designated display surfaces.

Interactive spaces have been mostly developed in the context of entertainment and art, starting with the pioneering work of Myron Krueger [79]. Krueger's works are capable of creating strong engagement using simple elements. For example, in his first major work people explored an empty space that reacted with sound and imagery. We may credit his ability to design enticing spaces to the fact that in earlier works he would personally control the room (instead of a computer). As once observed by him, being in charge of the interaction makes designers extremely aware of the diversity of the users' behavior and the many different ways to relate to a space.

Tod Machover created a large-scale interactive space with multiple music-producing objects, the *Brain Opera* [89, 119]. Although the room itself was not interactive, the final effect created in practice a feeling of agency, somewhat related to the theme itself — the emergence of intelligence as postulated by Marvin Minsky in *"The Society of Mind"* [106]. Jim Davis and Aaron Bobick, also at the MIT Media Laboratory, created the *Virtual Aerobics* environment [44] where the room becomes the personal trainer of the user, demonstrating and motivating the user to exercise, and making sure that the user actually works out.

The review of the innumerous other projects involving interactive space is beyond the scope of this thesis. As we commented in the introduction of this chapter, our real goal here is to discuss in depth some methods and concepts we have developed that facilitate the building of real-world interactive spaces, and, in particular, for spaces that are driven by a story.

6.1.3 Story-Driven Interactive Spaces

Narrative and story structures are very common in computer games. However, the number of physically interactive spaces with such elements is very restricted, although the interest in the area has considerably increased in the last years. We use the term *story-driven* interactive spaces to refer to spaces that immerse the users in a story that develops partially in response to

the users' actions and partially as a consequence of a sequence of events pre-determined by the designers of the space.

Interactive stories and narratives is still a genre to be born. In [109], Janet Murray analyses different aspects and experiences about how to create a story that changes in response to user actions and/or wishes. She observes that there are particular narrative genres that are more suitable for interaction, such as journey narratives. In particular, Murray studies multi-threaded stories where the user has mechanisms to choose different paths in a web of events, creating an individual, personal story as a result of the interaction.

In the realm of interactive spaces, the first experiments were also created by Myron Krueger [79], although most of his "narrative" installations portrayed very simple stories. Larry Friedman and Glorianna Davemport's *Wheel of Life* was one of the first works to include complex and rich narrative elements in an interactive space [41]. In this piece, the users would go through four different worlds, interacting with computer creatures and multimedia spaces representing water, air, earth, and fire. However, the level of control required was beyond the technology of its time and the system relied on human "guides" controlling the activities of the computers.

Placeholder, a project by Brenda Laurel, Rachel Strickland, and Rob Tow [84], aimed to integrate physical and virtual reality elements through a narrative about indigenous peoples of Canada. Although it is not clear if the project was ever completed, their description of the creation process [84] is one of the best discussions in literature about the issues involved in story-driven interactive spaces.

Naoko Tosa and Ryohei Nakatsu, at the ATR Laboratory in Japan, have created two interactive pieces with strong narrative structure. In the *Interactive Poem*, the user dialogues with a female stylized face projected on a large computer screen [174]. The objective of the piece is to create a poem by interchanging verses between the computer and the user. The computer says a verse and prompts the user to choose among three verses displaying on the screen by reading the verse aloud. Coupled with music and beautiful imagery, the final sensation is of integration and fulfillment. Tosa and Nakatsu have also created *Romeo and Juliet in Hades*, where two users take the roles of Romeo and Juliet after their death and, through speech and gesture interaction, follow their journey towards rediscovering who they are and their love [173].

The work of Aaron Bobick at the MIT Media Laboratory has pushed the sensing and the awareness envelope in the recent years. In *"The KidsRoom"* [18], a children bedroom-like space takes a group of children through an adventure covering four different worlds inhabited by friendly monsters. The interaction was very physical, involving running, dancing, jumping, and rowing. The system relied almost exclusively in visual sensing and was able to provide children with a complete and engaging experience with a narrative structure.

In many ways, three of the projects described in the next chapter, *"SingSong"*, *"It / I"*, and *"It"* can be considered among the most compelling examples of the possibilities of story-driven interactive spaces. They also differ from most of the examples mentioned above in our emphasis in **responsiveness** as the key element of creating immersion in a story, instead of the more commonly used device of choice among different story paths. Although choosing the path

of a story considerably empowers the user, it is very difficult to assure that all of the paths lead to equally satisfying experiences. In contrast, we have created powerful stories where the user is coerced, through mechanisms not explicitly visible, to remain inside the main path, while carefully designed interaction and good responsiveness keep the illusion that the story is unfolding in response to the user's actions.

6.2 The Story-Character-Device Architecture

Our work in building interactive spaces has focused in spaces that move their users through a story or interactive narrative. Not surprisingly, managing an interactive story with multiple characters and different input and output devices proved to be quite challenging. For reasons discussed later in this section, we decided to have the story managed by a single module, instead of the more common practice of distribute it through the characters' control mechanisms (for example in [14, 52, 125]).

In the project *"SingSong"* (described in the next chapter) the control is centralized in one module that contains both the story and the control of the behavior of the different characters. The control of the *"The KidsRoom"* [18], a project that we participated in the development, was also similarly structured. Based on the experience of these projects, we have identified that a common source of problems is an inadequate division between the control of a character and the control of the story.

For example, in the *"The KidsRoom"* there is a scene where computer graphics monsters interact with kids who are facing the screens where the monsters are projected, one kid per monster. The children perform dance steps that, when recognized by the monsters, make the system play audio files with the monster voices congratulating the corresponding child. A common source of problem is that two monsters can not speak at the same time — otherwise they are not properly heard. Managing this level of detail proved to be a difficult task for the developers of the control module. In the example above, we believe that the right approach is to make the character's responsibility to assure that it communicates to the children: the control of the story should be concentrated on getting the right succession of actions to take place.

To overcome these difficulties, we have developed an architecture for interactive, story-based systems that separates the control of story, characters, and devices into different, isolated layers — the *story-character-device* architecture, or *SCD*. This is basically a conceptual division but we will argue that it simplifies considerably the development of story-based systems.

6.2.1 Centralized vs. Decentralized Control of Story

Most of interactive spaces [79, 94, 161, 172] and virtual reality experiences (for instance, [84]) are based on the concept of *exploration*. That is, the user enters a world populated by characters, objects, and other users, and extracts most of its satisfaction from the encounter with the new. There is no narrative structure unfolding during the experience and therefore no need for story representation or control.

However, the existence of a story in an interactive system requires the management of multiple characters and events in orchestrated ways. A good image of this distinction is to observe that

while exploratory worlds have *creatures* living in them, story-based interaction requires *actors.* Actors know that a story must start, develop, reach a climax, and finish. Stories require the coordinated efforts of a cast of characters played by actors and thus any story-driven interactive system must have some mechanism to control the different actors.

An important question is where the story is represented and how it is controlled. For example, in most story-based interactive systems created until now, like Perlin and Goldberg [125] and Bates et al. [14], the story is carried by (semi-) autonomous computer-actors with partial knowledge of the story contents and development. The story is seen as the natural result of the interaction between the characters, the user, and the story traces in the character's brains.

We believe that centralized story control is very important for an interactive system to achieve successful story development. As pointed by Langer [83] and Murray [109], well-constructed stories require coordination and synchronicity of events and coincidences. For example, in Shakespeare's *"Romeo and Juliet"*, if *Romeo* does not arrive at *Juliet*'s tomb before she wakes up, we lose one of the most beautiful endings ever conceived. Coincidence is a powerful source of drama and a fundamental component of comedy. However, the coordination required to achieve coincidence, in our view, is only possible with centralized story control.

Also, as brilliantly pointed by Langer in her study of theater ([83], chapter 17), it is essential for dramatic structure to **forecast the future**:

> *"Before a play has progressed by many lines, one is aware not only of vague conditions of life in general, but of a special situation. Like the distribution of characters on a chessboard, the combination of characters makes a strategic pattern. In actual life we usually recognize a distinct situation only when it has reached, or nearly reached, a crisis; but in the theater we see the whole setup of human relationships and conflicting interests long before any abnormal event has occurred…(…) This creates the peculiar tension between the given present and its yet unrealized consequent, "form in suspense", the essential dramatic illusion." (Suzanne Langer,[83] , pg. 311).*

If we agree with Langer, it is necessary for an interactive system with dramatic structure to have, in some form, the ability to look ahead and sketch in the present the conflict of the future. To leave this job for each character, separately, and to expect that they will all come up with the same story structure is nonsense. A similar problem is faced by improvisational actors and the common solution is for an improv troupe to have well-defined story structures in stock and to choose one of the stories in the beginning of an improvisational performance (see [73]). Centralized control of story is the necessary answer to guarantee *"form in suspense"*.

Finally, if we want also to change a story as it progresses (as a truly interactive narrative), the imperative of central story control becomes even stronger. It is hardly feasible to alter the story if each character has its own beliefs about the current situation of the narrative and its future developments.

6.2.2 An Architecture for Interactive Story-Based Systems

Interactive spaces are normally complex structures involving multiple inputs, outputs, and control sources. In the interactive space *"It"* described in the next chapter we have used a particular software architecture called the *story-character-device* architecture, or simply, the *SCD* architecture. The concept was developed to overcome some difficulties observed in the construction of *"SingSong", "The KidsRoom"* [18], and, to a lesser extent, in *"It / I"*.

Like most of the work in software architecture, it is hard to characterize the success and even appropriateness of a particular model. We do not claim that SCD is the right or best architecture for a story-based interactive system. The reasoning presented below clearly shows that the model addresses issues that have been neglected by most of previous proposals, although it would certainly be possible to use other models (like Cohen's [39]) in the systems we built. However, our experience in *"It"* has shown that the SCD architecture provides a good framework for design and implementation of interactive spaces.

Figure 6.1 shows the basic elements of the SCD architecture that is composed of five levels. The *world level* corresponds to the actual sensors and output generators of the system. The *device level* corresponds to low-level hardware and software to track and recognize gestures and speech from users, and to control the output devices and applications (including low-level control of the movement of computer graphics characters). The *character level* contains software modules that control the actions of the main characters — including here the users and the virtual crew (e.g. cameraman, editor, and light designer) — by considering both the story goals and constraints and the actual events in the real space. The *story level* is responsible for coordinating characters and environment factors, aiming to produce the interactive structure contained in an *interaction script*. Finally, it is possible to have a *meta-story level* that dynamically changes the story as a result of the on-going interaction; such a module should encompass knowledge about how stories are structured and told [30].

The fundamental distinction with previous architecture models ([14, 125] is the existence of a distinct story level, separated from the character level. As discussed above, we do not believe that story development is achievable simply by character interaction but rather that central control is necessary. On the other hand, preserving the distinction between story and characters simplifies considerably the management of the story. Notice that in the SCD architecture the characters are semi-autonomous since they receive commands from the story control module. However, they still possess the ability to sense the on-going action and to coordinate the accomplishment of their goals according to the reality of the interactive space.

The best analogy to the *story level* role in the SCD architecture is the old theatrical figure of a *whisperer* or *prompter*, the person who used to sit underneath a small canopy in the front of stage, facing the actors, and whispering to them the next lines to be said. In this model, the actors are responsible for determining **how** the lines are uttered and for making up the accompanying action; the whisperer's job is to assure that story proceeds according to the script, that is, to determine **which** and **when** the lines are said. In the SCD architecture, the story level watches the user actions (regarding the user as just another character) and tries to adjust the story to match his actions, according to a script that describes how the interaction is

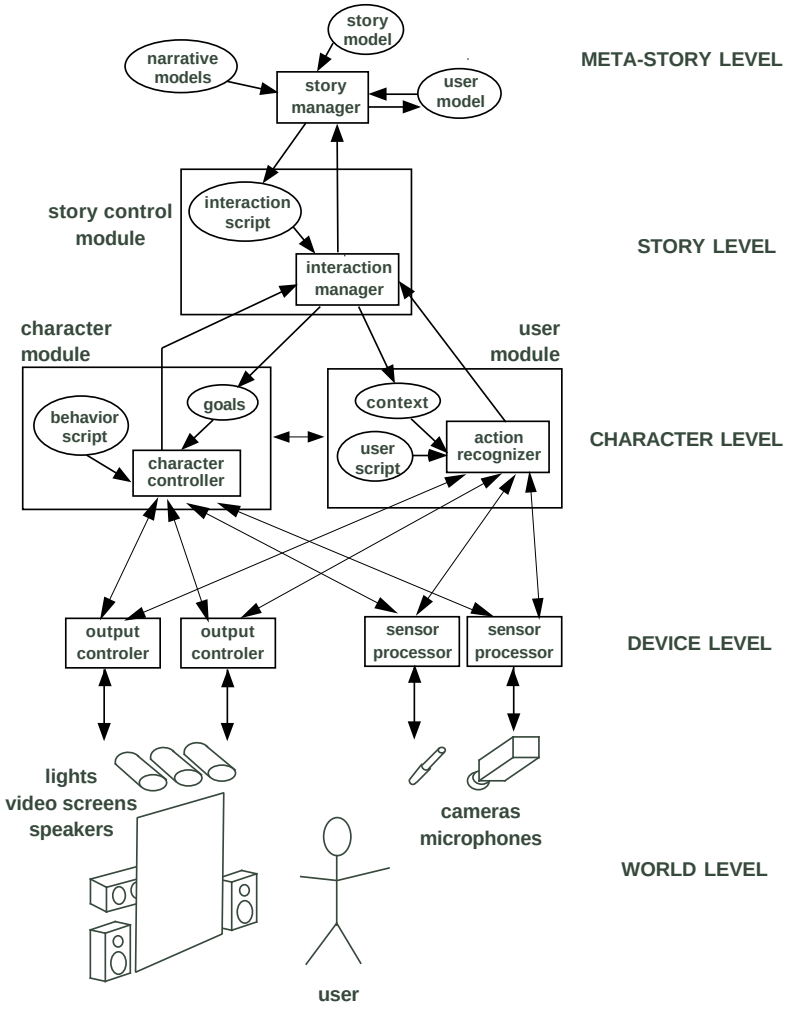


Figure 6.1 The SCD architecture for interactive story-based systems.

supposed to happen. If the meta-story level is present, the script itself can change — as if the printed text in front of an old-timer prompter suddenly shuffled its words into a new plot.

Since the SCD model segments the story components into different levels and assuming that actions are the main component of stories, the issue of **action representation** becomes extremely important because the modules have to communicate among themselves. Notice that to exchange messages between the story and the character levels it is necessary to use languages able to express goals, requests, and failures. Similarly, the encapsulation of the user monitoring in a module isolated from story and from the sensing devices requires methods to explicitly label actions (unlike in story-free reactive systems where sensor and actuators can be linked directly [172]), strengthening the need of **action recognition**.

Although the project *"It / I"* was designed with the concept of SCD architectures in mind (the idea predates the project), our first work that fully followed the model is the art installation *"It"* described in the next chapter. Since the story behind *"It"* is basically the same as the story in *"It / I"*, we can compare the impact of the different architectures in the process of building the systems. As described later, we found that separating story and characters considerably simplified the development of *"It"*.

6.3 Computer Theater

A common problem of traditional interactive spaces is that their users have to learn how to be seen, heard, and understood while the interaction is happening, and many times end up by getting lost. Detecting that the user is lost and providing adequate help is a major challenge faced by designers of interactive spaces given the still primitive state of sensor systems.

We found that an interesting type of environment where this problem is not present is a *performance space*, that is, a space where *"… an activity [is] done by an individual or a group (performers) in the presence of and for another individual or group (audience)."* (as defined by Schechner [151], pg. 30). Typical examples of performance spaces are classrooms, conference rooms, churches, sport arenas, cinema and TV studios, and theatrical stages.

In particular, we have been exploring the notion of **transforming the stage of a theater play into an interactive space**. Recent developments in image processing and speech recognition now permit that basic aspects of the live action performed on a stage to be recognized in real time by a computer system. Also, computer graphics and multimedia technology are achieving a state where live control of graphics and video on a stage screen is possible. These technological breakthroughs have opened the stage for artistic experiences involving computer-synthesized characters and environments that were virtually impossible less than half a decade ago.

In [128] we proposed the term *computer theater* to refer to live theatrical performances involving active use of computers on the stage and directly involved in the performance. Implicit in the concept is assumed that the performance happens in a physical space (not in a virtual stage), in a situation that transforms the stage into an interactive, computerized space.

Our work has focused on both the development of solid foundations for computer theater technology and on the exploration of the artistic possibilities enabled by inserting electronics and computers into the world of theater and performance art. Besides that, we have developed some theoretical work aiming to understand how computers can be used in theater.

This section develops a basic classification scheme for computer theater experiences and discusses their different aspects, artistic possibilities, and impact on performance arts. We also explore the relations between the research on action representation and recognition (such as the work presented in this thesis) for the development of computer theater projects. We finish by commenting on the possibility of creating *immersive stages*, that is, interactive spaces that can be used both by performers and audience, and by telling part of our story and involvement with computer theater.

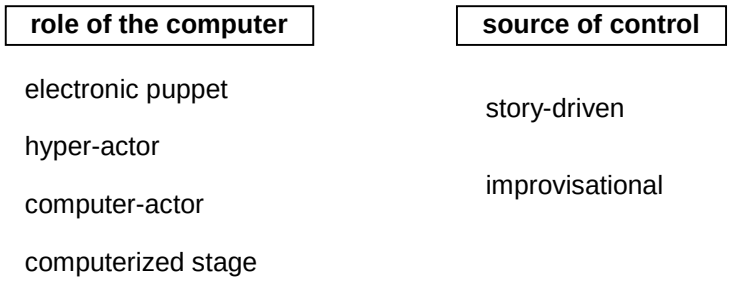


Figure 6.2 The two dimensions of the classification of computer theater systems.

6.3.1 A Classification for Computer Theater

One of the reasons for the term *computer theater* as used in this thesis (and previously in [131] and [129]) is to highlight the similarities to *computer music* and to the development of inter-relations between music and computers in the last three decades. In particular, we consider the classification of interactive computer music systems proposed by Rowe in [147] as the starting point for the understanding of the possibilities of computer theater.

Rowe [147] classifies music systems according to three dimensions, two of which naturally extend to theatrical systems. The first dimension of analysis is the differentiation between interactive music systems that take the role of a *player* from those which act as an *instrument* (to be controlled by a human player). The second dimension distinguishes between systems that are *score-driven* from those that are *performance-driven*.

Inspired on his classification, we propose the classification of computer theater systems according to two dimensions. The first dimension categorizes systems according to the role the computer plays in the performance: *electronic puppets*, *hyper-actors*, *computer-actors*, and *computerized stages*. The second dimension relates to the source of control, and distinguishes between *story-driven* and *improvisational* computer theater systems. Figure 6.2 displays a diagram that visualizes the two dimensions of the classification. The following paragraphs describe the different components of each dimension.

Electronic Puppets

Computers can be used to construct an *electronic puppet*, that is, a non-human body controlled by a human puppeteer who does not appear on the stage. In this situation, the computer has the role of **mediating between the puppeteer and the puppet**, for situations where only a computer can produce the "body" of the puppet. For instance, a puppeteer can control a computer graphics character displayed on a stage screen.

Examples of use of electronic puppets are more common in the context of *performance animation*, a technique used in computer graphics where 3D positional and attitude-sensors are

attached to the body an actor/puppeteer. Typical examples are the work of the company *Protozoa* who once wired the actor Symbad into the body of a CG-cockroach. The French performance group *D'Cuckoo* has included in its performances a character called *Rigby* (controlled off-stage by two puppeteers) that comments and dances with the music. Robots have also appeared on stage, as in the experimental dance piece *"Invisible Cities"* by Michael McNabb, Brenda Way, and Gayle Curtis, performed in Stanford in 1985.

In a short performance at the Sixth Biennial Symposium for Arts and Technology in Connecticut, in 1997, Flavia Sparacino [162] used animated, bouncing text lines projected in a background screen to illustrate the thoughts of the main character. The movement of the text was pre-choreographed and during the performance she cued the appearance of the text lines manually, making it into a simple form of puppet. A more interesting idea, bridging traditional puppetry and performance animation, can be seen in the work of Bruce Blumberg at the MIT Media Laboratory, where a sensor-loaded plush chicken is used to control the movements of a CG character in a virtual world [71]. Unfortunately, this system has not yet been used in performance.

Hyper-Actors

Rowe [147] classifies an interactive musical system as following the *instrument* paradigm if the basic objective is to construct an extended musical instrument. For instance, in the *hyperinstruments* project led by Tod Machover [88], musical instruments were built that sensed a virtuoso musician's gestures, enabling her to control and modulate a computerized counterpart to the acoustic sound produced by the instrument.

An actor's instrument is his body — including voice and facial expression. "Virtuosi" actors are able to control their bodies in extraordinary and different ways. For example, the actress Roberta Carreri of the Odin Theater is famous for her ability to control the movement of her long hair using a hand fan (see [13], pg. 111). Through the centuries actors have relied on masks, make-up, and costumes to alter their bodies

We suggest the term *hyper-actor* to denote a situation where the actor's body is enhanced by a computer. In this situation, the computer's role is **to expand the actor's body** through the use of electronic technology. A hyper-actor may be able, using body movements, to trigger lights, sounds, or images on a stage screen; to control his final appearance to the public if his image or voice is mediated through the computer; to expand its sensor capabilities by receiving information on earphones or video goggles; or to control physical devices like cameras, parts of the set, robots, or the theater machinery.

Another possibility is having the actor not on stage and providing him the means to control the physical appearance of his own image to the audience. Mark Reaney's *"The Adding Machine"* [142] is a curious illustration of this concept. In a typical scene an actor on stage plays with an off-stage actor whose image is seen by the audience on two large stereographic video screens (the audience wears special 3D-goggles). The off-stage actor's images on the screens expand and contract according to the play events and are used to symbolize and enrich the power struggle between the characters.

The idea of expanding the performer's body has been more explored in dance than in theater. Body suits wired with sensors having been widely explored in the pioneering work of Benoit Maubrey with his *audio-ballerinas* [100], and recently in the works of the New York based *Troika Ranch*. George Coates' experimented with actors receiving the script from Internet users during the live performance of *"Better Bad News"*. In Christopher Janney and Sara Rudner's piece "*Heartbeat:mb*", Michail Baryshnikov danced to the sound of his own heart.

Computer-Actors

The *player* paradigm in interactive music systems corresponds to situations where the intention is to build *"...an artificial player, a musical presence with a personality and behavior of its own..."* ([147], pg. 8). In the computer theater realm this paradigm corresponds to the idea of a *computer-actor*, a computer system that automatically interacts with human actors **in the role of one of the characters of the play**. In this case the computer character displays its actions using output devices such as video screens, monitors, speakers, or physical actuators.

We believe that our work in the performance *"SingSong"* and in the computer theater play *"It / I"* (described in the next chapter) pioneered the idea of having automatic computer controlled creatures interacting on a theatrical stage with a human actor. There a number of reasons for the absence of prior experiences with computer-actors: the lack of appropriate hardware for real-time interaction and CG-generation; the fact that real-time visual tracking and recognition of human motion was almost impossible until the middle of the 90s; and the lack of systems to script interaction, that is, to tell the computer-actor how the story must develop.

Sul et al. [168] built a *Virtual Stage* for a karaoke system where a user appears superimposed on a virtual space populated with computer-generated and controlled music players and singers. It is interesting that this is one of the few interactive systems that was built for performance situation. The variety of the behavior of the members of the virtual band is very limited, but the system explores the "script" of the performance, that is, the sequence of MIDI notes that correspond to the music being sing.

Most of the work related to computer-actors comes, in fact, from the research oriented towards user-driven interaction with computer-generated characters for game-like systems. Ken Perlin, working at New York University, has developed dancers and opera singers [125], and more recently a virtual performance, *"Sid and the Penguins"*, with autonomous CG-penguins that not only perform the story but also have basic notions of stage presence [126]. Particularly important is the work of Bruce Blumberg [16, 17, 94] in building a computer graphics generated dog, "*Silus*", that interacts with the user, not only obeying simple gestural commands ("sit", "catch the ball") but also having its own agenda of necessities (drinking, urinating).

A computer-actor must be able to follow the script (if there is one) and react according to its own role. Here, the issues of action recognition and automatic control of expressiveness seem to be more relevant than in the case of hyper-actors.

Computerized Stages

The last category of computer theater, in relation to the role played by the computer, is concerned with the expansion of the possibilities for the stage, set, props, costumes, light, and sound. In a *computerized stage*, the computer takes the role of controlling the physical space where the story develops. The fundamental distinction between computerized stages and hyper- and computer-actors is that in the former the computer does not play characters neither is involved in representing characters or their bodies.

A stage can react by changing illumination, generating visual and special sound effects, changing the appearance of backdrops and props, or controlling machinery. An example is the *Intelligent Stage* project at Arizona State University [86], that enables the mapping of volumes in the 3D space to MIDI outputs. Movement and presence are monitored by 3 cameras, triggering music and lights accordingly.

Kevin Atherton used a non-interactive computer generated stage in a very clever way in his performance *"Gallery Guide"*. In this piece, a human guide takes the audience through a computer-generated gallery where impossible artworks (generated in CG) are described with wit and humor. George Coates has also extensively used 3D scenarios in his long-term work involving theater and computers in San Francisco. Recently, the theater icon Bob Wilson ventured in this direction in *"Monsters of Grace"*, a joint piece with Phillip Glass. Oddly enough, although the final version featured only non-interactive, timed 3D imagery, many people seemed to prefer the "work-in-progress" versions where some of the scenes used live actors.

6.3.2 Scripts and Improvisations

The second dimension we propose for the classification of computer theater systems relates to the source of control information. This distinction is roughly equivalent to the second dimension of classification of computer music systems considered by Rowe that distinguishes between *score*- and *performance-driven* computer music [147]. In our classification, we map these concepts to *scripted* and *improvisational* computer theater.

Scripted computer theater systems are supposed to follow totally or partially the sequence of actions described in a script. During the performance the system synchronizes the on-going action with the script, giving back its "lines" as they were determined during the rehearsal process or by the director. *Improvisational* computer theater relies on well-defined characters and/or situations. This type of computer theater has immediate connections with the research on developing characters for computer games and software agents [14, 16], as we have already observed.

However, good improvisation requires recognition of intentions (see [73]). Knowing what the other character wants to do enables interesting and rich counteracting behavior. Otherwise, the resulting piece is flat, structurally resembling a "dialogue" with an animal: the sense of immediacy dictates most of the actions. We consider that building computer-actors able to improvise with human actors in a performance to be an extremely difficult goal that will require the solving of many AI issues related to contextual and common sense reasoning.

6.3.3 Computer Theater and Human Action Representation

It is certainly possible to have a computer theater system that just produces output following a timeline of pre-determined responses. Although human actors (and especially dancers) can adjust their performances to such situations, the results normally are devoid of richness and life. Computer theater seems to be worthwhile only if the computer system follows the actions of its human partners and reacts accordingly.

In the case of scripted theater the computer system must be able to recognize the actions being performed by the human actors and to match them with the information from the script. Minimally, the computer can use a list describing mappings between sensory inputs and the corresponding computer-generated outputs. The list can be provided manually by the "director" or technical assistants and, during performance, the recognition consists in synchronizing live action to the list according to the sensory mappings.

Although the "simple" system just described is hard to implement in practice due to noisy sensors and performance variability, we believe there is a much more interesting approach to computer theater based on **action representation and recognition**. Instead of providing a computer theater system a list of sensor-reaction cryptic mappings, the challenge is to use as input the actions and reactions as determined by the script or by the director.

As discussed by Suzanne Langer in [83], action is the basis of theater and, as such, we believe it needs to be fully incorporated in whatever model a computer is running during a computer-based theatrical performance. In fact, it is conceivable that the lack of good models for action has been one fundamental reason for the relative absence of experiments involving theater and computers. The relative ease of automatic translation of musical scores to a computational representation (MIDI, for example) seems to have played a major role in the development of computer music. Theater scripts capture many aspects of theater by describing the action and interaction among the characters but there is no method to translate characters' lines and stage directions written in natural language into something useful for a computer engaged in a performance. In our view, a fundamental goal is to use the textual description of actions in the script as input to a computer theater system. According to this view, such systems should be instructed by words like *"shout"* or *"whisper"* and be able to recognize automatically an action described simply as *"actor walks to the chair"*.

The concept of action is essential to the vitality of theatrical performance and must be incorporated, implicitly or explicitly, into any computer theater system. We do not foresee widespread use of computers in theater until a language paradigm is established which is expressive enough for computer manipulation and simple enough to be used by performers, directors, and play-writers.

6.3.4 Immersive Stages: Putting the Audience Inside the Play

We are particularly fascinated by the idea of using computerized environments and characters to make plays where a member of the audience is able to experience the feeling of being one of the characters from the play. This excitement drives us towards the development of interactive spaces that are able both to be part of a performance together with human actors and also to

control an interactive re-enacting of the play with members of the audience playing the main characters — what we call an *immersive stage*.

Techniques for character creation in theater often rely on extensive physicalization of activities and situations (for example, in Grotovsky's method [143, 185]). The movement of an actor's body tells her about feelings and emotions that can not be simulated by reasoning or imagination. Physically living a scene is usually a much richer experience than to imagine yourself in that situation, especially if the enactment develops continuously, that is, if the suspension of disbelief is kept. This is precisely the kind of experience we would like to give to the members of a theater audience: to get deeply involved in the story they have just watched, with mind and body.

The duality of the immersive stage, both a performance and a private space, contributes to add mystery and magic to the re-enactment of the play. Suddenly, the spectator is inside the very same space that until recently was occupied by the actor in a full-bodied incarnation of the character. We envision immersive stages as spaces that can be used by actors during the evening for performances and are open to the public for role-playing during the rest of the day.

To accomplish this kind of experiences, it is necessary to concentrate on fully automatic computer-actors "living" in a computerized stage. The experience between the user and the machine should unfold smoothly, taking in consideration what the user is accomplishing and how immersed she is estimated to be. Although there are certainly many challenges to realize this dream, we believe that carefully chosen stories matching the state-of-the-art of the sensing technology can create, today, immersive and complex worlds that can be explored by both actors and audience. The art installation *"It"*, described in the next chapter, is precisely an experiment on transforming the theater play *"It / I"* into an interactive space for users.

6.3.5 Making Theater with Computers

Our interest in computer theater started in the fall of 1994 when we considered the creation of an interactive space based on the play *"Waltz #6"* by Brazilian writer Nelson Rodrigues. The project would use the sensor technology employed in *ALIVE* [186] and would provide experiences both for performers and users through the construction of computer-actors for the characters imagined by *"Sonya"*, the main character. However, after some consideration, it became clear that the available hardware at that time would not be cope with the requirements of play.

A year later we considered again using the same technology in a computerized stage for the short theatrical piece *"e-Love"*, by Claudio Pinhanez, to be performed in Claire Mallardi's workshop at Radcliffe College. Unfortunately the computer part of the project ended up being abandoned due to logistic considerations. Immediately after, we start studying more carefully how to overcome the technological difficulties and the research potential of using computers in theater, leading to the proposal of the term computer theater and our first categorization of theatrical experiences with computers [128].

In the summer of 1996 at the ATR Laboratory in Kyoto, Japan, we conceived, developed, and performed the short computer theater piece *"SingSong"*. The primary intention of this piece

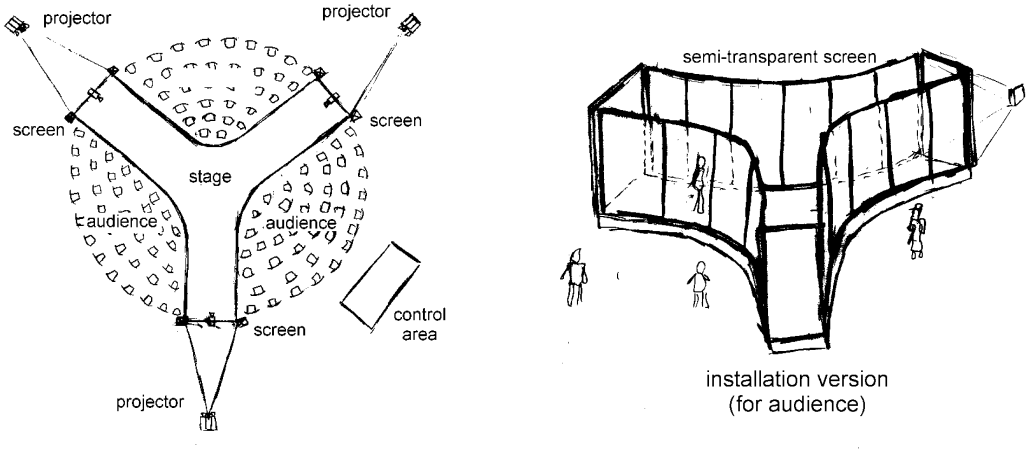


Figure 6.3 Two sketches for the *"Waltz #6"* project. The left figure shows a diagram of the stage space. The right figure shows the installation version (for audience) when the stage is enclosed into semi-transparent screens to increase the feeling of immersion.

was to experiment with the technological and aesthetic limitations of theater with computer-actors. In November of 1997 we premiered the play *"It / I"*, thematically examining the relation between people and technology through the creation of an environment dominated by a truly computerized creature. The play, 40 minutes long, was performed 6 times at the Villers Facility of the MIT Media Laboratory, for a total audience of about 500 people. Both projects are detailed in the next chapter.

In 1998 we reconceived the *"Waltz #6"* project, taking in consideration the increasing power of hardware and the technology and the experience acquired in *"SingSong"* and *"It / I"*. Figure 6.3 shows sketches of the project that includes a "Y"-shaped stage with three screens and employs both vision and speech technology. *"Waltz #6"* tries to fully realize the concept of immersive stage by a story that offers multiple narrative paths and by enclosing the user in semi-transparent screens that hide the user from the other viewers (see fig. 6.3). The project was submitted to the *Cyberstar'98* international competition in Germany and was selected among the finalists. We are currently working on obtaining funding for the realization of the project in the summer of 2000.

6.4 Summary

This chapter presented two novel concepts in the realm of story-driven interactive spaces. The first contribution of this chapter is the *story-character-device architecture* for interactive, story-based systems. Based on the observation that story requires centralized control, we argue that interactive systems with narrative structure should separate the control of characters and story, viewing the characters as actors. That is, a story-based system requires not creatures but actors that are able to follow high-level directives as needed to unfold the story while keeping the consistency and behavior specific of their characters. The *SCD* architecture was partially

realized in the computer theater play *"It / I"* and fully implemented in the art installation *"It"*, as described in the next chapter.

This chapter also discussed the use of computers in theater, proposing basic classification mechanisms and arguing that computer theater technology is intimately related to the issue of representing and recognizing human action. In particular, we have been focusing in the development of automatic *computer-actors* and *computerized stages*, aiming to realize the idea of an *immersive stage*, that is an interactive space that can be alternatively used for performance and personal use. Both the theory of computer theater as described in this chapter and the computer theater projects described in the next seem to have decisively raised the level of public and artistic awareness of the possible future roles of computers in theater and performance.

7. Building Interactive Spaces

In this chapter we present four different interactive spaces we built using the elements described in the previous chapters. We have two goals for this presentation: first, the description of each project demonstrates the effectiveness of the paradigms and algorithms described in previous chapters in large, complex, real-life applications. Second, we believe that in the process of building these interactive spaces we have developed interesting ideas concerning the architecture and the technology of such spaces.

The first project — the *Intelligent Studio* — is a system to automatically control cameras in a TV studio. The second and third projects are *computer theater* experiments, that is, theatrical performances employing computer systems: the performance *"SingSong"* and the play *"It / I"*. The last project is an interactive art installation, *"It"*, based on the play *"It / I"*.

Four common elements permeate the four different projects. First, we have only worked in real **physical spaces** inhabited by real people. Although many results of our research apply to virtual worlds and avatars, we found it important to ground our experiments — in particular, when action recognition was involved — in noisy, real-world data. Second, the basic sensing elements in all four projects are **video cameras**. Third, not surprisingly, the four projects involve **people performing actions**. And fourth, the interaction in the developed projects has always a narrative structure, available in a **script** that is represented in the computer.

We start in section 7.1 by describing the *Intelligent Studio* project, the basic testbed for the research on representation and reasoning on human actions described in chapter 2. Our first experiment with the idea of computer theater and immersive stages was the performance *"SingSong"* presented in section 1.1. Besides probably being the first time that autonomous computer characters were used in a theatrical performance, *"SingSong"* also started our work on interval scripts.

Section 1.1 examines our most ambitious project, the computer theater play *"It / I"*. The play was fully produced and publicly performed in November of 1997, and presented a human actor and an automatic computer-actor in a complex interaction that lasted for about 40 minutes. We discuss both the artistic and the technical aspects of the project which employed most of the

recognition and scripting methods described in this thesis. In fact, *"It / I"* was the major testbed for these methods.

The last project, the art installation *"It"*, is a study about the issues surrounding the creation of an immersive stage. In particular, we examine how to adapt performance material (in this case, the play *"It / I"*) into an interactive installation for users. *"It"* was also the testbed for the final versions of ACTSCRIPT, of the PNF-based action recognition method, of the *interval script* paradigm, and the *story-character-device architecture* described in the previous chapters. In fact, most of the examples presented in the previous chapters come from *"It"*.

We opted for structuring this chapter as a narrative describing our experiences, and how some ideas evolved through time. In particular, we use this approach when we describe the process of scripting the interactive spaces and how the concept and implementation of interval scripts has been progressively refined. For instance, in the description of scripting issues in *"SingSong"* we employ the original syntax and semantics of interval scripts as it was used in the development of the space. Although this may somewhat increase the difficulty of reading the thesis, we find it important to describe our results in their original setting.

Moreover, by examining how the paradigms and methods have evolved through time, we hope that we will make clear the reasons why particular solutions have been adopted and also why alternative options were discarded. For the same reason we include in this section comments about the success and failure of particular technologies used in different parts of our systems, from communication packages to computer graphics features. We hope that this collection of decisions, results, mistakes, and small ideas is insightful and useful to other people involved in the building of interactive spaces.

7.1 The *Intelligent Studio* Project

The goal of the *Intelligent Studio* project was to create a TV studio that could automatically control robotic cameras to produce the images for a TV show. In our concept, the *Intelligent Studio* is monitored by fixed, wide-angle cameras watching the basic objects and actions. Together with the script, the cameras are able to maintain a model of the objects and events in the studio. The model is used by automatic, high-quality, mobile cameras — *SmartCams* — responsible for the generation of the televised images under the command of a human TV director (see fig. 7.1).

The *Intelligent Studio* concept does not fit perfectly in our definition of interactive spaces as stated in the previous chapter. Although the computerized system monitors the actions in the studio, it interacts with the TV director who is outside of it. In spite of this, the basic structure is similar to the normal structure of an interactive space where the subject of interaction is inside in the space. Moreover, as it will become clear from the description, the control system faces similar tasks and conditions of any interactive space: it has to understand human activity through sensing and to respond to user commands within the context of the physical activity.

From a systems point of view, the *Intelligent Studio* project featured two main ideas: *approximate world models* as a repository of information for vision systems and the use of linguistic information from the script as source of information for the control module. In

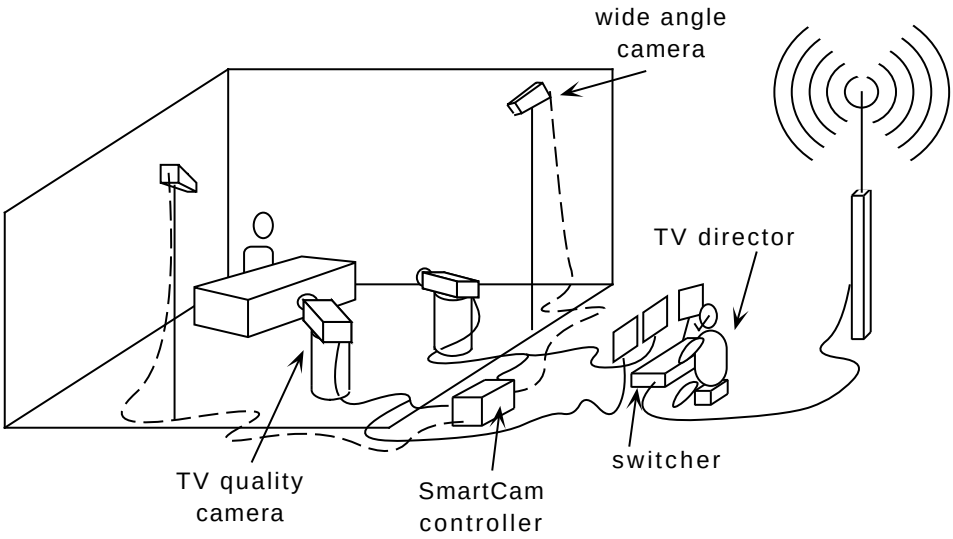


Figure 7.1 The concept of an Intelligent Studio which uses wide-angle imagery to understand the events happening in the studio.

chapter 2 we discussed how scripts can be represented and how they can be augmented by simple common sense reasoning inference procedures. Here, we will describe how that information is actually used by the vision system of a *SmartCam*.

The main component of our approach is to model the different elements of a TV studio with relaxed accuracy requirements and thereby gain the use of qualitative information from the script. The result is an *approximate model* that can provide basic information about position, attitude, movement, and action of the people and objects in the studio (see [23] for more details about approximate models).

The approximate model of the studio is used to control the automatic cameras. Basically, the vision system of each *SmartCam* uses the approximate model to select, initialize, and control task-specific vision routines that are able to produce the accuracy required for framing. The existence of the approximate model considerably reduces the complexity of the required vision routines [23].

On the other hand, the approximate nature of the approximate world model facilitates the use of linguistic information about the scene, available in our case from the script of the show. It is important to note that without the constraints imposed by the script, most of the computer vision techniques would be likely to fail in a complex environment like a TV studio.

A “cooking show” is the first domain in which we have experimented with our *SmartCams*. Throughout this section, examples drawn from the cooking show domain are used to illustrate some concepts, although most of the ideas apply to the general problem of modeling an interactive space. These ideas first appeared in [19], where the system partially worked without

accessing information from the script. In [130], we presented for the first time the full system, which is further detailed in [133].

The system described in this thesis does not use real moving cameras, but simulates them using a moving window on wide-angle images of the set. Several performances of a 5-minute scene as viewed by three wide-angle cameras were recorded and digitized. The *SmartCam* output image is generated by extracting a rectangular window of some size from the wide-angle images. The calling of the shots is truly responsive: the TV director watches the show develop and issues calls for the different cameras. The response is based only on the past information (no look-ahead to the future) and to which shot is being called.

7.1.1 The *Intelligent Studio* Concept

"Camera 3, ready for close-up of the chef. — More head-room. — Take camera 3. — Camera 1, ready for close-up of the bowl. — Take camera 1. — Follow the hands." This is how a TV director usually communicates with his cameramen in a TV studio. The TV director asks each camera for specific shots of the scene, or *calls*, and the *"take"* command signals that the camera is entering on air.

After receiving a call, the cameraman looks for the appropriate subject, adjusts the framing, and waits, keeping the best possible framing. After the shot has been used, the cameraman receives a new call. This is the standard procedure for most TV programs including news, talk shows, sitcoms, and cooking shows. It is important to notice that such procedures are quite different from those used in movies or more elaborate TV programs where the images from each camera direction are shot separately and later assembled during editing.

Framing is a quite complex task, requiring different kinds of information. For instance, a call for a close-up demands not only the information of the subject head's position and size but also the subject's direction of sight and the position of the eyes. Knowledge of the direction of sight is required because profiles are always framed leaving some space in front of the face (called "nose room"). The height of the eyes is used, for instance, in a rule of thumb that states that eyes in a close-up should be leveled at two thirds of the height of the screen (see [188], pp.111-122, for this and other simple rules).

Moreover, framing changes according to the current action of the subjects. If an object is being manipulated, either it is fully included or it is put outside of the frame. Also, subjects in the background must be either framed completely or removed from the picture. The information required in these cases involves considerable understanding of the actions happening on the set.

However, each camera's information about the world is constrained by its current framing. A camera providing a close-up is unable to detect changes outside the image area, significantly reducing its ability to understand activity. A simple solution to this problem is to use extra cameras in the studio whose only purpose is to monitor the scene. The resulting system, which we call an *Intelligent Studio*, is thus composed of coarse, fixed, wide-angle cameras watching the basic objects and actions and automatic, high-quality, mobile cameras blind to anything not inside their field of view. These high-quality cameras are the ultimate responsible for the generation of the show's images (see fig. 7.1).

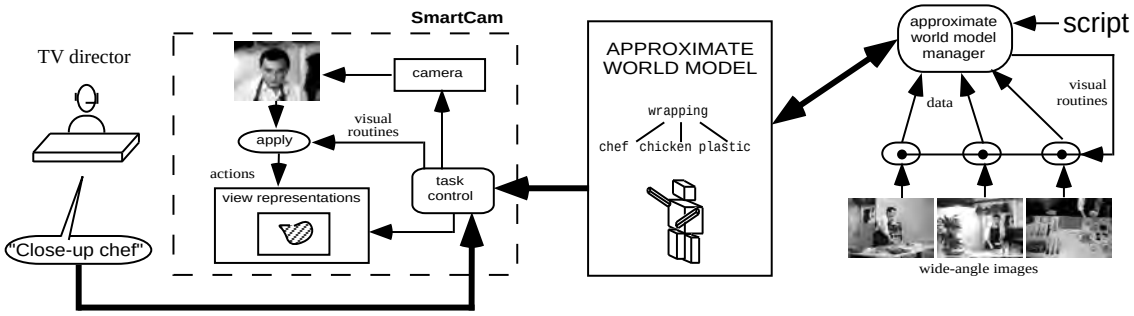


Figure 7.2 The architecture of a *SmartCam.* The bottom part of the figure shows the structure of the modules responsible for maintaining the approximate world models.

7.1.2 *SmartCams*

One of the objectives of having an "intelligent" interactive space is to have it react to the occurrence of actions by controlling physical devices that act in the space. In the case of the TV studio, we have focused in the task of building robotic cameras that are able to produce images acceptable for TV viewers. The task requires not only sophisticated image processing but considerable understanding of the actions happening in the studio.

Moreover, we want our robotic camera to be compatible to the normal structure of TV production. That means that the camera should not be responsible for the decision of what to frame and when. As in a normal studio, determining what is to be framed is the function of the TV director, the person ultimately responsible for the show. Our *SmartCam* is thus conceived as a robotic camera for TV that can operate without a cameraman, changing its attitude, zoom, and position to provide specific images upon the verbal request of the director.

The basic architecture of a *SmartCam* is shown in fig. 7.2. The main component is the *approximate world model* [23] that represents the subjects and objects in the scene geometrically as 3-D blocks, cylinders, and ellipsoids, and symbolically with frame-based representations (slots and keywords). The symbolic description of an object includes information about to which class of objects the object belongs, its potential use, and its roles in the current actions. The 3-D representations are positioned in a 3-D virtual space corresponding to the TV studio. The cameras' calibration parameters are also approximately known. The precision can be quite low: in our system the position of an object might be off by an amount comparable to its size.

For example, if there is a bowl present in the studio, its approximate world model is composed of a 3-D geometric model of the bowl and a frame-like symbolic description. The 3-D geometric model approximates the shape of the bowl and is positioned in the virtual space according to the available information. The objects in the approximate world model belong to different categories. For example, a bowl is a member of the "*handleable objects*" category. As such, its frame includes slots that describe whether a human is handling the bowl and, in this case, there is also a slot that explicitly points to him/her.

7.1.3 The Approximate World Model in The *SmartCam*

In our implementation of the *SmartCams* the 3D models of the subjects and objects were initialized and positioned manually in the first frame of each sequence. All changes to the models after the first frame are accomplished automatically using vision and by processing the linguistic information.

To track small changes in the approximate world model and especially in its 3D representations, the *Intelligent Studio* employs vision tracking routines capable of detecting movements of the main components of the scene, i.e., the chef and his hands. As shown in the right side of the diagram in fig. 7.2 this is accomplished directly by processing the wide-angle images of the scene, simulating the wide-angle, monitoring cameras described in fig. 7.1, and it is completely independent of whatever is occurring inside each *SmartCam*. The two-dimensional motions of an object detected by the different cameras are integrated to determine the movement of the object in the 3D world. The calibration parameters of the cameras are approximately known and, although as imprecise as the positional information in the approximate world model, they proved to be enough for the approximate 3D reconstruction of the scene.

To keep track of major changes in the scene, the approximate world manager also uses the information contained in the script of the show, which is represented using *action frames*. For this, the approximate world manager uses the inference system described in chapter 2 to extract visual-related information.

7.1.4 Results for Two Different Shows

The final version of our *SmartCams* handled three types of framing (*close-ups, medium close shots, medium shots*) for a scenario consisting of one or two chefs and about ten objects. All the results obtained employed only very simple vision routines based on movement detection by frame subtraction. The position of the objects was given in the first frame of the sequence. The system also employed a simple, dialogue-free version of the TV script composed of simple statements describing 10 different actions written in the action frames syntax (as explained in chapter 2).

The Influence of the Approximate World Models

In fig. 7.3 we can see that the inaccuracy of the approximate world models does not affect the final results of the vision routines. Two SmartCams, side and center, were tasked to provide a close-up of the chef. The geometric model corresponding to the chef is quite misaligned, as can be seen by its projection into the wide-angle images of the scene (left side), and in the images seen by the SmartCams. In those pictures, the projections of the approximate geometric models of the head and the trunk are displayed as rectangles.

However, the projection of the approximate models provides a good starting point for motion-based perceptual routines that ultimately succeed in finding the target of the shot. The results of the perceptual routines are shown as highlighted areas in the right side of fig. 7.3.

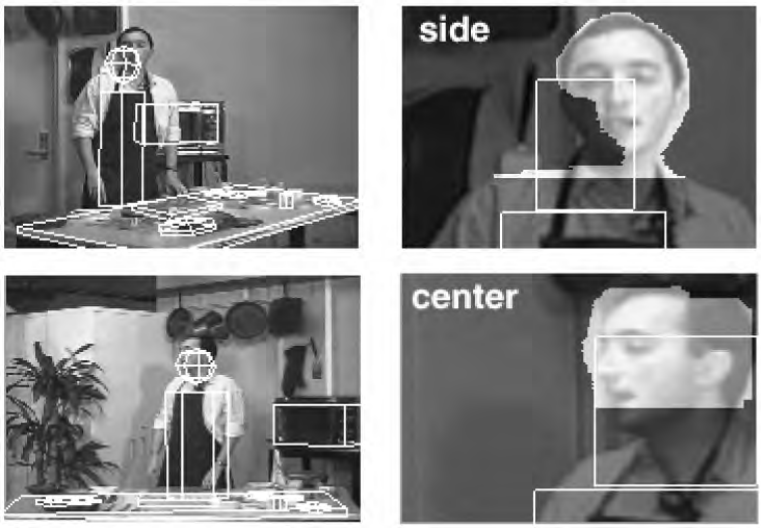


Figure 7.3 Example of response to the call *"close-up chef"* by two different cameras, *side* and *center* (see text).

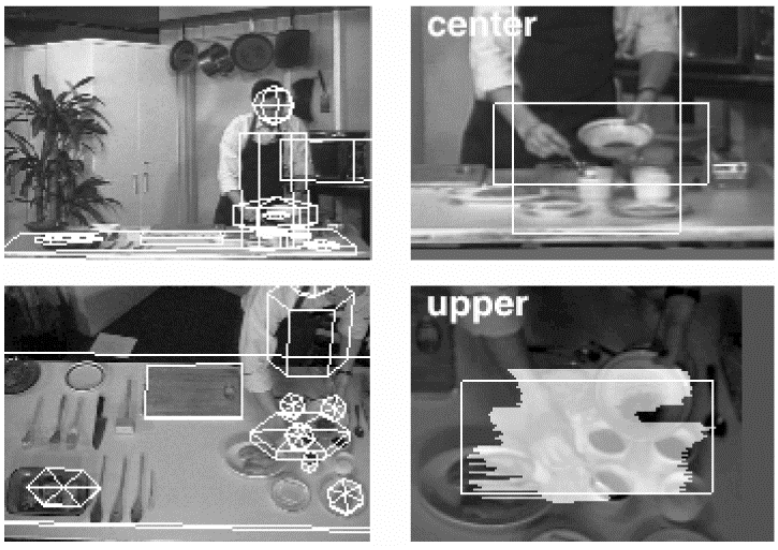


Figure 7.4 Example of response to the call *"close-up hands"* by two different cameras, *center* and *upper* (see text).

Figure 7.4 shows a situation where the approximate world models avoid the application of a routine in an unsafe condition. In this case, both cameras were asked to provide a close-up of the hands. In this situation the hands are approximately modeled by an ellipsoid in front of the trunk. This model comes from the information contained in the script indicating that a *"mixing ingredients"* action is happening at the current frame. Based solely on knowing the current action, it is possible to predict that the hands are likely to be in the front on the trunk and at the height of the waist.

In these conditions a routine that detects moving blobs can be applied to the upper camera images, resulting in the highlighted area on the bottom-right of fig. 7.4. However, in the case of the center camera, the predicted region for the hands is in front of the region of the trunk (see top-right image of fig. 7.4). Therefore it is not safe to apply a simple routine that extracts moving blobs, because there is a risk of detecting the movement of a background object (the trunk) instead of the movement of the hands. If all other applicability rules also fail, the system knows that it can not robustly determine the position of the hands from that particular viewpoint.

The Vismod Cooking Show

Our initial set of data came from staging a cooking show in the laboratory space. A set resembling a typical TV cooking show was built and three cameras were positioned in standard center/side/up positions. The selected recipe involved mixing breadcrumbs and spices in a bowl, flattening a piece of chicken with a meat-mallet, and rolling it over the crumbs. The whole action lasted around 8 minutes, of which 5 minutes were digitized. Two complete sequences were recorded and digitized, correspond to two different performances of the same script. In the second sequence the chef was wearing glasses and the actions were performed in a faster speed.

Figure 7.5 shows typical framing results obtained by the system. The leftmost column of fig. 7.5 displays some frames generated in response to the call *"close-up chef"*. The center column of fig. 7.5 contains another sequence of frames, showing the images provided by the *SmartCam* tasked to provide *"close-up hands"*. The rightmost column of fig. 7.5 is the response to a call for a *"close-up hands"*. In this situation, the action *"chef pounds the chicken with a meat-mallet"* is happening. This action determines, through the inference system described in chapter 2, that the hands must be close to the chopping board. This information is used by the system to initialize expectations for the hands in the beginning of the action (both in terms of position and movement), enabling the tracking system to detect the hands' position based solely on movement information.

Figure 7.5 Responses to the calls *"close-up chef"*, *"close-up hands"*, and *"close-up hands".* Refer to background objects to verify the amount of correction needed to answer those calls appropriately. The gray areas to the right of the last frames of the first *"close-up hands"* sequence correspond to areas outside of the field of view of the wide-angle image sequence used by the system.

Table 7.1 Example of sequence of calls during the *"Vismod Cooking Show".*

time	camera	type of shot	target	duration
0.0	center	medium-shot	chef	13.0
13.0	side	close-up	chef	6.0
19.0	center	close-up	ready-dish	2.0
21.0	side	close-up	chef	5.8
26.8	center	medium-shot	chef	5.2
32.0	upper	close-up	hands	3.0
35.0	center	medium-shot	chef	3.0
38.0	upper	close-up	hands	9.0
47.0	center	medium-close-shot	chef	3.0
50.0	upper	close-up	bowl	3.0
53.0	center	medium-shot	chef	11.0
64.0	side	medium-close-shot	chef	5.0
69.0	upper	close-up	chopping-board	7.0
76.0	side	close-up	hands	4.0

Table 7.1 shows the sequence of calls for different cameras during one of the runs of the system. The three short segments displayed in fig. 7.5 correspond to the calls with gray background in table 7.1. The whole sequence lasts about 80 seconds and comprises 14 different shots.

Table 7.2 Distribution of calls in the example sequence of the *"Vismod Cooking Show".*

	camera called			type of shot			target of shot		
	center	side	upper	close-up	med-close	medium	chef	hands	objects
number	6	4	4	8	4	2	8	3	3
total shot length	37.2	20.8	22.0	39.8	32.2	8.0	52.0	16.0	12.0
% of sequence	47%	26%	28%	50%	40%	10%	65%	20%	15%
avg. duration	6.2	5.2	5.5	5.0	8.1	4.0	6.5	5.3	4.0

Table 7.2 shows the distribution of the 14 calls according to three different criteria. First, we can see the *center* camera is active about half of the sequence, which is typical of TV shows. More interestingly, we have selected close-up shots about 50% of the time. Since close-ups require much more camera control and precise tracking than medium-close-shots or medium-shots, we can say that, in this example, half of the time the *SmartCams* were working under the most difficult conditions. Also notice that most of the time the cameras were framing moving objects: either the chef (65%), the hands of the chef (20%), or the bowl (4%), a total of 89% of the time of the sequence.

The results for the second sequence, portraying a different performance of the same script, were quite similar, with the exception of two brief intervals where the cameras did not frame correctly due to poor tracking.

By observing the generated image sequences, it can be clearly seen that the framing and the camera movements are as good as those in a standard cooking show. The two sequences demonstrate that acceptable results can be obtained by our *SmartCams* despite the simplicity of the vision routines employed.

The Alan Alda Cooking Show

In June of 1996 the *SmartCam* system was invited to appear in a special program of the TV series *"Scientific American Frontiers"*, hosted by the actor Alan Alda. We took the opportunity to record a second set of data in a more complex cooking show involving two chefs in the same kitchen. The recipe was selected among Alan Alda's favorites, an artichoke pasta sauce based on a recipe from his friend chef Giuliano.

We employed the same basic three-camera setup for the *"Alan Alda Cooking Show"*. Two sequences of about 10 minutes were recorded and we selected the second half of the first sequence to be digitized. In this sequence, Alan Alda demonstrates how to clean and cut the artichokes and explains how to assemble all the ingredients (artichokes, garlic, parsley, and basil) in a frying pan. The other chef discusses the making of the dish and occasionally helps with the manipulation of the objects.

In this situation, the system had to successfully track two persons who often occluded each other and to react to more complex actions. The approximate models simplified considerably the handling of occlusion by avoiding the call of motion-based vision routines in occluded views. Figure 7.6 shows some images from the *"Alan Alda Cooking Show"*. In the instant depicted by the images, one of the chefs is partially occluding the other. In spite of this, the two *SmartCams* successfully respond the calls for "*medium-close-shot alan"* and "*close-up claudio"*.

The script of the *"Alan Alda Cooking Show"* contains 86 descriptions of actions. This second experiment was performed some time after the first and we were pleased to verify that the inference system required only minor adjustments and extensions to successfully extract visual information from the show's script. This was particularly satisfying since the two cooking shows depict quite different actions.

Using the interactive interface, we were able to act as the TV director and to produce a high quality video sequence where the cameras successfully answered all the 38 commands in spite of occlusions and a complex set of actions. Table 7.3 lists the calls used in the sequence and demonstrates the variety of responses allowed by the system. The complete sequence lasted for 160 seconds, with one shot lasting an average of 4 seconds. The calls corresponding to the images of fig. 7.6 are marked with a gray background in table 7.3.

Figure 7.6 Images from the *Alan Alda Cooking Show,* showing on the top part the wide-angle images and on the bottom part the framing provided by two different *SmartCams* responding to the calls for a medium-close shot of Alan and a close-up of Claudio.

Table 7.4 summarizes how the shots were distributed according to different criteria. In this example, the side camera is on air half of the time since this is the camera used by Alan Alda to address the audience. In comparison with the previous example there are fewer close-up shots, since with two chefs there is a need for shots where they can be seen talking to each other. This is also reflected in the amount of time devoted to framing people (72%), and hands (14%), for a total of 86% of the time tracking moving targets. This data clearly demonstrates that our *Intelligent Studio* can handle different situations with varying degrees of complexity with only minor adjustments.

In both the *"Vismod"* and in the *"Alan Alda"* cooking shows we had to manually enter when each action was happening. During that time it became obvious that the next step in the development of the system would be to incorporate mechanisms for automatic detection of the actions in the script. Although our initial work in action recognition was based on the *Intelligent Studio* domain and data, when we finished the work on the *"Alan Alda"* sequence we started to look for interactive spaces where the result of the recognition would have a more directly impact on the interaction. At that time we began to realize that theatrical plays with automatic, computerized characters on stage would be an interesting domain.

Table 7.3 Sequence of calls during the *"Alan Alda Cooking Show"*.

time	camera	type of shot	target	duration
0.0	center	medium-shot	all	6.0
6.0	upper	close-up	chopping-board	2.2
10.2	side	close-up	claudio	2.8
13.0	center	medium-close-shot	alan	5.0
18.0	side	medium-shot	all	6.0
24.0	upper	close-up	alan-hands	6.0
30.0	center	medium-close-shot	alan	3.0
33.0	side	close-up	claudio	5.0
36.0	center	medium-close-shot	alan	4.0
40.0	upper	close-up	alan-hands	5.0
45.0	center	medium-shot	all	7.0
52.0	side	close-up	alan-hands	6.0
58.0	center	medium-close-shot	all	2.0
60.0	upper	close-up	chopping-board	4.0
64.0	center	medium-close-shot	all	5.0
69.0	side	close-up	claudio	2.0
71.0	center	medium-close-shot	alan	3.0
74.0	upper	close-up	juice-bowl	1.8
75.8	side	medium-shot	all	5.2
81.0	upper	close-up	alan-hands	4.0
85.0	side	close-up	alan	5.6
90.6	center	close-up	claudio	2.4
93.0	side	medium-close-shot	alan	9.8
102.8	center	medium-shot	all	2.0
104.8	side	medium-shot	alan	2.2
107.0	upper	close-up	frying-pan-1	4.0
111.0	center	medium-shot	all	4.0
115.0	side	medium-shot	alan	10.0
125.0	center	medium-shot	all	2.6
127.6	side	medium-close-shot	alan	2.2
129.8	upper	close-up	frying-pan-2	2.6
132.4	side	medium-shot	alan	4.0
136.4	upper	close-up	frying-pan-2	3.0
139.4	side	medium-close-shot	alan	4.6
144.0	upper	close-up	alan-hands	2.0
146.0	side	medium-shot	all	3.0
149.0	upper	close-up	pasta-dish	4.4
153.4	side	medium-close-shot	alan	6.6

Table 7.4 Distribution of calls in the example sequence of the *Alan Alda Cooking Show*.

	camera called			type of shot			target of shot		
	center	side	upper	close-up	med-close	medium	people	hands	objects
number	12	15	11	17	10	11	26	5	7
total shot length	46.0	75.0	39.0	62.8	45.2	52.0	115.0	23.0	22.0
% of sequence	29%	47%	24%	39%	28%	33%	72%	14%	14%
avg. duration	3.8	5.0	3.5	3.7	4.5	4.7	4.4	4.6	3.1

7.2 "*SingSong*"

"SingSong" is a short theatrical play produced in the summer of 1996 at ATR Laboratories in Kyoto, Japan. The goal of the project was to experiment with interactive spaces involving human performers interacting with automated computer-actors. The piece is a comical skit portraying a clown who is trying to conduct a chorus of computer-controlled (but not-so-well-behaved) singers.

The objective — more than just to produce a theatrical play — was to construct an interactive space where the dramatic concept of *action* was embodied in the computer. Within this framework, *"SingSong"* unfolds more as a result of the computer and the human actors acting upon each other than as a consequence of gestures or direct commands.

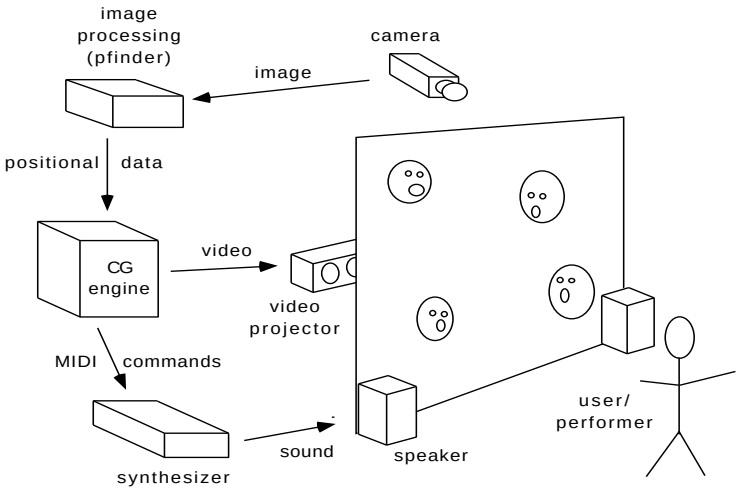


Figure 7.7 The basic setup of *"SingSong".*

Figure 7.7 shows the basic structure of *"SingSong"*: a large video screen with four computer graphics-animated characters which can "sing" musical notes (produced by the synthesizer). There is a CG-object — a tuning fork — that the user employs during one of the scenes. Sounds of applause can also be generated by the system. A computer vision system processes the input from a video camera, determining the position of the performer's head, hands, and feet. Both the computer system and the performer (or user) know the story of the play. The computer has a model of the story represented as an *interval script*, using the first implementation of the concept. The differences between this version of interval scripts and the version described in chapter 5 are discussed later.

7.2.1 The Story

The entire interaction of *"SingSong"* is nonverbal: the user or performer moves and mimes actions and the CG-characters sing notes, move, and make faces. *"SingSong"* is an interactive space that immerses the user or performer in a simple story.

> ***"SingSong"***
>
> *a skit for a clown and computer creatures*
>
> *by Claudio Pinhanez*
>
> *Singers of a chorus (the CG-creatures) are animatedly talking to each other. The conductor enters and commands them to stop by raising his arms. One of the singers — #1 — keeps talking until the conductor asks it to stop again. Singer #1 stops but complains (by expanding and making grudging sounds). The tuning fork appears and the conductor starts to tune the chorus: he points to a singer and "hits" the tuning fork by moving his arm down. Any singer can be tuned at any time. However, singer #1 does not get tuned: it keeps giving the conductor a wrong note until the conductor kneels down and pleads for its cooperation. After all the singers are tuned, a song is performed. The conductor controls only the tempo: the notes are played as he moves his arms up. When the song is finished, applause is heard, and when the conductor bows back the singers bow together with him. Just after that, singer #1 teases the conductor again and the singers go back to talking to each other.*

"SingSong" was conceived as a tribute to the early days of movies which employed comedy and pantomime as a creative solution to cope with the absence of sound and with undeveloped editing techniques. In our view, we are in a comparable state of knowledge and technological apparatus for designing and building interactive, immersive spaces and systems.

7.2.2 The Technology

"SingSong" was produced at a time when the necessary hardware for an interactive space started to become available. In our case, the system was comprised of a "cheap" machine for real-time video processing (an SGI Indy) and a "not-so-cheap" real-time computer graphics engine (an SGI Reality Engine). It is interesting to note that just two or three years before this, it would have been extremely difficult to find hardware with these capabilities. Coupled with a large-scale video projector and a Yamaha MIDI-synthesizer (controlled by the SGI Indy), it was possible to build the computer theater play in about six weeks.

At the same time, basic software for computer vision was also available. We employed the *pfinder* [186] program that extracts the silhouette of the performer by background subtraction. Like any vision system based on background subtraction, it must be calibrated with the room completely empty. The program computes the position of the hands and the head of the performer based on the curvature of the silhouette and on local color histograms. In the closed

context of *"SingSong"*, this tracking information could be safely translated into actions like pointing, requests to stop, and conducting.

The CG-creatures were created using the software *SoftImage* and the resulting files were converted into *SGI Inventor* format. During run-time, the animation is generated based on the commands coming from the interval script engine which are translated into matrix transformations. Sound is generated as the result of commands directly issued by the interval script which are translated into MIDI signals through the serial port of the SGI Indy. The script issues individual commands for each note of each singer.

To communicate between processes running in different machines, we employed the *RPC* protocol. Although efficient, this mechanism has some inconveniences: it is necessary to explicitly name the hosts of the processes; also, the binding among processes is based on the processes' *pid*, requiring re-starting all client processes whenever a server dies (making the debugging process quite cumbersome).

At the time of its production, interactive spaces technologically similar to *"SingSong"* had already been built [79, 94, 161, 172]. The technical innovation present in this interactive space was its control structure based in interval scripts. The real conceptual innovation was the idea of computer theater and the centrality of action as the chief carrier of the interaction.

7.2.3 The Scripting Process

The paradigm of interval scripts was conceived to satisfy the scripting needs of *"SingSong"*. Although *"SingSong"* portrays a very simple, quasi-linear story, there is a large number of parallel actions and conditional interaction during the performance.

Interval Scripts: the *"SingSong"* Version

"SingSong" used the first version of the concept of interval scripts, described in detail in [137]. The *"SingSong"* version of interval scripts already included the important idea of separating the desired action from the actual action. However, in that version, this had been implemented using two implicitly defined intervals instead the simpler use of START, STOP, and STATE functions. The use of two intervals required a more complicated model of the situation, making scripting more difficult than in the subsequent versions.

To ease these problems, the *"SingSong"* version of interval scripts pre-defines three categories of intervals (referred to as *externa*ls): *actuators*, *sensors*, and *timers*. As shown in fig. 7.8, an *actuator* automatically defines a *desired* interval, corresponding to the interval in which the action is supposed to happen and an *actual* interval, that is, the actual occurrence of the action. A *timer* is a particular case of an actuator that automatically goes to *P* after the schedule time expires. A *sensor* has an *activity* interval describing when the sensing is being performed and an *event* interval corresponding to the actual occurrence of the sensed event (refer to fig. 7.8).

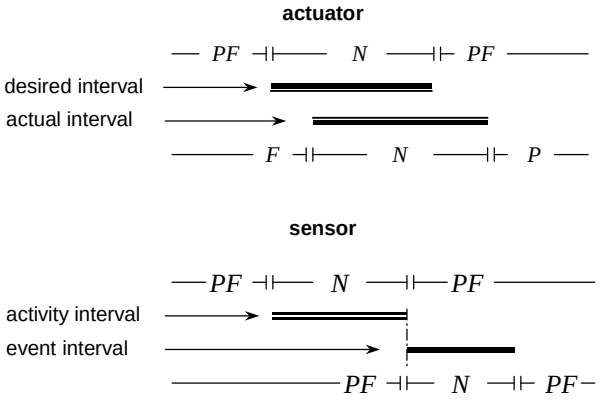


Figure 7.8 Intervals associated with an actuator and a sensor.

With these primitive categories, basic functionality to control the interaction is provided. For example, when a *desired* interval becomes *N*, the system tries to start the action corresponding to the interval. In the final version of the interval scripts, the *desired* interval corresponds to the START and STOP functions and the *actual* interval to the STATE function. The idea of this simplification came only later and it was first implemented in the version of interval scripts used in *"It / I"*.

Other basic concepts already were present in this version, including here the idea of "forgetting" an interval (referred as *resetting* the interval). However, the *"SingSong"* version of interval scripts did not provide any scripting language but instead a group of C++ classes that were used to define functions corresponding to each interval in the script. Compiling interval script-like language into C++ code was implemented for the first time in the *"It / I"* version.

Script Example from *"SingSong"*

Let us examine an example from the interval script of *"SingSong"*. In spite of the differences in the script syntax and in the structure of an interval script it is interesting to see that the basic method of scripting is very similar to the one discussed in chapter 5. This is not the most complex scene in *"SingSong"* but it provides an idea of the control issues in the scripting process.

In the first scene of *"SingSong"*, the four singers are chatting until the conductor commands them to stop by raising his arms. *Singer #1* acts differently from the other singers: it does not obey the conductor promptly and, after being commanded to stop, it complains. After the singers stop chatting, they start to stare at the conductor, following him around the space. The following are the externals used to define this scene:

- *Chatting* (actuator): 4 copies, one for each singer, controls the sound and the mouth movements that simulate chatting;
- *BeQuiet* (sensor): fires if the user raises both arms above his head;

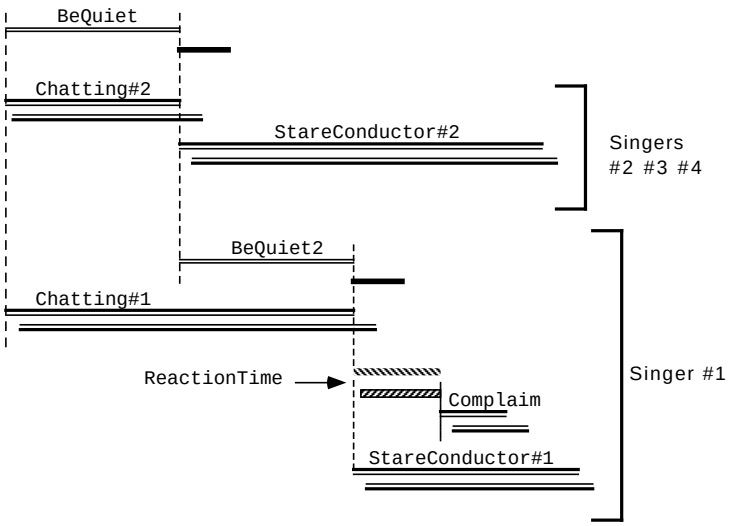


Figure 7.9 Diagram of the temporal relations in the first scene of *"SingSong".*

- BeQuiet2 (sensor): identical to *BeQuiet*;
- *StareConductor* (actuator): 4 copies, implements the behavior that makes the eyes of the creatures follow the conductor around the space;
- *ReactionTime* (timer): provides a pause of 3 seconds between the conductor's gesture and the complaint from singer #1;
- *Complain* (actuator): only for singer #1, controls the sounds and graphics related to the complaining action.

The diagram of fig. 7.9 shows the relationships for *singer #1* and *singer #2* in the first scene of *"SingSong"*. The relationships for the other two singers are identical to those of *#2*. To facilitate the understanding of the diagram, we use the same visual representations for *desired*, *actual*, *activity*, and *event* intervals as those employed in fig. 7.8. Figure 7.10 shows the temporal constraints that define the interval script corresponding to the first scene.

The *desired* interval of all *Chatting* actuators (the top interval in fig. 7.9) and the *actual* interval of *BeQuiet* should start together. This is shown in the first part of fig. 7.10 that states that the *actual* interval of the sensor *BeQuiet* must start OR equal OR i-start with the *desired* interval of *Chatting#1* and *Chatting#2*. In fig. 7.9 we represent this definition by the dashed line joining the beginning of both intervals. The beginning of these intervals is triggered by other intervals not shown here.

The next block in the script states that the *event* interval of *BeQuiet* — *BeQuiet.event* — terminates the *desired* interval of *Chatting#2*, i.e., the singers stop chatting when the conductor raises his arms. *BeQuiet.event* also turns off the *activity* of *BeQuiet*. Also, this event starts *StareConductor#2.desired*. The turning on and off of intervals is described by the start OR equal OR i-start and the meet OR before relationships, respectively, as shown in fig. 7.10.

BeQuiet.activity start OR equal OR i-start *Chatting#1.desired*
BeQuiet.activity start OR equal OR i-start *Chatting#2.desired*

Chatting#2.desired meet OR before *BeQuiet.event*
BeQuiet.activity meet OR before *BeQuiet.event*
BeQuiet.event start OR equal OR i-*start* *StareConductor#2.desired*
BeQuiet.event start OR equal OR i-start *BeQuiet2.activity*

BeQuiet2.activity meet OR before *BeQuiet2.event*
BeQuiet2.event start OR equal OR i-*start* *ReactionTime.desired*
BeQuiet2.event start OR equal OR i-start *StareConductor#1.desired*
ReactionTime.event meet *Complain.desired*

Figure 7.10 Interval script corresponding to the first scene of *"SingSong".*

However, since singer #1 does not stop chatting till the conductor raises his arms for a second time, and *BeQuiet.event* neither turns off *Chatting#1* nor turns on *StareConductor#1*. Instead, *BeQuiet.event* triggers (start OR equal OR i-start) *BeQuiet2.activity*. A detection of an event by *BeQuiet2* shuts off the sensor's activity, and starts the *StareConductor#1* and the *desired* interval of the timer *ReactionTime*. The end of *ReactionTime.actual* starts the *Complain* actuator, finishing the first scene.

The detailed examination of the complete script of *"SingSong"* is beyond the scope of this thesis. However, it is useful to mention typical cases of interaction that were addressed during the development of *"SingSong"*. For example, the tuning scene is basically a loop of short tuning interactions between the conductor and one of the singers until all the singers are tuned. To implement the loop, we used a forgetting mechanism similar to the one defined in chapter 5. In the *"SingSong"* version of the interval scripts, however, the forgetting scheme is handled by a third interval associated with all externals, the *reset* interval. Whenever a *reset* interval happens, the other intervals associated with that particular external are set to *PNF*.

The Complexity of the Interval Script

The complete script of *"SingSong"* includes many different constructions that are handled conveniently by our time interval relationship paradigm. The final script contains 92 externals, each comprised of three intervals, and therefore the whole script involves a PNF-network of 276 nodes. Table 7.5 shows the composition of the interval script of *"SingSong"*. Among the externals, 52 were actuators, 19 were sensors, and 21 were timers. There were 210 constraints explicitly imposed on the nodes of the network. Running PNF propagation on this network was

Table 7.5 The composition of the interval script of *"SingSong"*.

nodes	externals	actuators		sensors		timers		constraints
276	92	52	57%	19	21%	21	23%	210

hardly noticeable on the *SGI Reality Engine*: the majority of the computer power was spent on graphics processing.

The script of *"SingSong"* is simple and its implementation using traditional event-loops would certainly be possible. However, we do not believe that it would be possible to implement *"SingSong"* as fast as we did without the interval script structure. The interval script provided a flexible method to change the script as we designed new routines and tested the interaction.

Problems in the Scripting Process

The main problem with the *"SingSong"* version of the interval scripts ideas was that the script had to be coded implicitly in a long and complex piece of C++ code. Although the conceptual model avoids extremely complex and undebuggable control structures, after a while it became quite difficult to read the C++ file, as it contained not only the interval script code but also all the calls for *Inventor* and *MIDI* routines.

Also, it is not possible to express Boolean constructions as temporal constraints. In *"SingSong"*, triggering conditions depending on more than one interval forced the definition of new externals. We also felt limited by the existence of only three categories of externals. For example, there were no simple ways to define a "loop" external, a quite common structure, except by creating a sensor detector that detected the end of the interval and reset the external through an actuator.

However, the some of the most important ideas were already present in the *"SingSong"* version of the interval script paradigm. Among them, we found very convenient the distinction between *desired* and *actual* intervals; the codification of loops as sequences of intervals that are forgotten; and the possibility to express mutually exclusive intervals.

7.2.4 The Experience

"SingSong" was produced in the summer of 1996 in the Advanced Technological Research Laboratories (ATR) in Kyoto, Japan, with direction, art direction, and performance by Claudio Pinhanez. The design of the computer characters was inspired by masks created by Isamu Noguchi. The song sung by the computer characters was based on a melody executed every day in Japanese schools to mark the end of the classes.

"SingSong" was designed to be enjoyed both as a user experience and as a computer theater performance, that is, as an *immersive stage*. The story lived by the performer and the user is identical, enabling the user to experience the story as lived by the performer. The performer was able to produce a more vivid and interesting result for those observing from outside the system because he could clearly react to the situations and expressively displays his emotions.

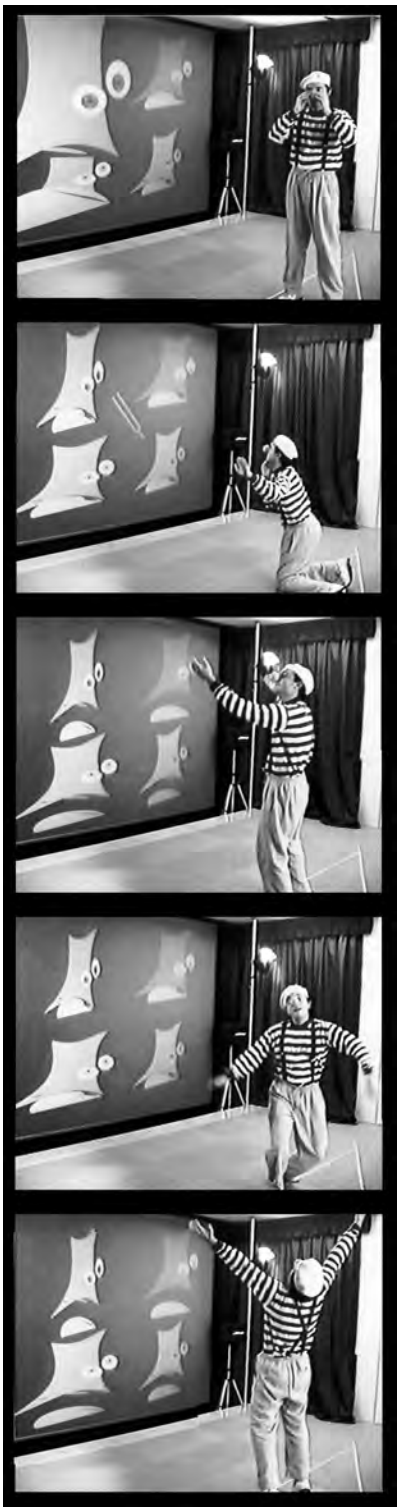


Figure 7.11 Scenes from *"SingSong".*

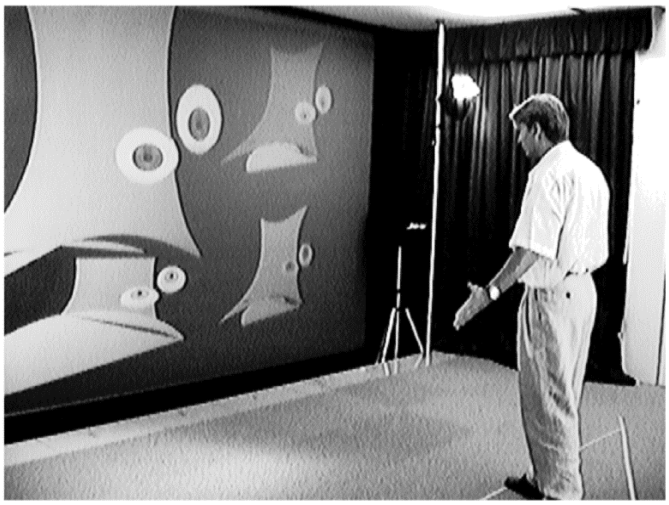


Figure 7.12 Audience playing with *"SingSong".*

"SingSong" was initially performed twice inside ATR for an audience composed of artists and technicians from ATR and NTT. The interactive space was rebuilt at the MIT Media Laboratory and the piece was performed during an open house of the laboratory in September 1996. Figure 7.11 displays some images from a performance of *"SingSong"*. From top to bottom: *singer #1* complains; the conductor pleads to *singer #1* get in tune; the conductor raises his arms, triggering a new note in the song; the conductor keeps conducting while dancing; the *grand finale* of the song.

As a performance, *"SingSong"* was very successful, underscoring our interest in computer theater. In spite of the short length, the audience was engaged and emotionally involved with the story. Many factors contributed to this: first, the choice of a clown costume, complete with red nose, seemed to produce an interesting effect of blending the real and the CG world. The performer's characterization as a clown seemed to put him in a world as fantastic as the singer's virtual world. Second, the strong story created a space where interaction unfolded at the right pace, contributing to the immersion of the performer and the engagement of the audience. Third, there was true "dialogue" between the human and the computer actors as a result of the action-based structure. Finally, pantomime and

clowning produced a mixture of simplicity and appeal that enticed the audiences.

Figure 7.12 shows a user reacting to *singer #1*'s complaints (just after it was commanded to stop chatting). Users seemed to be quite comfortable in assuming the role of the conductor. Sometimes they talked back to the characters and they always laughed when forced to kneel down to attend to *singer #1*. In particular, users appeared to have a great time conducting the chorus. The simplicity of the interface coupled with the joy of generating interesting music provided a very pleasant experience. Also, the well-defined end to the interaction (signaled by the applause) allowed *"SingSong"* to terminate with a dramatic climax.

7.3 *"It / I"*

Our experience with *"SingSong"* showed that it was possible to create an interesting performance with computer-actors. However, after its completion, we became interested in determining whether a computer character on stage could keep its effectiveness in a long and more complex story. Also, we wanted a more challenging environment to develop and explore the newly conceived idea of interval scripts. In the beginning of 1997 we began thinking about a new piece of computer theater, *"It / I"*.

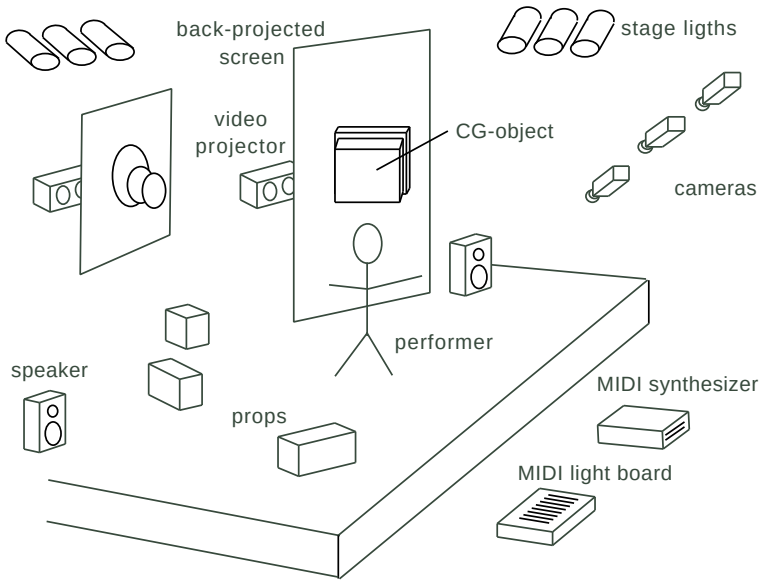


Figure 7.13 Physical setup of *"It / I"*.

"It / I" is a two-character theater play where the human character ***I*** is taunted and played by the autonomous computer-graphics character ***It***. *"It / I"* is about the feeling of being trapped by the fantasy produced by technology.

Figure 7.13 depicts a diagram of the different components of the physical setup of *"It / I"*. The sensor system is composed of three cameras rigged in front of the stage. The computer controls

different output devices: two large back-projected screens; speakers connected to a MIDI-synthesizer; and stage lights controlled by a MIDI light-board.

The play was written considering the sensory limitations of computer vision. That is, the actions of ***I*** were restricted to those that the computer could recognize automatically through image processing. In many ways, ***It***'s understanding of the world reflects the state-of-art of real-time automatic vision: ***It***'s reaction is mostly based on tracking ***I***'s movements and position and on the recognition of some specific gestures (using [43]).

It is relatively easy to create pieces with technology that dazzles the audience for some time, but typically the novelty wears off after ten minutes or so. Among other targets, we wanted to create an experience that lasted for a significant amount of time (at least half an hour) so that we could evaluate whether the audience engagement could go beyond the novelty level. We also wanted the play to be good theater and to be interesting both as theme and story. For this, we were particularly concerned about creating a true "dialogue" between the human and the computer actor. Too often in performances with on-stage screens the performers are upstaged by the screens.

We also wanted to experiment with the idea of immersive stages in *"It / I"*, and we initially aimed at making the play completely "livable" by audience members. In fact, in the initial script of *"It / I"* we included audience participation during the performance but later we decided that the intended scene was too disruptive to the progression of the play.

In terms of technology we were interested in testing our paradigm of interval scripts in a long and complex interaction to determine its effectiveness and possible shortcomings. Also, we felt it would necessary to evaluate whether our action-based approach to interaction (as opposed to gesture-based approaches used in other interactive spaces, [94]) could facilitate the development of interactive spaces.

Finally, we wanted to test a method based on stereo vision (previously developed in our laboratory by Ivanov, Bobick, and Liu [66]) to segment the actor from the background. The main benefit of using this method was that we would be allowed to use stage lighting (including color) and to have large displays in the background of the stage. Until that time, most of the vision systems used in interactive spaces required either uniform background [79], or at least, constant color and luminosity [186].

7.3.1 The Story

The play *"It / I"* is composed of four scenes, each being a repetition of a basic cycle where the human character ***I*** is lured by the computer character ***It***, is played with, gets frustrated, quits, and is punished for quitting. ***It*** has a non-human body composed of CG-objects projected on screens. The CG-objects are used to play with ***I***. ***It*** can also "speak" through images and videos projected on the screens, through sound played on stage speakers, and through the stage lights. There is also a live pianist who plays a piano situated in the middle of the audience. *"It / I"* was inspired by three plays by Samuel Beckett, *"Act Without Words I"*, *"Ghost Trio"*, and *"Waiting for Godot"*.

"It / I"

a pantomime for a human and a computer actor

by Claudio Pinhanez

Prologue

***I** is at the center of the stage, screens are dark, **I**'s eyes are fixated on an electric cable in front of him. After some time, **I** picks up the cable and connect it to a plug in his bellybutton. A piano, off-stage, plays repetitively excerpts from a Beethoven sonata.*

Scene I

***I** is sitting at the center of the stage, distracted by the music played by the pianist. **It** attracts **I**'s attention by displaying a beautiful image of the sun on the left stage screen. When **I** stands up the image moves away, and **It** projects a movie clip with a stopping gesture. The movie plays continuously until **I** executes the gesture. Following, another two movies showing gestures are projected, one depicting a hanging gesture and the other showing a person shooting himself. When **I** performs these gestures the movies stop. A CG-clock appears, running a countdown. **I** tries to hide from what he believes is an imminent explosion. **It** projects again the first movie clip, implying that the clock can be stopped by the stopping gesture. When **I** executes the gesture the clock disappears, immediately being replaced by a new one. **I** tries the stopping gesture, it doesn't work. However, the hanging gesture stops the clock. Clocks continue to appear at a faster rate than **I** can stop them. After some time **I** gives up and protects himself from the explosion he believes is coming. The explosion does not happen. The clocks disappear and the piano music returns.*

Scene II

***I** is listening to the piano but **It** again attracts his attention with an image — a picture of a family. When **I** stands up the image moves away. This time, a CG-object similar to a photographic camera appears on the right screen and follows him around. When **I** makes a pose, the camera shutter opens with a burst of light and a corresponding clicking sound is heard. On the other screen a CG-television appears and, when **I** gets close, the television starts to display a slide show composed of silhouette-images of **I** as if they were "taken" by the CG-camera. After some pictures are shown, the camera "calls" **I** to take another picture. This cycle is repeated until **I** refuses to take yet another picture and stays in front of the television, provoking an irate reaction from **It**, which throws explosions of CG-blocks into **I** while flickering the lights and playing really loud noise. The camera and the television disappear. Piano music returns.*

Scene III

***I** is attracted again to play by an images of angels. When **I** stands up the image moves away. An agitated, restless electric switch-like CG-object appears on the left screen. Whenever **I** gets close to the screen the switch moves up, always staying far from the reaching of **I**'s hands. **I** is then told (by video clips) that he can control the switch's position by standing on the top of blocks and making special gestures. When **I** finally gets close enough and seems to be able to turn the switch off, **I** discovers that the objects are only projections on a screen. In a Beckettian way, **I** tries to hang himself — without success — and provokes another round of explosions of CG-blocks from **It**. The switch disappears and the piano plays repetitively a very sad and short segment of Beethoven's Ghost Trio.*

Scene IV

*An image of flowers succeeds again to attract **I**'s attention. But this time, after the image disappears, **I** decides to ignore **It** and disconnects the plug from his bellybutton. Nothing changes. **It** then brings all the objects to the screen, trying to force **I** to play. **It** also plays all the movie clips with the gestures trying to motivate **I** to interact. Finally, given the lack of response, **It** brings the switch to the screen and turns it off, leaving **I** in an empty, silent, dark world.*

Epilogue

***I** is at the center of the stage, screens are dark, **I**'s eyes are fixated on an electric cable in front of him. After some time, **I** picks up the cable and connects it to a plug in his belly-button. A piano, off-stage, plays repetitively excerpts from a Beethoven sonata. Light fades.*

Notice that the basic sensory capacity in scene I is the recognition of body shapes. The information that ***I*** has stood up triggers the disappearance of the image of the sun and the arrival of the clocks. The stopping gestures control the turning off of the clocks and their absence causes the explosion.

The actions of ***It*** in scene II are solely based on the position of ***I***. ***I***'s interest in ***It*** is assumed when he gets close to either the camera or the television. For instance, the refusal to continue taking pictures is detected when the camera "calls" three times and ***I*** does not move from the front of the TV screen. However, the sensing allows quite flexible interaction in this scene. The number of "take-picture-watch-tv" cycles is not pre-determined: the human actor decides to refuse to play when he — as a performer — believes the character (and maybe the audience) has reached the emotional potential to do that. Also, there is a great deal of room for improvisation in terms of when and how the pictures are taken since the camera clicks only when the actor has been still for some seconds.

Figure 7.14 Moment from scene II of *"It / I".* The actor is interacting with the TV-like object projected on the screen behind him.

In scene III, the interaction is based on a more complex set of elements. Initially just ***I***'s position relative to the screen is used to control the height of the switch on the big screen. But to lower the switch, a complex action must be recognized. First, the human character has to bring a block close to the screen, then he has to pick up another block and then execute special gestures while standing on top of the first block. To recognize such complex structure using only information from the silhouette — as detailed later — we employ a method based on the action recognition scheme developed in chapter 4. In the last scene and in the epilogue, the play relies mostly on the actions of the computer character ***It***.

From the script it is evident that the complexity of the interaction in *"It / I"* is quite beyond previous full-body interactive systems. For example, in *ALIVE* [94], although the main character (the dog-like CG-character *"Silus"*) had quite a complex internal structure, the meaning of the user's gestures remained constant as the interaction proceeds. In addition, *"It / I"* has a clear dramatic structure not present in most previous interactive spaces as, for instance, the spaces designed by Krueger [79] and Sommerer and Mignonneau [161].

7.3.2 The Technology

The major technical novelties in *"It / I"* are the interval script paradigm used to describe the story, the interaction with the human actor, and the behavior of ***It***. Before examining in more detail how the story described above is represented using interval scripting, we want to describe other technological solutions we used in the play that can be employed in other interactive

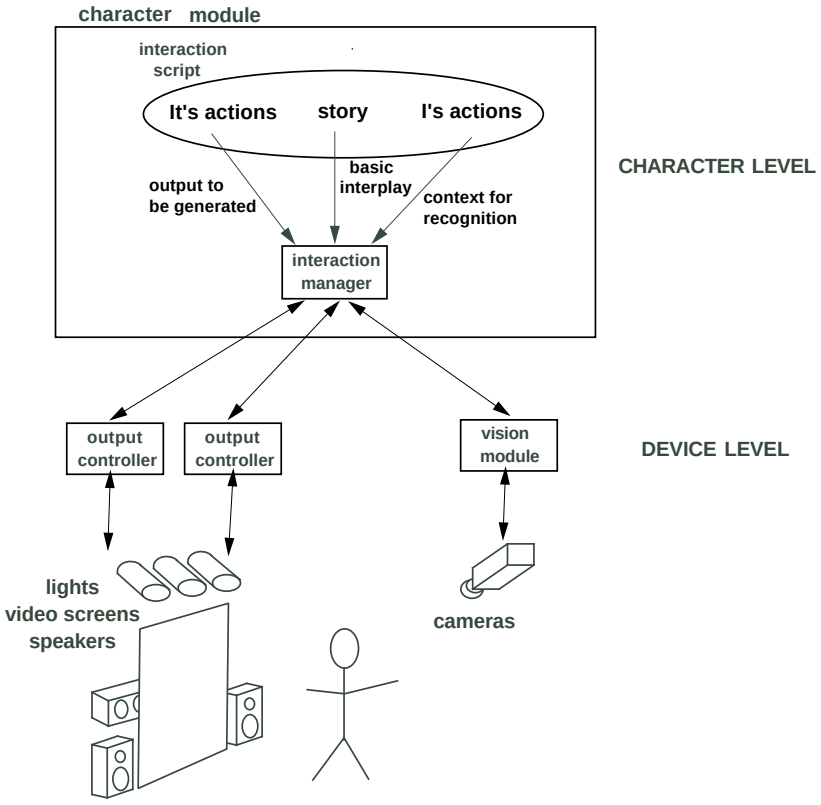


Figure 7.15 System architecture of *"It / I".*

spaces. In particular, in *"It / I"* the architecture of the system was based on creating a special module corresponding to the character ***It***.

System Architecture

Figure 7.15 displays the control architecture used in the performances of *"It / I"*. It is a 2-layer architecture in which the upper layer contains information specific to the computer character and the bottom layer is comprised of modules directly interacting with the actual input and output devices. This control model is a simplification of the 3-layered architecture, called *story-character-device* architecture, or *SCD*, that was described in chapter 6.

As shown in fig. 7.15, the computer character control system is composed of one active element, the *interaction manager*, that processes the interaction script. The interaction script contains three types of information: the description of the *story* in terms of actions to be performed by the computer and the human characters; the specification of ***It***'s actions, that is, what the computer character actually does when trying to perform the actions needed by the story; and the description of how the human actor's movements are recognized according to the current moment in the story, or of ***I****'s actions*.

For instance, in a scene where the computer attracts the attention of the human character by displaying images on the screen, the basic interplay is the goal of attracting ***I***'s attention. As part of the description of ***It***'s actions there is a method that associates attracting the human character's attention with the displaying of particular images, each moving with a particular trajectory on the screen, accompanied by music and warm light. This is how the action *"It attracts I's attention"* is translated into output to be generated. At the same time, the story sets up the context for recognition of a movement of ***I*** walking towards the screen as *"I is paying attention"*; in most other situations of the play, such movement is not recognized as so.

The Vision System

The vision module shown in fig. 7.15 performs basic tracking and gesture recognition. The tracking module answers queries about the number of persons on stage (none, one, more than one); the position (x,y,z coordinates) and size of one person (when present) and three large blocks; and the occurrence of the five pre-trained gestures.

In the performance setup we employed a frontal 3-camera stereo system able to segment the actor and the blocks and to compute the silhouette image that is used to track and recognize gestures. The stereo system, based on the work of Ivanov et al. [66], constructs off-line a depth map of the background — stage, backdrops, and screens. Based on the depth map, it is possible to determine in real-time whether a pixel in the central camera image belongs to the background or to the foreground, in spite of lighting or background screen changes.

This is an considerable improvement over vision systems based on background subtraction used before in many previous interactive spaces [18, 94, 161] because it enables lighting change, an important dramatic element. During the performances of *"It / I"* in 1997, the segmentation part of the vision system ran at 8 frames per second on a *SGI Indy 5000*. The analog video signals from the three cameras were compressed into a single video stream using a quad-splitter and then digitized into 640x480 images (a video quad-splitter is a device that takes 4 input analog video streams and produces a single analog signal composed of the four images in half-size and tiled).

Figure 7.16 shows a typical visual input to the system and the silhouette found. Using the silhouette, a tracking system analyses the different blobs in the image to find the actor. The analysis is performed under assumptions about the continuity and stability of movement, position, and shape of the actor. Smaller blobs are labeled as blocks only when isolated from the person's silhouette.

The vision system also must recognize the five different gestures shown in fig. 7.17. To perform gesture recognition we employ a simplification of the technique described by Davis and Bobick in [43]. During the performances we also employed manual detection of some of the gestures that could not be detected by the system with enough reliability. We had to resort to manual detection due to an absolute lack of time to improve the reliability to performance levels, which must be very high, since during a performance accuracy is essential to avoid confusion in the actor or breaking the flow of the performance.

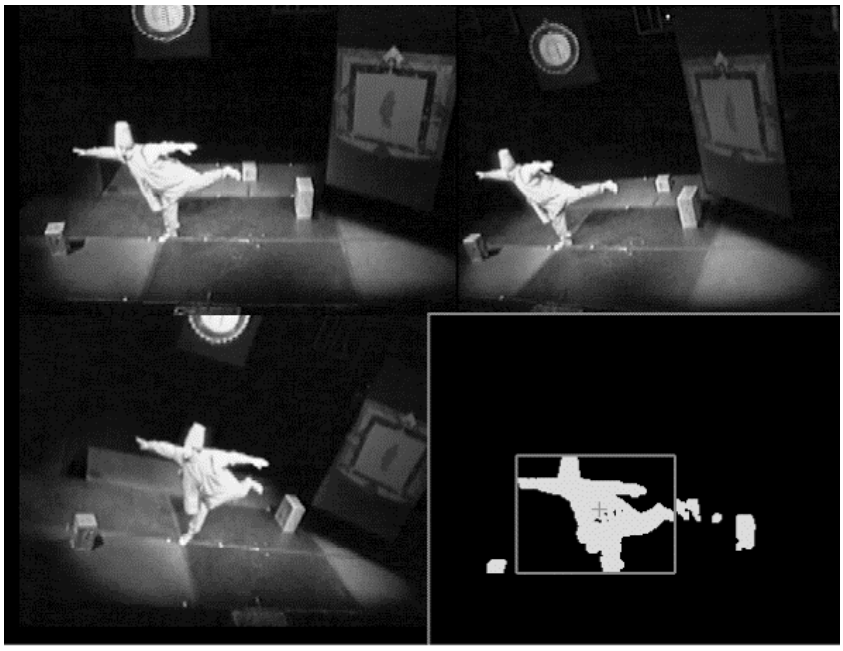


Figure 7.16 The 3-camera input to the vision module and the computed silhouette of the human actor (enclosed in the rectangle).

Figure 7.17 The 5 gestures used in *"It / I".*

Because the recognition of this type of gesture has already been demonstrated in computer vision (see [43]) and in a real-time environment [44], we believe that there were no real technical obstacles for a fully autonomous run of the play. Also, three of the scenes of the play could be run completely without human help.

The Computer Graphics Modules

The computer graphics modules control the generation and movement of the different objects and flat images that appear on the two stage screens. Each CG module basically processes ACTSCRIPT requests (whose format we exemplify later) translating them into *Inventor* commands.

In our system, whenever an object or image moves, sound is produced. The importance of sound to enhance image is well known in the movie industry but, surprisingly, only a few interactive CG characters and environments seem to have explored this facet of animation. Here we are not talking about the inclusion of sound effects in the post-production phase of CG videos, or about synchronizing speech to talking characters. Rather, we are interested in the automatic generation of pre-defined sound effects as the objects move around the world.

We implement sound production through an interesting method. The sound effects are contained in short MIDI-files that are associated with each object's translational and rotational movements through an *Inventor Engine* object. The MIDI-file is sent to the synthesizer — through a request command to the sound module — according to a threshold function describing when the file must be played. Typical examples of such functions used in *"It / I"* are: a function that plays the sound periodically while the movement lasts; a function that looks for peaks in the velocity of the CG-object; and a function that triggers the MIDI-file whenever there is a peak in the derivative of the curvature of the path. The last function is used quite successfully to automatically play swiveling sounds when the CG-camera is performing fast rotational movements.

The Sound Module

All the sounds and music in *"It / I"* are produced by a Korg X5DR MIDI-synthesizer. To control the production of the sounds, we have developed a module that could send the MIDI events corresponding to the different MIDI files associated to each sound and piece of music. In particular, this module can produce the sound of multiple files simultaneously. Since the synthesizer had only 16 channels, we had to constrain all the sounds to use the same 16 instruments, or MIDI patches. This constraint ended up producing the positive effect of creating a common theme and a similar sonority among the different sounds, like a "palette" of basic colors in a painting.

The Light Control Module

In general, light boards employ DMX protocol to communicate with the light dimmers and with external devices. In our case, however, the light board was able to accept MIDI commands to trigger pre-defined light cues. We used this MIDI interface to connect the light board to our computers. Each light wash was associated to a cue and the light control module had a look-up

```
(request
   (action "left-cg-module"
      (move (object "camera")
         (direction
            to (location (0.0 0.4 0.5))
            from (location (0.0 3.0 -1.0)))
         (path
            (location (0.0 -0.03 0.0)))
      (when NOW)
      (velocity (special "crescendo"))
      (interval (timed duration 5.0)))))
```

Figure 7.18 Example of a request for a movement of the camera in ACTSCRIPT.

table with the correspondence between a state of the stage lights and a MIDI note. When a cue was requested, the module simply fired the MIDI-note corresponding to the cue. Notice that all the control of intensity of lights and change speed was done by the light board, as well as simple effects such as flickering lights.

Communication Between Modules

For the communication among modules we employed the ACTSCRIPT language described earlier in chapter 2. As noted there, ACTSCRIPT is able to represent actions, requests, queries, and goals in a physical environment. Unlike previous work in languages for CG characters (for example, [77, 125, 170]), ACTSCRIPT allows recursive decomposition of actions, specification of complex temporal relationships, and translation to/from natural language.

The communication is done by the exchange of ASCII strings between the modules. The actual transmission of the text strings among machines used the communication package *PVM* [54], modified to make sure that only one version of each module was running on all the machines at all times. Compared to our previous experience using the *RPC* protocol (in *"SingSong"*), we noticed a significant improvement in the reliability of the communications and in the easiness of use. For instance, it is trivial to kill a module, restart it, and keep the communication with other modules working seamlessly.

Upon receiving a message, the string is parsed by an interpreter that identifies the interesting components for that particular module. The expressiveness of the ACTSCRIPT language was the main reason for its use. Besides that, we notice that it is very convenient to use a language where calls and messages can be easily simulated, composed, and read. In fact, in some rehearsals we were able to use manually controlled interaction whenever it was needed, simply by typing ACTSCRIPT requests and questions.

Figures 7.18 shows examples of a movement as expressed in ACTSCRIPT. Figure 7.18 is an example of a request from the ***It*** module to the `left-cg-module` to move the computer graphics camera (the object that appears on the screen) from one determined location to other going through a specified `path` position. The movement is determined to take 5 seconds, start immediately, and follow the pre-defined velocity profile curve called `"crescendo"`. Upon receiving this request, the CG module checks if the object is available for movement, and in the

positive case, the movement is started. A message — also in ACTSCRIPT — is sent back to the ***It*** module in both cases to communicate the actual state of the request. When the movement is finished, another message is sent to ***It***.

7.3.3 The Scripting Process

"It / I" was conceived from the beginning to be a full-length play with multiple scenes and complex interactive structures. This made the play a great test for our paradigm of interval scripts. However, as soon as the project started in January of 1997, it became clear that the interval script paradigm needed to be improved and, especially, that a better interface should be provided.

By the end of May of 1997 a new version of interval scripts had been developed and it proved to be essential for the accomplishment of our goals of having an interesting and responsive automatic computer character on stage. Here we present excerpts from the script of one of the scenes of the play, covering from the middle-level control structures to the highest level structure of the scene. We believe that those examples decisively support our claim that interval scripts are a powerful paradigm to script interactive spaces.

Interval Scripts: the *"It / I"* Version

There are major differences between the interval scripts of *"SingSong"* and *"It / I"*. First, the initial version employed C++ classes to interface the interval structure. Since we found that the resulting code ended up being difficult to read, we decided to develop a language to describe intervals and their constraints. Interval script files in this language are converted into C++ code through a special compiler that encapsulates the different intervals in function calls. *"It / I"* was developed using this compiler.

By this time, we had also found that a better way to represent the distinction between wish and occurrence is the START, STOP, and STATE functions (as opposed to the double interval concept that founded the *"SingSong"* version). Using this structure, the triggering of the functions needs to be accomplished by comparing the current and the desired state of all intervals, as explained in chapter 5.

The biggest difference between the version described in chapter 5 and the *"It / I"* version of interval scripts is in the way that when statements are handled. In the current version, when statements are translated into an interval containing the condition and, whenever the condition becomes true, it becomes "desirable" that the resulting action happens. In the *"It / I"* version, when statements are implemented as functions that are called after the current state of all intervals is determined and PNF propagation is run. Although conceptually simpler, in the *"It / I"* implementation the ability to forecast the next state is not present, weakening the control power of the temporal constraints.

There are other differences between the *"It / I"* version and the described in chapter 5 (the current version). In particular, the former allowed the declaration of conditions sensitive to **changes** of state. For instance, in the *"It / I"* version it is possible to declare that an interval should be started exactly when another interval changes its state — using the special syntax

```
"family image scene" =                                                          (1)
{   START: tryto start "bring family image".                                    (2)
    STOP: tryto start "remove family image".                                    (3)
    when "bring family image" is past                                           (4)
        tryto start "minimum time of family image".                             (5)
    when "I is seated" is past                                                  (6)
            AND "minimum time of family image" is past                          (7)
        tryto start "remove family image".                                      (8)
    STATE:                                                                      (9)
        if "bring family image" is now                                          (10)
            OR "remove family image" is now assert now,                         (11)
        if "remove family image" is past assert past.                           (12)
    "bring family image", "remove family image", "minimum time of family image" (13)
        only-during "family image scene".                                       (14)
}.                                                                              (15)
```

Figure 7.19 Example of a definition of an interval from scene II of *"It/I".*

becomes. After the performances, we realized that it is conceptually simpler to think only about states of intervals and never about changes since it makes the run-time engine more robust to errors and unexpected situations.

The *"It / I"* version also did not allow the declaration of nested intervals and therefore all sub-actions of an action needed to be manually constrained (by only-during constraints). The incorporation of nested structures into the interval script language considerably simplified the definition of those situations.

Finally, there are other small differences in syntax between the *"It / I"* version and the current. To set the state of the current interval, the *"It / I"* version used the keyword assert instead of set-state. Also, we used the structure repeat if-after instead of the simpler concept of cycle intervals.

Script Examples from *"It / I"*

In chapter 5 we have covered the basic syntax and semantics of interval scripts. Our goal here is to examine three segments of the interval script of scene II of *"It / I"* to have a better idea of how the interaction described in the script of the play is mapped into the control code.

Our first example corresponds to the beginning of scene II when ***It*** brings an image of a family to the screen and moves it away when ***I*** shows interest on it by standing up. Figure 7.19 shows the definition of the interval *"family image scene"* that controls this short scene. As we saw in chapter 5, interval scripts allow complex combinations of start and stop functions of different intervals when describing higher level functions, as it is the case of *"family image scene"*.

According to fig. 7.19, the definition of the interval *"family image scene"* is composed of the definitions of START, STOP, and STATE functions, and two when statements. The associated meaning is that when the interval *"family image scene"* is started (by a when statement not

shown in the figure), the executed action is to call the START function of the *"bring family image"* interval (in this case, executing its corresponding C++ code) as stated in line 2. Similarly, to stop the scene, the image is removed from the screen by a call to start the interval *"remove family image"*. The STATE function of *"family image scene"* is also defined based on previously defined intervals. If either *"bring family image"* or *"remove family image"* are happening now, the PNF-state of the interval is defined to be *now*; if *"remove family image"* is over, the state is *past*; otherwise, an undetermined PNF-state is assumed as default.

Notice that when the STOP function is called, the interval does not immediately goes to the *past* state, since it takes time to remove the family image from the screen. This is the typical case of the distinction between desire and actual occurrence of an interval as discussed in chapter 5. When the interval *"family image scene"* is asked to STOP, it starts the execution of an action that eventually finishes the interval. This guarantees the designer that, no matter what happens, a defined interval can be set to start and stop smoothly.

The scene of the family image involves bringing the image to the screen, waiting some time to make sure that the image is seen by the audience and then checking if the human actor has stood up. This succession of actions is controlled by two when statements. In lines 4 and 5, the end of the interval *"bring family image"* is set to trigger the beginning of the timer *"minimum time of family image"* (defined previously in the script). The end of the scene is triggered by the other when statement that asserts that when both the timer *"minimum time of family image"* has expired and the detector *"I is seated"* is no longer true then it is time to start removing the family image. Notice that, once started, the interval has mechanisms to stop itself; it also can be stopped at any moment by an external call to its STOP function.

Line 15 of *"family image scene"* is a statement that declares that *"bring family image"*, *"remove family image"*, and *"minimum time of family image"* can only happen during *"family image scene"*. This statement creates a temporal constraint linking the PNF-state of all these actions and preventing, for instance, the call of the start function of *"remove family image"* if *"family image scene"* is in the *past* state, even if all the conditions listed in the when declaration apply.

This first example from the script of *"It / I"* shows how easily scenes composed of multiple segments can be assembled in interval scripts. Also, as noticed above, it demonstrates how simple is to create a scene that develops by itself but also has graceful finishing mechanisms.

Let us now examine a more complex segment of scene II. In this case, we want to define the core of the scene, when ***It*** repetitively plays with ***I*** a "game" of taking and showing pictures. Basically, the segment is composed of a loop where the ***It*** calls ***I*** to take a picture (by "gesturing" with the camera), aims the camera, waits for the human actor to be still, takes the picture, and displays all the pictures in a sort of "slide" show in a TV-like computer graphics object. The cycle is finished when the human character refuses to attend the call of ***It*** by not walking to the area in front of the screen where the camera is, provoking a series of explosions, composed of CG-blocks, on the screens.

```
repeat "It aims the camera"                                                    (1)
   if-after  NOT "I is close to big screen"                                    (2)
        AND NOT "It takes picture" is past-or-now.                             (3)

"point and click" =                                                            (4)

{                                                                              (5)
START: tryto start "It aims the camera".                                       (6)

STOP: tryto start "It takes picture".                                          (7)

STATE: if "It aims the camera" OR "It takes picture" assert now,               (8)
          if "It takes picture" IS PAST assert past.                           (9)
}.                                                                             (10)

"It aims the camera" only-during "point and click".                            (11)

"It takes picture" only-during "point and click".                              (12)

when "It aims the camera" becomes past                                         (13)
     AND ((NOT "I is close to big screen") AND "I is in the same aiming area" AND "I is still")  (14)
  tryto start "It takes picture".                                              (15)

when "point and click" becomes past                                            (16)
     tryto start "It brings tv".                                               (17)

when "I is close to big" AND "It brings tv" is past AND "point and click" is past  (18)
    tryto start "tv shows slide show".                                         (19)

when "tv shows slide show" becomes past                                        (20)
    tryto start "camera calls attention".                                      (21)

"tvcamera cycle" =                                                             (22)
{                                                                              (23)
START: tryto start "point and click".                                          (24)

STOP: tryto stop "tv displays pictures".                                       (25)

STATE: if "point and click" OR "tv shows slide show" OR "camera calls attention"  (26)
          assert now,                                                          (27)
       if "camera calls attention" is past OR "explosion cycle" is now         (28)
          assert past.                                                         (29)

RESET: tryto reset "point and click", "tv shows slide show", :"camera calls attention".  (30)
}.                                                                             (31)
```

Figure 7.20 Part of the script of scene II of *"It / I".*

Figure 7.20 shows the part of the script of scene II that corresponds to the definition of repetition cycle. This segment uses other intervals whose definitions are not included here. In most cases these intervals simply encapsulate C++ code and their effects are easily understood by their names.

The first lines of the script in fig. 7.20 define a repetition cycle based on the previously defined action *"It aims the camera"* which corresponds to a rotation of the camera object so it points to the area where ***I*** is. Basically, the action of aiming the camera is repeated as long as, at the end of the cycle, the human character is not close to the big screen (what happens when he is watching the "slide" show) and the camera is not taking a picture. This guarantees the repetition of the aiming action while ***I*** wanders close to the small screen.

Following, in lines 4 to 10, the interval *"point and click"* is defined to control the occurrence of the actions of aiming and clicking. In particular, this interval is used to guarantee that the actions *"It aims the camera"* and *"It takes picture"* occur only when the interval is happening, by the two constraints declared on lines 11 and 12. The when statement of lines 13 to 15 captures the moment when there is a chance to take a picture. Notice that the triggering of the action *"It takes a picture"* will stop the repetition of *"It aims the camera"* but, according to the when statement, this can happen only when one iteration of the aiming action is finished (as written in line 13).

When the picture is taken, lines 16 and 17 trigger the bringing of the TV-like object to the big screen. Although the action of taking pictures is repeated many times, the TV needs to be brought only once. What happens is that, although the interval *"It takes picture"* goes to the *past* state many times (and it is forgotten by the action of a repeat statement shown in the excerpt), the action of bringing the TV goes to *past* and is not forgotten. Therefore, although the condition of the when statement is true, the command tryto start has no effect on an interval that has occurred.

To make the TV display the "slide show" of the pictures taken of ***I***, we employ the when statement of lines 18 and 19. Notice that it requires the TV object to be visible, the human character to be close to the screen, and a picture to have already been taken. To complete the cycle, lines 20 and 21 force the camera to make movements to attract ***I***'s attention, through the action *"camera calls attention"*.

Finally, a complete iteration of the cycle is defined in *"tv camera cycle"* (lines 22 to 31). The definition is quite similar to the action *"family image scene"* described in fig. 7.19, except for the declaration of the RESET function that provides instructions about how to forget the occurrence of an interval. In this case, it simply runs the resetting function of the action's constituent sub-actions.

The script in fig. 7.20 is a good example of scripting interaction of intermediate level. As we see, the interaction is completely described using interval script structures and references to low-level intervals. This ability to abstract into more complex actions proved to be extremely helpful during the programming of *"It / I"*. Not only is the language cleaner than normal programming code (i.e., C++), but it also avoids explicit references to what happens in the periods of time when actions are starting or finishing. The START, STOP, and STATE functions, by decoupling wish and occurrence of actions, prevent the designer from thinking about intermediate states.

Finally, we want to examine the high-level script that controls the whole scene II of *"It / I"*, as depicted in fig. 7.21. The first two lines ensure that after the *"family image scene"* happens, the camera object is brought to the small screen, and starts to look for picture opportunities.

```
when "family image scene" becomes past tryto start "It brings camera".                (1)
when "It brings camera" becomes past tryto start "point and click".                    (2)
"family image scene" before OR meet "point and click".                                 (3)
"It takes picture" before OR meet "explosion cycle".                                   (4)
repeat "tvcamera cycle" if-after "I pays attention".                                   (5)
when "tvcamera cycle" becomes past AND "I does not pay attention"                      (6)
    tryto start "explosion cycle".                                                     (7)
"tvcamera cycle" before OR meet "explosion cycle".                                     (8)
when "explosion cycle" is past                                                         (9)
    tryto start "remove tv", "remove camera".                                          (10)
```

Figure 7.21 Basic script for the whole scene II of *"It / I"*.

Lines 3 and 4 put constraints on the occurrence of some of the involved actions; these constraints are important to prevent unexpected interactions during the development of the script of the scene. In fact, these two constraints are typical examples of information that is apparently redundant but in fact necessary to prevent run-time mistakes. These problems typically arise from the fact that it takes different lengths of time for different actions to start happening. In those conditions, weakly defined when statements may trigger and it is hard and cumbersome to write down all the particular circumstances when they should not be activated. Notice that these kind of "natural" constraints can always be added to the script if they are redundant. Besides providing safety, they also make the script clearer.

The main piece of the script is the repeat cycle of line 5, which keeps repeating the action *"tv camera cycle"* whenever the ***I*** character pays attention after being called (remember that the last sub-action of *"tv camera cycle"* is *"camera calls attention"*). In the play, *"I pays attention"* just by walking towards the camera and therefore, when that does not happen after he is called, the condition of line 6 becomes true and ***It*** starts the *"explosion cycle"*, that is, computer graphics rocks are thrown towards ***I*** in both screens. This cycle finishes automatically and, after that, both the camera and the TV are removed from the screens according to line 10.

Figure 7.21 exemplifies the simplicity obtained by an interval script description of a complex scene. This script is a compelling example of the level of abstraction achieved by our paradigm. Without this power, we do not think it would be possible to manage the difficulty of creating a script of a 40-minute interactive performance.

The Complexity of the Interval Script of *"It / I"*

For performance safety, we implemented each scene of *"It / I"* in a separate script, generating four different executable files that were loaded in the beginning of the corresponding scene. Table 7.6 summarizes the composition of the interval scripts of each scene of *"It / I"*, displaying the total number of intervals (i.e., the number of nodes of the PNF-network), the number of those that were composed exclusively of C++ code, the number of intervals defined

Table 7.6 The composition of the interval scripts for the different scenes of *"It / I".*

	total number of intervals	"C++"-only intervals		intervals defined on previous intervals		timer intervals		when	repeat	constraints
scene I	120	51	43%	31	26%	38	32%	77	7	11
scene II	80	33	41%	17	21%	30	38%	52	8	18
scene III	92	46	50%	18	20%	28	30%	55	8	11
scene IV	115	41	36%	27	23%	47	41%	77	2	20

based on previous intervals, the number of timers, and the number of when statements, repeat structures, and explicitly defined constraints.

As can be seen in the table, we have approximately 100 intervals per scene, of which about half are related to the interface to other elements of the system — the "C++"-only intervals. On average, 20% of the intervals were constructed based on previously defined intervals (as described in chapter 5 and exemplified above). In fact, in our experience we have found that the ease of abstracting interval scripts to be extremely helpful for the design and development process.

Notice that about 40% of the intervals are simply timers. This does not come as a surprise to anyone who has actually built an interactive space, since it is extremely important to have pauses and "beats" to smooth and connect the different pieces. Table 7.6 also shows that the primary mechanism for controlling the experience in *"It / I"* ended up being the when statements and not the temporal constraints. However, their role of avoiding undesired situations is very important, as we could see in the excerpts of the script shown in figs. 7.19, 7.20, and 7.21.

In general, we found that the two most important contributions of the interval scripts to the development of *"It / I"* were the ease of defining composite actions from previously defined intervals; and the separation of wish and reality provided by the START, STOP, and STATE functions and the when statements.

Problems in the Script Process

The problems with intervals scripts detected during the implementation of *"It / I"* informed the design of the current version of the interval script language. For example, it was tedious to set manual constraints for sub-actions of every action and, in practice, we ended up only declaring those constraints whose lack resulted in a failure of a particular run. In the current version, some constraints are automatically defined by the concept of nested intervals, i.e., by the definition of an interval inside another. We also solved a similar problem with when statements: in our current version, if they are defined inside an interval, their condition automatically requires the inclosing interval to be happening *now*.

Although simple, the design of the run-time engine of the *"It / I"* version of interval scripts was not able to take full advantage of the prediction powers of constraint propagation as discussed in chapter 5. In particular, there were no recovery mechanisms and therefore constraints had to be carefully placed to avoid unexpected interactions that could, sometimes, stop all actions until some sensed state changed again. The idea of computing the desired state by PNF-

propagation and thinning (as described in chapter 5) considerably improved the robustness of the run-time engine, allowing, for instance, the graceful survival of conflicts. In particular, the new engine eliminated the need of implementing the call of the when functions in a separate segment of the running cycle, resulting in a more homogeneous and stable system.

7.3.4 Action Recognition in *"It / I"*

To recognize the actions of the character ***I***, the script of *"It / I"* employs techniques similar to the ones described in chapter 4. Although the situations in the play are in general quite simpler than the cooking show cases we discussed in chapter 4, it was necessary to make sure that any tracked movement or detected gesture considers the current context of the play. For instance, if the human actor approaches the right screen, this movement is recognized as *"I pay attention to camera"* in the middle of the second scene and as *"I tries to attract the camera attention"* at the end of the same scene. The only difference between these two situations is the context in which they happen.

Such conditions are easily expressed in interval scripts by only-during constraints and/or STATE functions. Similarly, we implemented the recognition of pieces of interaction using the same mechanisms. For instance, in fig. 7.2 the interaction *"tv camera cycle"* is detected when either of its sub-actions happens and finishes when either the last action is finished or a subsequent action — *"explosion cycle"* — starts to happen. Although we do not see an explicit IA-network representing the action, the definition is implicitly provided by the temporal structures of the interval script and, in practice, the detection of the occurrence of the action follows the same principles as the ones described in chapter 4.

Notice that in the previous example that we are not recognizing an isolated human action, but the interaction process between the machine and user. That example typifies our view that, in an interactive space, it is important to model and recognize not only the user actions, but also the computer actions and the interaction itself. In fact, *"It / I"* has been an interesting domain for recognition research precisely because the high level actions are in reality, dialogues between the man and the machine.

7.3.5 The Experience

"It / I" was produced in the summer/fall of 1997 by Aaron Bobick with direction of Claudio Pinhanez. Raquel Coelho was responsible for the design of the set, the costumes, and the creation of the computer creature ***It***. All the CG-objects that compose ***It*** have a non-realistic look and are composed of the same basic CG-elements assembled in different proportions. Freedom Baird composed the special sound effects associated to the movements of ***It***, preserving the non-realistic concept, and based on a common ensemble of 15 MIDI instruments.

John Liu and Chris Bentzel built the vision system and also contributed for the implementation of the run-time system. Richard Marcus designed the light considering not only the motifs of the play but also making sure that there was enough light for the vision system to operate. Monica Pinhanez selected and arranged the excerpts from Beethoven's piano sonatas and from the *"Ghost Trio"*, and played the piano during the performances. Joshua Pritchard performed

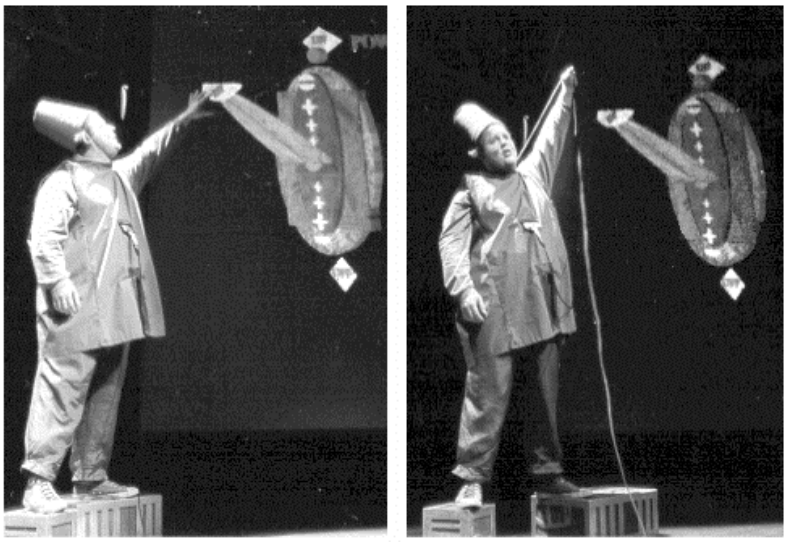


Figure 7.22 Two moments from scene III of *"It / I".*

as ***I***, creating a character based on the idea of tragic clowns. The production was advised by Prof. Janet Sonenberg from the MIT Music and Theater Arts Department.

"It / I" was performed six times at the MIT Media Laboratory for a total audience of about 500 people. We clearly noticed that the audience easily understood the computer character's actions and intentions. Particularly, the play managed to keep the "suspension of disbelief" throughout its 40 minutes. The sound effects played a key role on creating the illusion that ***It*** was alive and to convey the mood and personality of the character.

We were very pleased to see that the highly interactive structure of *"It / I"* seemed to avoid the problem, mentioned earlier, of on-stage screens attracting excessive attention. The play unfolded as a dialogue between the two actors, with the initiative and preeminence shifting from one character to the other.

We did not expect that the audience would react to the theme and the story of the play as strongly as they did. In general, as the play approached its end, there was a deep silence in the house, in what seemed to be an expression of sadness and worry. Talking to the audience after the performance revealed that the people found it disturbing to see a human being, even a comic clown, to be treated in such a way by a machine. We believe that the play stroke a fear that is hidden in our everyday interaction with computers: the fear of being dominated by machines.

Each performance was followed by an explanation of the workings of the computer-actor. After the explanation, we invited the audience to go up on stage and play scene II — the scene where ***It*** plays with a camera and a television in front of the audience. It was a lot of fun and most participants actively looked for new ways of producing their pictures on the screen (see fig 7.23).

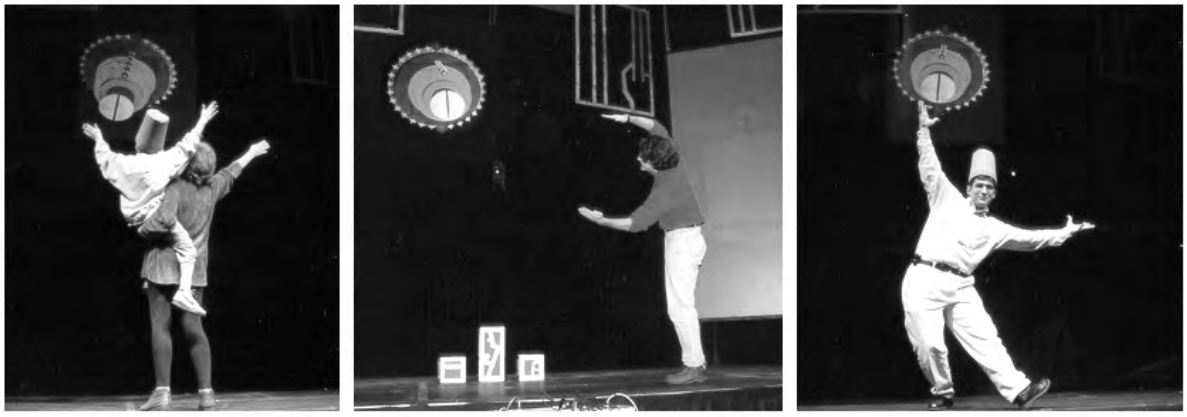


Figure 7.23 Audience participation in *"It / I"*.

However, we never observed the audience-transformed-into-actors to get deeply involved in the story. One of the reasons for this is that they performed in front of an audience and it is hard — for non-actors — to become emotionally engaged in front of a large group of people. Besides that, the control system was not really designed to handle normal users. In particular, it had poor ways to handle user confusion and no strategies to provide help or suggestions. To solve the first problem we thought about creating pieces of scenario that would materialize the fourth wall (that is, cover the front of the stage), as we envisioned in the *"Waltz #6"* project described in chapter 6. To address the confusion problem, we decided to rethink the play as an interactive space for users in what became the project *"It"* described in the next section.

In the performances held in November of 1997 the recognition of gestures was not robust enough and the occurrence of some gestures in two of the five scenes had to be manually communicated to the system. Otherwise, the 40-minute performance was completely autonomous, especially during the audience part. To our knowledge, *"It / I"* is the first play ever produced involving a character automatically controlled by a computer that was truly interactive.

7.4 *"It"*

Although in *"It / I"* we experimented with the idea of putting the audience inside a theatrical play, we felt that it was necessary to develop further the automatic control system to allow full dramatic immersion. Moreover, the open stage proved to be a hard place for non-actors to experience the story of the play, mostly due to the discomfort of being watched.

To overcome these problems we decided to re-create the environment of the play *"It / I"* in a version for users that we called *"It"*. Using the same visual and sound material, basic story/theme, and conception, a new interactive space was created. Inhabited by the *It* character the space tries to trap the user inside the space, under the disguise of a game of taking and

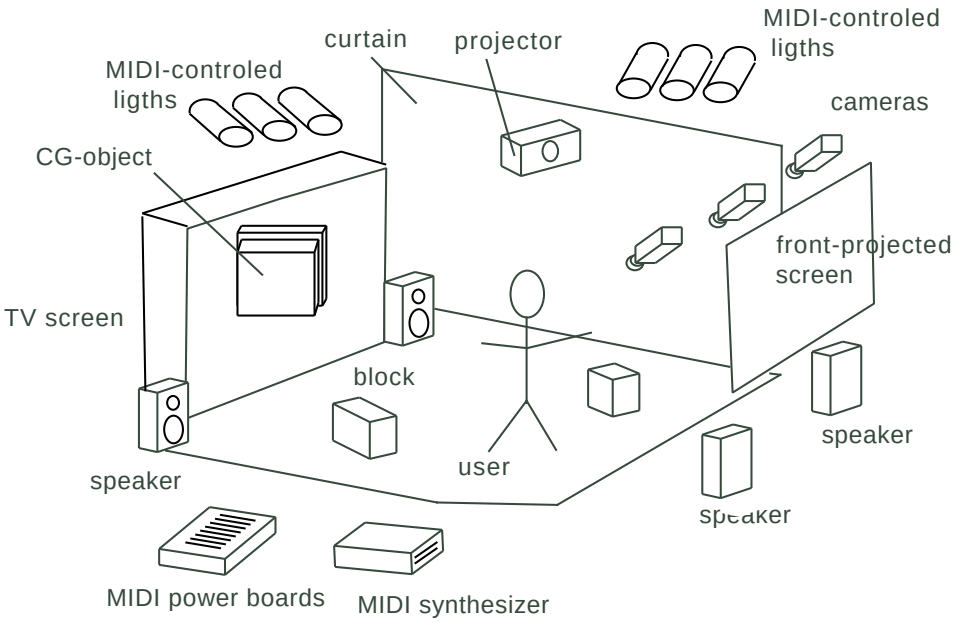


Figure 7.24 Physical setup of *"It"*.

showing pictures. The installation was first run in March of 1999, during an open house for sponsors of the MIT Media Laboratory.

Figure 7.24 shows the basic physical setup of *"It"*. It consists of two facing screens inside an enclosed, dark room. Three cameras, in a stereo vision configuration similar to one employed in *"It / I"*, monitor the space, detecting the user's presence and position. The installation only admits one user each time. Two independent sets of audio speakers are associated to each screen, reproducing the sound generated by a MIDI synthesizer. Theatrical fixtures illuminate the space, controlled by MIDI power devices.

7.4.1 Objectives and Context

We had many different goals and ideas to experiment with in *"It"*. The main objective was to create a story-driven interactive space where users could experiment some of the feelings and ideas previously developed in *"It / I"*. By revisiting the thematic of the play we also wanted to explore different treatments for the same experience — for performers and for users — in a situation where we somewhat could compare the two different experiences with the same underlying structure.

Considering that our two previous projects in interactive spaces have relied in performers, we also believed that some of our ideas about scripting were due to be tested in user-centric spaces. Also, we wanted to address the issue of working with non-performers, who require automatic characters to actively understand their actions and to provide help when confusion or inactivity is detected.

In terms of technology, we had three goals for the project *"It"*. First, we wanted a complex testbed to test the new versions of the interval script language and its run-time engine. Second, we want to develop an interactive system fully based in the layered story-character-device (SCD) architecture. Finally, there were improvements in the expressiveness and run-time parsing of the ACTSCRIPT language that we wanted to check in a real communication situation.

The *"It"* project also contains a great deal of experimentation with user-oriented interactive elements. In particular, we were seeking to examine how to make a character pro-active in a space, that is, a character that pushes the user into performing some actions and actively tries to change the user mood. Among other means, in *"It"* we investigated changes of lighting (including color changes) as a dramatic and immersive element.

7.4.2 The Story

The theme of *"It"* is the same of *"It / I"*, the entrapment of people by technology. However, in *"It"*, we want the user to be actually attracted by narrative devices, seduced to play with the machine, gradually start to feel uncomfortable about the situation, and suddenly discover that he has no way to escape. The interactive story is basically described by the following script.

"It"

an interactive installation

by Claudio Pinhanez

A large space containing two screens in the opposite walls and wood blocks on the floor. One of the screens (named as the left screen) is high enough so people can not reach a significant portion of it. When there is nobody in the space, ***It*** *(the computer character that inhabits the space) is dormant: green lights, soft machine sound in the background. When* ***I*** *(an individual user) enters the space,* ***It*** *wakes up, switching the lights on and stopping the music.* ***It*** *tries to attract and manipulate the attention of* ***I*** *by displaying beautiful images (of suns, families, angels, flowers) on the left screen, and then removing them through the right screen, as if the images have gone through the space, trying to make* ***I*** *to focus on the right screen.* ***It*** *then surprises* ***I*** *by bringing a camera-like object to the left screen. If at any moment* ***I*** *leaves the space,* ***It*** *tries to bring her back by playing loud sound and fast switching color lights, as if complaining about* ***I****'s actions. The camera on the left screen follows* ***I*** *around the space and when she stops moving, zooms in and "take a picture" (flashing the lights). Immediately after,* ***It*** *moves a TV-like object to the right screen displaying the silhouette of the picture taken by the camera. This silhouette is shown for some moments. The goal of* ***It*** *is to involve* ***I*** *in this game of taking and seeing pictures, in a "crescendo" frenzy of action. Each iteration of the taking-viewing cycle is shorter than the previous but all the pictures are shown by the TV.* ***I*** *can be encouraged to use the wood blocks to make more interesting silhouettes. After the game reaches a very fast pace,* ***It*** *looses interest in* ***I****, removes the camera*

> *and the TV from the screens and starts trying to push **I** out of the space. Initially **It** switches different sets of color lights to create an uncomfortable atmosphere but if **I** persists inside the space, **It** simulates on the screen the action of throwing blocks into **I**'s direction, together with very loud sound. When **I** leaves the space, **It** goes back to the dormant state, ready for the next victim.*

"It" uses the same visual (images, computer graphics) and sound elements as *"It / I"*, but it was designed to be a self contained, stand-alone piece that can be enjoyed by users completely unfamiliar with the play.

7.4.3 The Technology

Most of the technical structure of *"It"* is based on the technology built for *"It / I"*. The major improvements to the play are in the architecture of the control modules that fully implement the story-character-device (SCD) model; and the final version of the interval script language, that removed most of the weak structures and assumptions of the version used in *"It / I"*.

"It" uses the same computer vision, sound control, and computer graphics modules employed in the play, with the exception of minor improvements. To control the lights we employed two 4-channel MIDI-controlled TOPAZ power bricks in a daisy configuration, providing 8 independent dimmer channels. This proved to be a very simple and inexpensive solution to control light in an interactive environment, avoiding altogether the expensive dimmer controllers and light boards used in theater.

One of the reasons for building *"It"* was to experiment with the idea of implementing an interactive space using the SCD architecture. Unlike in *"It / I"*, in this work the control is clearly divided in three levels. Figure 7.25 diagrams the architecture of *"It"*. The device level contains the software modules responsible for the direct control of the computer graphics, sound, lights, and vision system. The character level implements three modules: one corresponding to the ***It*** character, one for the user (or ***I*** character), and a third that corresponds to a crew member, a light designer.

There are two reasons for singling out light control at the character level. First, unlike the other actuators such as the computer graphics modules, light is controlled both by the character ***It*** and by the top-level story module. Second, the user module changes its behavior according to the light conditions in the space. For instance, when strong, dark color lighting is used, the positional data becomes unreliable. We found it conceptually easier to have the user module sending a single query just to the light designer module, asking whether the light conditions are normal, instead of checking every dimmer of the low-level dimmer control module.

In this view, all the modules in the character level exchange middle-level information about their activity and state. The ***It*** module questions the ***I*** module directly about the position of the character: it is the user module responsibility to analyze the information coming from the vision system and to decide if there is a user in the space and where she is.

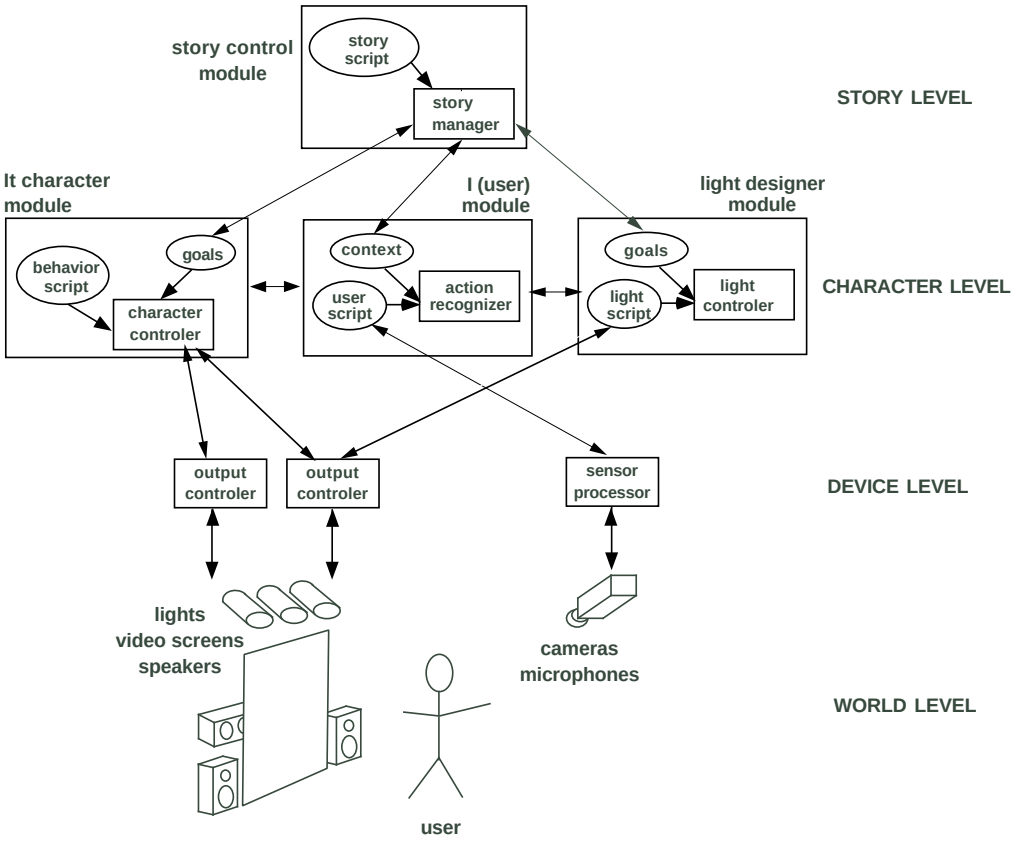


Figure 7.25 System architecture of *"It"*.

The basic distinction between the story-level module and the character and crew modules is that the latter contain a great deal of information about how to perform an action while the former is mostly concerned about when and why to do it. For example, the story module has to decide when it is the time for the ***It*** character module to attract ***I***'s attention using the images, when to play ***I*** with pictures, and when to expel him from the space. In this last situation, a request for expelling ***I*** from the space is translated by the ***It*** module into a request to the light designer module to flash the lights from red to white, continuously, and, after some seconds, in a sequence of commands to the computer graphics modules to generate explosions on the screens.

Our experience using the SCD model has been extremely positive. Although it increases the amount of communication between modules in a system, it reduces the conceptual complexity and the flexibility for making changes. For instance, it becomes trivial to experiment with different ordering for the story events, since the character level responds to the change without the need for alterations.

However, the nature of the communication between the story and the character models require a protocol where goals, actions, and intentions can be easily expressed. Conventional

procedural calls can hardly satisfy these conditions, making it important the use of a high level language for actions such as the one we employed, ACTSCRIPT. Not surprisingly, we found the ACTSCRIPT much more appropriate for the communication at this level than to exchange data with the device modules. We could even observe a synergy between the two ideas, where the SCD architecture created the need of expressive communications, while the ACTSCRIPT made it easy for the story module to control the experience.

7.4.4 The Scripting Process

In *"It"* we employed the version of interval scripts described in detail in chapter 5. The major differences between that version and the one used in *"It / I"* are, as pointed above, threefold. First, in *"It"* we eliminated conditions based on change, allowing a much clearer handling of the control structure. Second, when statements in the current version are translated directly into constraints expressing occurrence relationships between the condition and the action to be performed. And third, the current version allows the declaration of nested intervals and automatically establishes containing relations between the declared intervals. In particular, when statements declared inside an interval are only effective when the interval is happening.

The result is a language where the explicit declaration of temporal constraints is almost never necessary. Instead, the designer writes the script file considering inclusion relationships, triggering conditions, sequences, cycles, and timers. With such structures it is easy to describe the basic structure of an interactive story and to refine it by explicitly incorporating constraints to describe specific situations that must or must not occur, independently of the story sequence.

In other words, in *"It"* we could experiment with a scripting language closer to what we consider the ideal paradigm where both procedural and constraint-based elements can be used seamlessly. In our model, both are ultimately translated into the temporal constraint structure of a PNF-network, allowing the run-time engine to reactively look for globally consistent states of the intervals describing the script.

Table 7.7 The composition of the interval scripts for the different modules in *"It"*.

	number of nodes	total number of constraints	declared constraints	when statements	sequences	cycles	timers
story	59	101	4	23	0	0	6
it character	224	355	4	62	5	7	32
ic character	8	8	2	1	0	0	0
light designer	94	133	0	33	0	2	5

In table 7.7 we summarize the composition of the interval scripts used for the different modules of *"It"*. First, notice that although the number of constraints in the network is quite large, the number of explicitly declared constraints is quite low. This demonstrates the hiding of constraints into language structures (such as when statements and nested intervals) as discussed above.

Table 7.7 also shows that interval scripts can be used to construct and control large and complex collections of actions. For example, the ***It*** character module employs 224 intervals, 62 when statements, and 32 timers, for a total of more than 350 constraints. During run-time, we apply PNF propagation on this network at the rate of 20 Hz without any visible impact on the performance of the *SGI R10000* machine used in the installation. In fact, all four modules run simultaneously in one machine, consuming at most 15% of the CPU time, including here the communication overhead.

In the current state of the installation *"It"* there is no complex case of action recognition, as shown by the simplicity of the interval script for the ***I*** character. However, as discussed in chapter 4, we have been using the *"It"* structure in our experiments in action recognition, and future extensions of the installation are sought to contain more complex actions for the user, including the manipulation of the wood blocks in situations similar to the ones depicted in *"It / I"*.

The most major shortcoming of the current version of the interval scripts, based on our experience with *"It"*, is the lack of a better interface for the designer of the space. Although for the implementation we have implemented a graphical interactive display for the monitoring of run-time execution, we still rely on writing long script files to describe the story and the behavior of the characters. We envision as the next step of the development of interval scripts the implementation of a graphical user interface or a management system for the script files that could provide, for instance, the hiding of nesting intervals and the visual manipulation of temporal constraints.

7.4.5 The Experience

We concluded the present installation of *"It"* in March 1999, in a laboratory space at the MIT Media Laboratory. Since then we have had dozens of users experiencing the feeling of being trapped by ***It***. The current site is far from ideal, since one of the walls is made of glass and there is a considerable amount of surrounding noise. Nevertheless, we are very pleased with the reaction of the users and with the easiness that the interaction is understood. In particular, we had some very interesting runs with children.

Like in *"It / I"*, sound proved to be a key component to provoke reaction in the users. However, in *"It"* we introduced a great deal of intention and expression by changing lighting conditions. We designed the light in *"It"* using three different colors, a white wash for the normal interaction, a green wash when the creature is dormant, and a red blinking light used to warn the user of inappropriate behavior (such as living the story in the middle). Combined with sound, we witnessed an impressive impact of the lighting changes on people's feelings.

Our current goal is to keep developing *"It"* by observing people's reactions to the piece and aiming to a future installation in a museum or gallery where a wider audience can be reached.

7.5 Summary

In this chapter we demonstrated the effectiveness of the use of the methods and ideas described in this thesis in the context of real, complex interactive spaces. In the first project described, the *Intelligent Studio*, we were able to apply the *action frames* scheme for human action representation (detailed in chapter 2) in a system to automatically control cameras in a TV studio. Based on the information contained in the script of a cooking show we could guarantee that a 3D model of the scene was always reasonably correct. Good results were obtained in three different sequences in two different scenarios.

The following three projects, *"SingSong"*, *"It / I"*, and *"It"* demonstrated the expressive power and appropriateness of our idea of interval scripts, as can be seen from the examples collected in this chapter. The three projects also brought evidence that PNF propagation is a sound and efficient foundation for real-time temporal reasoning. *"SingSong"*, as our first experiment in *computer theater*, showed that the concept of automatic computer-actors in performance was feasible. Later, *"It / I"* proved that it is already possible to perform long and complex stories with a computer partner. *"It / I"* and *"It"* furthered the development of ACTSCRIPT as a communication language and assured us of the power of a human action-based language for inter-module communication. Finally, the experience of building *"It"* based on the story-character-device architecture was very rewarding, showing that there are advantages in separating the character level from the story control.

Finally, we would like to say that computerized characters and plays have been our personal path to explore the frontier between real and virtual worlds that is increasingly blurred by technology. We have consciously explored the idea as theme of our work as part of the paradox of fusing machine and creature, computer and actor. *"SingSong"*, *"It / I"*, and *"It"* are, however, still simple, first experiments about designing, scripting, rehearsing, and performing with characters — and "actors" — which are born virtual, truly inhabit a non-physical world, interact with human performers and with computer entities, and tragically disappear with the click of a switch.

8. Conclusion

The fundamental observation that guided this thesis is that the temporal structure of actions can often be complex, involving multiple parallel threads that can not be adequately represented by finite-state machines. This is especially true in the domain of interactive spaces where there are multiple sources of agency for the actions. Therefore, the likelihood of concomitant and at the same time loosely synchronized action is very high, making very large the potential number of individual states of the system.

The second reason for complex temporal structures in interactive spaces is precisely their physicality. Unlike in desktop interaction, both the users' physical movements and the devices' actuation and control take time and therefore it is necessary to move from a basically discrete space of events (or sets of events) into a much denser space composed of time intervals (and configurations of intervals).

Such shift in the complexity of the temporal space must be matched by an increase in sophistication and expressiveness of the temporal structures. This thesis investigates feasible alternatives to finite-state machines in the context of real-time recognition of actions and interaction control.

Our revision of classic AI methods such as conceptualizations and IA-networks has been informed by our work on creating real interactive spaces. By doing so, we avoided the pitfall of elegant but exponential theories of time or action. Besides being formally well rooted, our work emphasized simpler but efficient formulations that preserve some desired properties of the original theory. For instance, PNF-networks are certainly less expressive than IA-networks but the 3-valued nodes eliminated completely a major source of exponential growth, that is, the arithmetic of disjunct intervals. Nevertheless, we preserved the ability of determining whether nodes are happening or not, which is the fundamental information for a control system of an interactive space.

At the same time, our work also explored new domains for interactive applications and, in particular, performance spaces. Both TV studios and theatrical stages have not been regarded before as spaces for experiments in action recognition and representation. In fact, we took advantage of some simplifications brought by performances in interactive spaces, the most

prominent of them being the availability of some type of script that describes the actions which are likely to occur.

However, our interest in interactive, story-driven space transcends their adequacy for our scientific experiments. Particularly in the case of computer theater, the work described in this thesis is a genuine effort to look for new media for art, entertainment, learning, and social life. We believe that as important as developing tools that accommodate the needs of these new media is to advance the understanding of their language and the impact that they have on people.

In the rest of this chapter we review the main results and contributions of this thesis. We conclude by examining future directions for our work.

8.1 Results and Contributions

The first contribution of our thesis is the **action frames** formalism to represent human action. In many ways, the more important aspect of this part of our work is the "redemption" of Schank's conceptualizations [149]. Although having several limitations when applied in the original domain (natural language processing), our approach proved to be well suited to the purpose of providing a framework for action recognition. In particular, we have augmented the original proposal by expressing the temporal structure of the action as constraints in an IA-network where the nodes are the action's basic units.

We have shown that it is possible to implement simple inference systems (in our case, based on Rieger's work [144]) that analyze an action frame into its simpler component sub-actions and identify states of the world that, in theory, can be empirically tested. For instance, in one of our examples from the cooking show domain, we were able to infer, from the description of an action of *"pounding chicken"*, that the hands of the chef would be in the proximity of the chopping-board.

Our representation of the temporal structure of an action is based on mapping all the component units of the action into nodes of an interval algebra network and then determining the temporal constraints that exist among these nodes. For each particular sensing situation, we map the state of perceptual routines into new nodes of the network (constrained by sets of temporal constraints). During run-time, the interval of activation of the sensor nodes can be propagated to the other nodes and we have shown in examples that this is sufficient to allow the recognition of the action provided that the constraints and perceptual routines are adequate.

However, constraint propagation in IA-networks is basically exponential. We circumvented this problem by proposing a new class of temporal networks, **PNF-networks**, where each node assumes one of the three values: *past*, *now*, or *future*. These networks are built so that their solutions are conservative approximations of the solutions of the original IA-network. The main advantage of PNF-networks is that the minimal domain of the network can be conservatively approximated using *arc-consistency* [90], which is linear in the number of constraints. The approximation is shown to have some nice properties and its application in practice has been extremely successful.

To recognize actions whose temporal structure is represented by a PNF-network (a projection of the original IA-network), we developed the **PNF propagation** method that combines the current information coming from perceptual routines with a compact representation of the past occurrences of the action units. In the core of the method we employ the arc-consistency algorithm to eliminate all the values from the PNF-state of the nodes that can not be consistent with the rest of the network. This is a global procedure that is independent of node ordering and that reaches a solution for the whole network.

In the thesis we have examined the PNF propagation in different conditions, first assuming that all the sensors are always correct. In such cases it is possible to study the impact of different constraints and sensors on the ability of a recognition system to detect an action. We then developed a formulation that enables handling of sensor errors by keeping track of all different threads of events that have solutions.

In this case we can not assure linear time or memory requirements except by arbitrarily imposing a limit on how many threads are kept. However, when applied to practical problems of action recognition, we have observed that the number of consistent threads tends to be stable for a given action and that the recognition of an action with a larger number of units does not necessarily require the keeping of a larger number of threads. This can be justified by the fact that the set of consistent IA-networks is not dense in the set of IA-networks, particularly for networks with more than 15 nodes [82].

Another contribution of this thesis is the **interval script** paradigm for the control of interaction. First we observed that, since interaction is a special case of multi-agent action, the same sort of complex temporal structures should be expected in the description of an interaction. In fact, it is even worse if we consider the fact that actuators often take time to respond, if they respond at all.

According to our proposal, an interval script describes, for each action and state, how to determine its current state and how to start and stop (in the case of actions). Such descriptions can be specified by considering previously defined actions, thereby providing a nice framework for system development. During run-time, the interval script engine determines the current state of all actions of the script and, using a variation of the PNF propagation method, predicts and selects a consistent, “desired” state for all actions. In our implementation, we always select states that minimize the amount of change in the sensed world. Given the current and the desired state of each action, the interval script run-time engine can correspondingly determine whether the action should start, stop, or remain in its current state. A feature of the interval script paradigm is that it does not assume that an action starts because it was commanded to do so.

On the other hand, a limitation of interval scripts is the lack of mechanisms to explicitly construct plans. Since the PNF propagation algorithm does not distinguish between actions happening in different times in the future, it is impossible to automatically reason about steps to achieve a certain goal. Notice, however, that this does not impede the system to follow pre-defined, scripted sequences with quite complex patterns of activation and reaction, departing

from the traditional assumption of sequential actions employed in most planner-based interaction control systems.

We have employed interval scripts to develop large-scale interactive spaces running in performance situations. Our own experience of using the paradigm has been extremely rewarding and we believe that the implementation of some of our interactive spaces would be virtually a nightmare if we had not used interval scripts, particularly in the case of the computer theater play *"It / I"*.

Two other contributions described in this thesis are the **story-device-character architecture** (SCD) for story-driven interactive spaces; and the systematization of the ideas concerning **computer theater**. We also believe that the interactive systems we built have also answered some questions in the research community. In particular, the **Intelligent Studio** project showed that high-level linguistic information not only can be incorporated into a vision system (through adequate representation and inference) but also that its presence can significantly enhance the capabilities of real-world system.

With ***"It / I"*** and, to some extent with ***"SingSong",*** we demonstrated that computers can control automatic actors in theater plays. Moreover, the experience with ***"It"*** has shown that **immersive stages** can be quite appealing.

8.2 Directions to Explore

Having examined the multiple facets of our contributions, it is important to look at different opportunities for research that have been left unexplored by our work.

In terms of action representation, it is certainly arguable whether the action frames formalism is appropriate for the description of actions. The reduced number of primitives, although very convenient when designing inference systems, sometimes produces long and difficult to read descriptions of an action. Similarly, we have not investigated the translation of natural language sentences into action frames. Specifying actions directly in natural language would certainly facilitate enormously the design of action recognition systems. That is probably possible in the case of action frames given the prior work on translation of conceptualizations [150].

Notice that in this thesis we did not discuss automatic mechanisms of inferring the temporal structure from the action frame representation of an action but only the inference of the action's component sub-actions. We believe that the implementation of such an inference system is not only desirable but also possible, especially considering that in IA-networks the temporal constraints are disjunctions of primitive relationships.

The main question remaining from our analysis of IA-networks is under which conditions the result of the arc-consistency algorithm is exactly equal to the PNF restriction of a component domain. That is, what the structure should the PNF-networks have such that it is always true that $R(W) = AC(W)$. In chapter 3 we have presented some of these conditions in very specific scenarios but it would be very useful to have an algorithm that could evaluate whether arc-consistency can determine all the solutions for a specific PNF-network.

Many interesting directions arise from our idea of using PNF propagation for action recognition. As pointed in chapter 4, it is important to investigate how to incorporate confidence measures and probabilistic reasoning into the framework. Especially in the case of visual recognition of actions, there is a clear trend in the computer vision community towards probabilistic methods. In that chapter we sketched a method to include confidence measures into PNF propagation but we have not tested yet those ideas. Also, it seems promising to explore the possibility of including probabilistic reasoning inside the temporal constraint propagation process, although going in this direction might remove one of the chief features of PNF propagation, that is, the detection and removal of contradictory configurations of sensor values.

We see many different ways to extend our interval script language. First, as mentioned in chapter 5, we should investigate the appropriateness of a graphical user interface as the front-end of the system. Another approach, especially for high-level scripting, would be to consider the translation of natural language descriptions of interaction and story directly into interval scripts. There is also room for the design of debugging tools for multi-module, interval script-based interactive spaces.

A problem we have noticed while working with interval scripts is that it is hard to find the cause of inconsistencies in a PNF-network. An inconsistency can be generated after any number of steps of the arc-consistency cycle, making it impossible to find the original violation that triggered the inconsistency. This is an intrinsic component of the constraint propagation paradigm, in some ways corresponding to the price to be paid for the ability to find globally consistent solutions. In the interval script framework, however, it is possible to better localize and, more importantly, isolate sources of inconsistency by using the nested structure of the intervals.

One idea is to implement the PNF-network of an interval script as a set of nested PNF-networks where each sub-network corresponds to the intervals that are defined inside an interval. Instead of running the arc-consistency algorithm in the whole network, we start by applying it to the sub-networks, following the temporal order implicit in their definition. If a sub-network is inconsistent, it is possible to avoid the spread of the problem by simply assigning *PNF* to all the nodes of the sub-network. We have not yet experimented with this method in practice but we believe it can alleviate considerably the occurrence of large-scale propagation of inconsistencies.

Another approach we want to explore in the future is the use of the error recovery methods described in chapter 4 in the run-time engine of interval scripts. In other words, we believe the reliability of the execution of an interval script can be improved if we explicitly model the possible failure of sensors and track the multiple threads of sensor events as we did for action recognition. Also, the method to compute likelihood of component domains presented in chapter 4 can be used to assure smoother runs of interval scripts.

A much more complex problem is the lack of means for reasoning about plans and goals in the interval script structure. As discussed in chapter 5, this relates to the fact that PNF propagation lumps together all the intervals to happen after the current instant of time. We would like to

incorporate to the language mechanisms that would allow it to check for globally consistent solutions not only for the instant of time immediately after the current but for some reasonable amount of time afterwards. To achieve this, it is necessary to include the concept of action length into the language and to investigate fast algorithms that approximate solutions to the network.

Finally, it is necessary to make the interval script tools available to a larger audience, so it becomes possible to evaluate if designers of interactive spaces and systems can use the interval script paradigm to develop real world applications.

In terms of application and domains, we believe that there is an incredible potential yet to be realized in computer theater. Although our work has been quite effective in raising the attention and interest in the use computers in theater among the art and technology community, our next step should involve computer theater productions that reach the theater community and the general public. Our current plan is to professionally produce the *"Waltz #6"* project described in chapter 6 in the summer of 2000. We are also interested in developing further the concept of hyper-actors, that is, electronic, controllable expansions of the actor's body, in particular using wearable computers [123] and affective computing [127] technology.

We also see a great deal of potential in our concept of immersive stages. The experience with *"It"*, although extremely rewarding, could not demonstrate conclusively the public acceptance of the idea of an interactive space designed both for performance and for private enjoyment. The project *"Waltz #6"* includes in its concept, as shown in chapter 6, the conversion of the performer's space into an interactive art installation when performance is not happening.

Finally, we want to explore applications for interactive spaces beyond the art and entertainment domain. We see the technology discussed in this thesis being ported first to other performance spaces: auditoriums, classrooms, showrooms, live TV shows, sport arenas, maybe even churches. Beyond that, action recognition and interaction scripting can be applied in more generic interactive spaces, such as meeting rooms, stores, customer service areas, libraries, parks and other leisure areas, airport terminals and train stations, and hospitals. And, of course, at home.

8.3 A Final Word

Much of the work presented in this thesis can be characterized as an effort to recycle the artificial intelligence research done in the past. After the hype of 70s and the downfall of the 90s, we believe it is time to re-evaluate the proposals and ideas introduced by AI in the light of the changes brought by the widespread of personal computers and the emergence of the Internet.

This does not mean that we should all go back to develop expert systems. In our view, the correct approach is to re-read the vast AI literature with new applications and approaches in mind. For instance, the amount of personal information available today in digital form (through consumer profiles, personal schedulers, computerized documents, etc.) can not be compared to the AI scenario of 20 years ago, making feasible to trade off deep reasoning by extra knowledge (see, for example [93, 95]).

Our work with conceptualizations follows this trend of revisiting AI. By considering Schank's paradigm not as a representation for communicative thought but instead as a formalism for action, we were able to employ the theory effectively in a real interactive system. Similarly, we approached temporal representation theory thinking not about reasoning but in real-time recognition. That allowed us to abstract the interval concept into a three-valued structure that could be used in real-time and still provide the information basically needed in the detection problem.

A particular area where we see a potentially huge impact of artificial intelligence methods and particularly, action representation and recognition, is human-computer interfaces. Current computers treat us as if we were machines: no past, no individuality, no intentions, no emotions, and no culture. By providing computers with action representation mechanisms, it might be possible to make machines that can consider the context of their actions and therefore provide much clear and responsive support for the user's activities.

Such influence in the ways human and computer interact can be even greater if we consider that more and more we will interact with computers outside the desktop paradigm. If ubiquitous computing [179], wearable computers [123], and tangible interfaces [64] become a reality, we are creating scenarios where the insertion of computer happens in more precise contexts and in the presence of more information. At the same time, a computer's actions, if not appropriately controlled, can become as inconvenient as today's indiscriminate ringing of cellular phones. We believe that such technologies can have a place in the market only if the interface recognizes what the user is doing and in which context she is, reacting appropriately.

Finally, we want to build the *Holodeck* (the imaginary space portrayed in the *"Start Trek"* TV series where holography immerses people in interactive stories). However, we believe that we do not need life-size holography or tele-transported matter but that immersion is less a product of high-quality displays or physical stimuli than a consequence of responsive characters and good stories. Building the *Holodeck* is a matter of creating believable worlds where the body is transported by the mind.

I dream of being able to choose the dreams I will dream.

Appendices

A.Script and Inferences in the Cooking Show Example

Action Frame Script

```
;;;=====================================
;; class definitions

  (chef sc::human-being)
  (bread-crumbs sc::powdered-substance)
  (parsley sc::powdered-substance)
  (paprika sc::powdered-substance)
  (basil sc::powdered-substance)
  (bowl sc::handleable-object)
  (chicken sc::handleable-object)
  (plastic-bag sc::handleable-object)
  (meat-mallet sc::handleable-object)
  (chopping-board sc::static-object)

  (hands sc::human-hands)
  (mouth sc::human-mouth)

  (public sc::virtual-human-being)
  (sound sc::expression-medium)
  (gestures sc::expression-medium)
  (text1 sc::idea)
  (text2 sc::idea)
  (how-to-pound sc::idea)

  (:side sc::camera-object)
  (:center sc::camera-object)
  (:upper sc::camera-object)

  (ready-dish sc::handleable-object)
  (cup sc::handleable-object)
  (tomato-bowl sc::handleable-object)
  (ham-bowl sc::handleable-object)

  (table sc::static-object)
  (oven sc::static-object)

;;;=====================================
;; list of actions

;; "chef greets the public (to center camera)"
(mtrans
  (time (group (begin 1) (end 62)))
  (actor chef) (to public)
  (object text1)
  (instrument
    (speak (actor chef) (object sound)
      (to (direction :center))
      (from (location
        (description
          (object mouth)
          (concept
          (have
            (object mouth)
            (attribute
              (part chef)))))))))

;; "chef turns to side camera"
(ptrans
  (time (group (begin 63) (end 67)))
  (actor chef) (object chef)
  (result
    (change (object chef)
      (to (direction :side)))))

;; "chef talks about today's recipe (to side camera)"
(mtrans
  (time (group (begin 68) (end 135)))
  (actor chef) (to public)
  (object text2)
  (instrument
    (speak (actor chef)
      (object sound)
      (to (direction :side))
      (from (location
        (description
          (object mouth)
          (concept
          (have
            (object mouth)
            (attribute
              (part chef)))))))))
```

```
;; "chef turns back to center camera"
(ptrans
  (time (group (begin 136) (end 138)))
  (actor chef) (object chef)
  (result
    (change (object chef)
      (to (direction :center)))))

;; "chef mixes bread-crumbs, parsley, paprika, and basil in a
bowl"
(do-something
  (time (group (begin 139) (end 275)))
  (actor chef)
  (result
    (change
      (object
        (group bread-crumbs parsley
          paprika basil))
      (to (phys-cont
          (group bread-crumbs parsley
            paprika basil))))))

;; "chef wraps chicken with a plastic bag"
(do-something
  (time (group (begin 276) (end 336)))
  (actor chef)
  (result
    (change (object chicken)
      (to (group (contain plastic-bag)
          (phys-cont plastic-bag))))))

;; "chef shows how to pound the chicken (on the chopping-
board)"
(mtrans
  (time (group (begin 337) (end 380)))
  (actor chef)
  (object how-to-pound)
  (to public)
  (instrument (group
    (speak (actor chef)
      (object sound)
      (from (location
        (description (object mouth)
          (concept
          (have
            (object mouth)
            (attribute (part chef))))))
      (to (direction :center)))
    (speak (actor chef)
      (object gestures)
      (from (location
        (description (object hands)
          (concept
          (have
            (object hands)
            (group (attribute (part chef)
          (proximity chopping-board))))))
      (to (direction :center)))))

;; ""chef pounds the chicken with a meat-mallet (on the
chopping-board)"
(propel
  (time (group (begin 381) (end 400)))
  (aider
    (group back-and-forth fast strong))
  (actor chef)
  (object meat-mallet)
  (from (location
    (description (object ?in-contact)
      (concept (have (object ?in-contact)
        (attribute
          (phys-cont chicken)))))))
  (from (location
    (description (object ?not-in-contact)
      (concept (have (tense negation)
        (object ?not-in-contact)
        (attribute
          (phys-cont chicken)))))))
  (result
    (change (object
      (description (object chicken)
        (concept (have (object chicken)
          (attribute
          (phys-cont chopping-board))))))
      (from (flatness ?X))
      (to (flatness (greater ?X))))))
```

List of All Inferences for Each Action

```
;;;=======================================
;; "chef greets the public (to center camera)"
(mtrans
  (time (group (begin 1) (end 62)))
  (actor chef) (to public) (object text1)
  (instrument
    (speak (actor chef) (object sound)
      (to (direction :center))
      (from (location
        (description (object mouth)
          (concept
          (have
            (object mouth)
            (attribute
              (part chef)))))))))

;; inferred actions and attributes
(have
  (time (group (begin 1) (end 62)))
  (object mouth)
  (attribute (part chef)))

;;;=======================================
;; "chef turns to side camera"
(ptrans
  (time (group (begin 63) (end 67)))
  (actor chef) (object chef)
  (result
    (change (object chef)
      (to (direction :side)))))

;; inferred actions and attributes
(propel
  (time (group (begin 63) (end 67)))
  (actor chef) (object chef))

(move
    (time (group (begin 63) (end 67)))
    (actor chef) (object chef)))

;;;=======================================
;; "chef talks about today's recipe (to side camera)"
(mtrans
  (time (group (begin 68) (end 135)))
  (actor chef) (to public)
  (object text2)
  (instrument
    (speak (actor chef)
      (object sound)
      (to (direction :side))
```

```
      (from (location
        (description (object mouth)
          (concept
          (have
            (object mouth)
            (attribute
              (part chef)))))))))

;; inferred actions and attributes
(have
  (time (group (begin 68) (end 135)))
  (object mouth)
  (attribute (part chef))))

;;;=======================================
;; "chef turns back to center camera"
(ptrans
  (time (group (begin 136) (end 138)))
  (actor chef) (object chef)
  (result
    (change (object chef)
      (to (direction :center)))))

;; inferred actions and attributes
(propel
  (time (group (begin 136) (end 138)))
  (actor chef) (object chef))

(move
    (time (group (begin 136) (end 138)))
    (actor chef) (object chef)))

;;;=======================================
;; "chef mixes bread-crumbs, parsley, paprika, and basil in a
bowl"
(do-something
  (time (group (begin 139) (end 275)))
  (actor chef)
  (result
    (change
      (object
        (group bread-crumbs parsley
          paprika basil))
      (to (phys-cont
          (group bread-crumbs parsley
            paprika basil))))))

;; inferred actions
(have
  (time (group (begin 139) (end 275)))
  (object (group bread-crumbs parsley
            paprika basil))
  (attribute (contain bowl)))

(ptrans
  (time (group (begin 139) (end 275)))
  (actor chef)
  (object (group bread-crumbs parsley
            paprika basil))
  (to (location x70764)))

(have
  (time (group (begin 139) (end 275)))
  (object x70764)
  (attribute (phys-cont
      (group bread-crumbs parsley paprika
        basil))))

(propel
  (time (group (begin 139) (end 275)))
  (actor chef)
  (object (group bread-crumbs parsley
            paprika basil))
  (to (location x70764)))

(grasp
  (time (group (begin 139) (end 275)))
  (actor chef) (object bowl)
  (to hands))

(have
  (time (group (begin 139) (end 275)))
  (object hands)
  (attribute (part chef)))

(move
  (time (group (begin 139) (end 275)))
  (actor chef) (object hands)
  (to (location x70764)))

(have
  (time (group (begin 139) (end 275)))
  (object   (group   bread-crumbs   parsley
paprika basil))
  (attribute (phys-cont hands)))

(have
  (time (group (begin 139) (end 275)))
  (object (group bread-crumbs parsley
            paprika basil))
  (attribute (phys-cont x70764)))

(have
  (time (group (begin 139) (end 275)))
  (object bowl)
  (attribute (phys-cont hands)))

(have
  (time (group (begin 139) (end 275)))
  (object hands)
  (attribute
    (phys-cont (group bread-crumbs parsley
                 paprika basil))))

(have
  (time (group (begin 139) (end 275)))
  (object hands)
  (attribute (phys-cont bowl)))

(have
  (time (group (begin 139) (end 275)))
  (object x70764)
  (attribute
    (proximity (group bread-crumbs parsley
                 paprika basil))))

(have
  (time (group (begin 139) (end 275)))
  (object (group bread-crumbs parsley
            paprika basil))
  (attribute (proximity hands)))

(have
  (time (group (begin 139) (end 275)))
  (object (group bread-crumbs parsley
            paprika basil))
  (attribute (proximity x70764)))

(have
  (time (group (begin 139) (end 275)))
  (object bowl)
  (attribute (proximity hands)))

(have
  (time (group (begin 139) (end 275)))
  (object hands)
```

```
  (attribute
    (proximity (group bread-crumbs parsley
                 paprika basil))))

(have
  (time (group (begin 139) (end 275)))
  (object hands)
  (attribute (proximity bowl)))

(have
  (time (group (begin 139) (end 275)))
  (object (group bread-crumbs parsley
           paprika basil))
  (attribute
    (proximity (group bread-crumbs parsley
                 paprika basil))))

(have
  (time (group (begin 139) (end 275)))
  (object (group bread-crumbs parsley
           paprika basil))
  (attribute (proximity bowl)))

(have
  (time (group (begin 139) (end 275)))
  (object bowl)
  (attribute
  (proximity (group bread-crumbs parsley
               paprika basil)))))

;;;=======================================
;; "chef wraps chicken with a plastic bag"
(do-something
  (time (group (begin 276) (end 336)))
  (actor chef)
  (result
    (change (object chicken)
      (to (group (contain plastic-bag)
          (phys-cont plastic-bag))))))

(ptrans
  (time (group (begin 276) (end 336)))
  (actor chef) (object chicken)
  (to (location x70765)))

(have
  (time (group (begin 276) (end 336)))
  (object x70765)
  (attribute (phys-cont plastic-bag)))

(propel
  (time (group (begin 276) (end 336)))
  (actor chef) (object chicken)
  (to (location x70765)))

(grasp
  (time (group (begin 276) (end 336)))
  (actor chef) (object chicken)
  (to hands))

(have
  (time (group (begin 276) (end 336)))
  (object hands)
  (attribute (part chef)))

(move
  (time (group (begin 276) (end 336)))
  (actor chef) (object hands)
  (to (location x70765)))

(have
  (time (group (begin 276) (end 336)))
  (object hands)
  (attribute (phys-cont chicken)))

(have
  (time (group (begin 276) (end 336)))
  (object chicken)
  (attribute (phys-cont hands)))

(have
  (time (group (begin 276) (end 336)))
  (object chicken)
  (attribute (phys-cont x70765)))

(have
  (time (group (begin 276) (end 336)))
  (object plastic-bag)
  (attribute (phys-cont x70765)))

(have
  (time (group (begin 276) (end 336)))
  (object x70765)
  (attribute (phys-cont chicken)))

(have
  (time (group (begin 276) (end 336)))
  (object x70765)
  (attribute (proximity plastic-bag)))

(have
  (time (group (begin 276) (end 336)))
  (object hands)
  (attribute (proximity chicken)))

(have
  (time (group (begin 276) (end 336)))
  (object chicken)
  (attribute (proximity hands)))

(have
  (time (group (begin 276) (end 336)))
  (object chicken)
  (attribute (proximity x70765)))

(have
  (time (group (begin 276) (end 336)))
  (object plastic-bag)
  (attribute (proximity x70765)))

(have
  (time (group (begin 276) (end 336)))
  (object x70765)
  (attribute (proximity chicken)))

(have
  (time (group (begin 276) (end 336)))
  (object plastic-bag)
  (attribute (proximity chicken)))

(have
  (time (group (begin 276) (end 336)))
  (object plastic-bag)
  (attribute (proximity hands)))

(have
  (time (group (begin 276) (end 336)))
  (object x70765)
  (attribute (proximity hands)))

(have
  (time (group (begin 276) (end 336)))
  (object chicken)
  (attribute (proximity plastic-bag)))

(have
```

```
  (time (group (begin 276) (end 336)))
  (object hands)
  (attribute (proximity plastic-bag)))

(have
  (time (group (begin 276) (end 336)))
  (object hands)
  (attribute (proximity x70765))))

;;;=====================================
;; "chef shows how to pound the chicken (on the chopping-
board)"
(mtrans
  (time (group (begin 337) (end 380)))
  (actor chef)
  (object how-to-pound)
  (to public)
  (instrument (group
    (speak (actor chef)
      (object sound)
      (from (location
        (description (object mouth)
          (concept
          (have
            (object mouth)
            (attribute (part chef))))))
      (to (direction :center)))
    (speak (actor chef)
      (object gestures)
      (from (location
        (description (object hands)
          (concept
          (have
            (object hands)
            (group (attribute (part chef)
              (proximity
                chopping-board))))))
      (to (direction :center)))))

(have
  (time (group (begin 337) (end 380)))
  (object mouth)
  (attribute (part chef)))

(have
  (time (group (begin 337) (end 380)))
  (object hands)
  (attribute
    (group (part chef)
      (proximity chopping-board))))

(have
  (time (group (begin 337) (end 380)))
  (object chopping-board)
  (attribute (proximity hands)))

(have
  (time (group (begin 337) (end 380)))
  (object hands)
  (attribute (proximity chopping-board))))

;;;=====================================
;; ""chef pounds the chicken with a meat-mallet (on the
chopping board)"
(propel
  (time (group (begin 381) (end 400)))
  (aider (group back-and-forth fast strong))
  (actor chef)
  (object meat-mallet)
  (from (location
    (description (object ?in-contact)
      (concept (have (object ?in-contact)
        (attribute
          (phys-cont chicken)))))))
  (to (location
    (description (object ?not-in-contact)
      (concept (have (tense negation)
        (object ?not-in-contact)
        (attribute
          (phys-cont chicken)))))))
  (result
    (change (object
      (description (object chicken)
        (concept (have (object chicken)
          (attribute
          (phys-cont chopping-board))))))
      (from (flatness ?X))
      (to (flatness (greater ?X))))))

(have
  (time (group (begin 381) (end 400)))
  (object u65888)
  (attribute (phys-cont chicken)))

(have
  (time (group (begin 381) (end 400)))
  (tense negation)
  (object m65889)
  (attribute (phys-cont chicken)))

(have
  (time (group (begin 381) (end 400)))
  (object chicken)
  (attribute (phys-cont chopping-board)))

(grasp
  (time (group (begin 381) (end 400)))
  (aider
    (group back-and-forth fast strong))
  (actor chef)
  (object meat-mallet)
  (to hands))

(have
  (time (group (begin 381) (end 400)))
  (object hands)
  (attribute (part chef)))

(move
  (time (group (begin 381) (end 400)))
  (aider
    (group back-and-forth fast strong))
  (actor chef) (object hands)
  (to (location m65889))
  (from (location u65888)))

(have
  (time (group (begin 381) (end 400)))
  (object hands)
  (attribute (phys-cont meat-mallet)))

(have
  (time (group (begin 381) (end 400)))
  (object meat-mallet)
  (attribute (phys-cont u65888)))

(have
  (time (group (begin 381) (end 400)))
  (object meat-mallet)
  (attribute (phys-cont m65889)))

(have
  (time (group (begin 381) (end 400)))
  (object meat-mallet)
  (attribute (phys-cont hands)))
```

```
(have
  (time (group (begin 381) (end 400)))
  (object chicken)
  (attribute (phys-cont u65888)))

(have
  (time (group (begin 381) (end 400)))
  (object chopping-board)
  (attribute (phys-cont chicken)))

(have
  (time (group (begin 381) (end 400)))
  (object u65888)
  (attribute (phys-cont meat-mallet)))

(have
  (time (group (begin 381) (end 400)))
  (object m65889)
  (attribute (phys-cont meat-mallet)))

(have
  (time (group (begin 381) (end 400)))
  (object u65888)
  (attribute (proximity chicken)))

(have
  (time (group (begin 381) (end 400)))
  (object chicken)
  (attribute (proximity chopping-board)))

(have
  (time (group (begin 381) (end 400)))
  (object hands)
  (attribute
    (proximity meat-mallet)))

(have
  (time (group (begin 381) (end 400)))
  (object meat-mallet)
  (attribute (proximity u65888)))

(have
  (time (group (begin 381) (end 400)))
  (object meat-mallet)
  (attribute (proximity m65889)))

(have
  (time (group (begin 381) (end 400)))
  (object meat-mallet)
  (attribute (proximity hands)))

(have
  (time (group (begin 381) (end 400)))
  (object chicken)
  (attribute (proximity u65888)))

(have
  (time (group (begin 381) (end 400)))
  (object chopping-board)
  (attribute (proximity chicken)))

(have
  (time (group (begin 381) (end 400)))
  (object u65888)
  (attribute (proximity meat-mallet)))

(have
  (time (group (begin 381) (end 400)))
  (object m65889)
  (attribute (proximity meat-mallet)))

(have
  (time (group (begin 381) (end 400)))
  (object chicken)
  (attribute (proximity m65889)))

(have
  (time (group (begin 381) (end 400)))
  (object chicken)
  (attribute (proximity meat-mallet)))

(have
  (time (group (begin 381) (end 400)))
  (object chicken)
  (attribute (proximity hands)))

(have
  (time (group (begin 381) (end 400)))
  (object chopping-board)
  (attribute (proximity u65888)))

(have
  (time (group (begin 381) (end 400)))
  (object chopping-board)
  (attribute (proximity m65889)))

(have
  (time (group (begin 381) (end 400)))
  (object chopping-board)
  (attribute (proximity meat-mallet)))

(have
  (time (group (begin 381) (end 400)))
  (object chopping-board)
  (attribute (proximity hands)))

(have
  (time (group (begin 381) (end 400)))
  (object u65888)
  (attribute (proximity chopping-board)))

(have
  (time (group (begin 381) (end 400)))
  (object u65888)
  (attribute (proximity m65889)))

(have
  (time (group (begin 381) (end 400)))
  (object u65888)
  (attribute (proximity hands)))

(have
  (time (group (begin 381) (end 400)))
  (object m65889)
  (attribute (proximity chicken)))

(have
  (time (group (begin 381) (end 400)))
  (object m65889)
  (attribute (proximity chopping-board)))

(have
  (time (group (begin 381) (end 400)))
  (object m65889)
  (attribute (proximity u65888)))

(have
  (time (group (begin 381) (end 400)))
  (object m65889)
  (attribute (proximity hands)))

(have
  (time (group (begin 381) (end 400)))
  (object meat-mallet)
  (attribute (proximity chicken)))
```

```
(have
  (time (group (begin 381) (end 400)))
  (object meat-mallet)
  (attribute (proximity chopping-board)))

(have
  (time (group (begin 381) (end 400)))
  (object hands)
  (attribute (proximity chicken)))

(have
  (time (group (begin 381) (end 400)))
  (object hands)
  (attribute (proximity chopping-board)))

(have
  (time (group (begin 381) (end 400)))
  (object hands)
  (attribute (proximity u65888)))

(have
  (time (group (begin 381) (end 400)))
  (object hands)
  (attribute (proximity m65889))))
```

B.Grammar of ACTSCRIPT

```
message:  '(' REQUEST message_name concept ')'
| '(' ANSWER message_name concept ')'
| '(' STATEMENT  message_name concept ')'
| '(' QUESTION  message_name concept ')';

message_name: '(' GENERATOR name number name response ')';

response:
| '(' RESPONSE name number ')';

concept: action | description | '(' NOT action ')'
| '(' NOT description ')' | INTERROGATION | WILDCARD;

name:     string  | INTERROGATION | WILDCARD;

action: '(' ACTION name action_description ')';

action_description: '(' action_primitive act_attribute_list ')'
| INTERROGATION | WILDCARD;


action_primitive: DOSOMETHING |  MOVE     | PRODUCE | ATTEND
| PTRANS | PROPEL | GRASP | ATRANS | MTRANS | MBUILD
| INTERROGATION | WILDCARD;

act_attribute_list: act_attribute | act_attribute_list act_attribute
| INTERROGATION | WILDCARD ;

act_attribute: '(' SPECIAL string_list ')'
| '(' OBJECT name ')'
| '(' OBJECTCONCEPT concept_list ')'
| '(' DIRECTION to_position from_position ')'
| '(' PATH position_list ')'
| '(' RECIPIENT to_name from_name ')'
```

```
| '(' LOCATION position ')'
| '(' INTENSITY value ')'
| '(' VELOCITY value ')'
| '(' INTERVAL time_interval ')'
| '(' WHEN time_description ')'
| '(' MODE string_list ')'
| '(' RESULT '(' CHANGE to_attribute from_attribute ')' ')'
| '(' INSTRUMENT concept_list ')'
| '(' REASON concept_list ')'
| '(' ENABLE concept_list ')'
| '(' ENABLEDBY concept_list ')' ;

concept_list: concept | concept_list concept ;

to_position: | TO position ;

from_position: | FROM position ;

to_name: | TO name ;

from_name: | FROM name ;

to_attribute: | TO description ;

from_attribute: | FROM description ;

description: '(' DESCRIPTION name attribute_description ')' ;

attribute_description: '(' ATTRIBUTE attribute_list ')'
| INTERROGATION | WILDCARD ;

attribute_list: attribute | attribute_list  attribute
| INTERROGATION | WILDCARD ;


attribute: '(' SPECIAL string_list ')'
| '(' WHEREABOUT position ')'
| '(' SPEED measure ')'
| '(' SIZE  measure ')'
| '(' NUMBEROF value ')'
| '(' BOUNDELLIPSE position measure measure measure ')'
| '(' WHEN time_description ')'
| '(' INTERVAL time_interval ')'
| '(' MENTALSTATE value ')'
| '(' PHYSICALSTATE value ')'
| '(' CONSCIOUSNESS value ')'
| '(' HEALTH value ')'
| '(' FEAR value ')'
| '(' ANGER value ')'
| '(' DISGUST     value ')'
| '(' SURPRISE value ')'
| '(' KNOW concept_list ')'
```

```
| '(' UNDERSTAND concept_list ')' ;

position_list: position | position_list position ;

position: '(' SPECIAL string_list ')'
| '(' COORD measure measure measure ')'
| '(' measure measure measure ')'
| ORIGIN | INTERROGATION | WILDCARD ;

measure: value | '(' UNITS value unit ')'
| ZEROLENGTH | ONELENGTH ;

unit:     '(' SPECIAL string_list ')' | UNITLENGTH | UNITTIME ;

value: number | ZERO | ONE  | '(' SPECIAL string_list ')'
| INTERROGATION | WILDCARD ;

time_description: ___ | _N_ | P__ | __F | PN_ | _NF | P_F | PNF
| INTERROGATION | WILDCARD ;

time_interval:     '(' TIMED time_start time_end time_duration ')' ;

time_start: | START time_unit ;

time_end: | END time_unit ;

time_duration: | DURATION time_unit ;

time_unit: value | '(' UNITS value UNITTIME ')' ;

number:   <INTNUMBER> | <FLOATNUMBER> ;

string_list:  string | string_list string
| INTERROGATION | WILDCARD ;

string:   <STRING> ;
```

C.Action Frames for “Mix” and “Wrap”

Action Frame Representation for *“Mixing Ingredients”*

```
;;=======================================
;; "chef mixes bread-crumbs, parsley,
;;   paprika, salt, and basil in a bowl"
;;=======================================

(do-something
   (c-time (interval ?chef-is-mixing))
   (actor chef)

   ;; the result of mix is physical
   ;; contact
   (result
    (to
     (have
  (c-time
   (interval
    ?mixed-ingredients
    (relation
     (?mixed-ingredients
     (:oi) ?chef-is-mixing))))
  (object (group bread-crumbs parsley
            paprika salt basil))
  (attribute (phys-cont
   (group bread-crumbs parsley paprika
     salt basil))))))

;; mixing IN something

  (instrument
  (have
  (c-time
  (interval
   ?ingredients-in-bowl
   (relation
    (?ingredients-in-bowl (:oi)
      ?chef-is-mixing)
    (?ingredients-in-bowl (:s :o)
      ?mixed-ingredients))))
     (object (group bread-crumbs parsley
       paprika salt basil))
     (attribute (contained bowl))))

;; the physical mixing

  (instrument
   (move
   (c-time
  (interval
   ?physical-mix
   (relation
    (?physical-mix (:e :s :d :f)
      ?chef-is-mixing)
    (?physical-mix (:s :o)
      ?mixed-ingredients)
    (?physical-mix (:e :d :s :f)
       ?ingredients-in-bowl))))
     (actor chef)
     (object hands)
     (visual
  (detector DET-chaos-front-chef)
        (relation ?physical-mix (:e)
    DET-chaos-front-chef)
  (kinematic
   (motion-type :chaotic)
   (location (front-of (trunk-of chef)))))

  ;; instrument of physical mixing
  (instrument
  (have
   (c-time (interval ?physical-mix))
   (object hands)
   (attribute
     (physical-contact bowl))))))

;; picking up  the bowl

  (instrument
  (grasp
  (c-time
  (interval
   ?pick-up-bowl
   (relation
    (?pick-up-bowl (:b :m :o :s :d)
      ?chef-is-mixing)
    (?pick-up-bowl (:b :m)
```

```
      ?mixed-ingredients)
    (?pick-up-bowl (:b :m)
      ?ingredients-in-bowl)
    (?pick-up-bowl (:b :m)
      ?physical-mix))))
     (actor chef)
     (object bowl)
     (to (location hands))

;; instrument for picking up the bowl
  (instrument
  (move
   (c-time
    (interval
     ?reach-for-bowl
     (relation
      (?reach-for-bowl (:s)
      ?pick-up-bowl))))
   (actor chef)
   (object hands)
   (visual
    (kinematic
     (motion-type :translational)
     (path-geometry ((location ?unknown)
         (location bowl)))))

;; instrument for moving the hands
  (instrument
    (have
     (c-time
      (interval
       ?DET-hands-close-bowl
       (relation
        (?DET-hands-close-bowl (:f)
          ?reach-for-bowl))))
     (object hands)
     (attribute (proximity bowl))
     (visual
      (detector hands-close-bowl)
          (relation hands-close-bowl (:e)
            DET-hands-close-bowl)
      (static
       (position-type :proximity)
       (location bowl)))))))
     (result
  (from
   (have
    (c-time
     (interval
      ?bowl-off-hands
      (relation
       (?bowl-off-hands (:b :m :o :s)
         ?chef-is-mixing)
       (?bowl-off-hands (:b)
         ?mixed-ingredients)
       (?bowl-off-hands (:b)
        ?ingredients-in-bowl)
       (?bowl-off-hands (:b :m)
         ?physical-mix)
       (?bowl-off-hands (:m :o :fi)
         ?pick-up-bowl)
       )))
    (object bowl)
    (attribute
      (not (physical-contact hands)))
    (visual
     (detector DET-no-motion-of-bowl)
     (relation ?bowl-off-hands (:e)
       DET-no-motion-of-bowl)
     (kinematic
      (motion-type :none)))))
  (to
   (have
    (c-time
     (interval
      ?bowl-in-hands
      (relation
       (?bowl-in-hands (:e :f :oi :d :s)
        ?chef-is-mixing)
       (?bowl-in-hands (:o)
        ?mixed-ingredients)
       (?bowl-in-hands (:o)
        ?ingredients-in-bowl)
       (?bowl-in-hands (:e :di :si :fi)
        ?physical-mix)
       (?bowl-in-hands (:mi :oi)
        ?pick-up-bowl)
       (?bowl-in-hands (:mi)
        ?bowl-off-hands)
       )))
    (object bowl)
    (attribute
      (physical-contact hands)))))))

;; putting down the bowl, after mixing

  (instrument
    (grasp
     (c-time
   (interval
    ?put-down-bowl
    (relation
     (?put-down-bowl (:bi :mi :oi :f)
      ?chef-is-mixing)
     (?put-down-bowl (:d)
      ?mixed-ingredients)
     (?put-down-bowl (:d)
      ?ingredients-in-bowl)
     (?put-down-bowl (:bi :mi)
      ?physical-mix)
     (?put-down-bowl (:bi :mi)
      ?pick-up-bowl)
     (?put-down-bowl (:bi)
      ?bowl-off-hands)
     (?put-down-bowl (:mi :oi :f)
      ?bowl-in-hands))))
     (actor chef)
     (object bowl)
     (from (location hands))

;; instrument for puting down the bowl
  (instrument
  (move
   (c-time
    (interval
     ?depart-from-bowl
     (relation
      (?depart-from-bowl (:f)
       ?put-down-bowl))))
   (actor chef)
```

```
   (object hands)
   (visual
    (kinematic
     (motion-type :translational)
     (path-geometry ((location bowl)
         (location ?unknown)))))

;; instrument for moving the hands
  (instrument
    (have
     (c-time
      (interval
       ?hands-close-bowl2
       (relation
        (?hands-close-bowl2 (:s)
         ?depart-from-bowl))))
     (object hands)
     (attribute (proximity bowl))
     (visual
      (detector DET-hands-close-bowl2)
       (relation hands-close-bowl2 (:e)
        DET-hands-close-bowl2 )
      (static
       (position-type :proximity)
       (location bowl)))))))

;; result from putting down the mixing
;;  bowl : bowl off hands

  (result
  (from
   (have
    (c-time
     (interval
      ?bowl-in-hands2
      (relation
       (?bowl-in-hands2 (:e)
        ?bowl-in-hands)
       )))
    (object bowl)
    (attribute
     (physical-contact hands))))
  (to
   (have
    (c-time
     (interval
      ?bowl-off-hands2
      (relation
       (?bowl-off-hands2 (:bi :mi)
        ?chef-is-mixing)
       (?bowl-off-hands2 (:bi :oi :f :d)
        ?mixed-ingredients)
       (?bowl-off-hands2 (:bi :oi :f :d)
        ?ingredients-in-bowl)
       (?bowl-off-hands2 (:bi :mi)
        ?physical-mix)
       (?bowl-off-hands2 (:bi)
        ?pick-up-bowl)
       (?bowl-off-hands2 (:bi :mi)
        ?bowl-off-hands)
       (?bowl-off-hands2 (:mi)
        ?bowl-in-hands)
       (?bowl-off-hands2 (:mi :oi :si)
        ?put-down-bowl)
       )))
    (object bowl)
    (attribute
     (not (physical-contact hands)))
    (visual
     (detector DET-no-motion-of-bowl2)
           (relation bowl-off-hands2 (:e)
       DET-no-motion-of-bowl2)
     (kinematic
      (motion-type :none))))))))

;; geting some bread-crumbs

  (instrument
    (ptrans
     (c-time
  (interval
   ?get-bread-crumbs
   (relation
    (?get-bread-crumbs (:s :d)
     ?chef-is-mixing)
    (?get-bread-crumbs (:b :m)
     ?mixed-ingredients)
      (?get-bread-crumbs (:b :m)
       ?ingredients-in-bowl)
    (?get-bread-crumbs (:b :m)
     ?physical-mix)
    (?get-bread-crumbs (:bi)
     ?pick-up-bowl)
    (?get-bread-crumbs (:bi)
     ?bowl-off-hands)
    (?get-bread-crumbs (:s :d)
     ?bowl-in-hands)
    (?get-bread-crumbs (:b)
     ?put-down-bowl)
    (?get-bread-crumbs (:b)
     ?bowl-off-hands2))))
     (actor chef)
     (object (some bread-crumbs))
     (from (inside bread-crumbs))
     (to (inside bowl))

  (instrument
  (have
   (c-time
    (interval
     ?crumbs-in-container
     (relation
      (?crumbs-in-container (:o :di)
       ?chef-is-mixing)
      (?crumbs-in-container (:b :m :di)
       ?mixed-ingredients)
      (?crumbs-in-container (:b :m :di)
       ?ingredients-in-bowl)
      (?crumbs-in-container (:b :m :di)
       ?physical-mix)
      (?crumbs-in-container (:di)
       ?pick-up-bowl)
      (?crumbs-in-container (:di)
       ?bowl-off-hands)
      (?crumbs-in-container (:o :di)
       ?bowl-in-hands)
      (?crumbs-in-container (:b :di)
       ?put-down-bowl)
      (?crumbs-in-container (:b :di)
       ?bowl-off-hands2)
```

```
      (?crumbs-in-container (:di :fi)
       ?get-bread-crumbs))))
   (object bread-crums)
   (attribute (contained bread-crumbs))))
     (instrument
  (move
   (c-time (interval ?get-bread-crumbs))
   (actor chef)
   (object hands)
   (visual
    (kinematic
     (motion-type :translational
       :repetitive)
     (path-geometry
      ((location bread-crumbs)
         (location bowl)))))

;; instrument for moving the hands
  (instrument
    (have
     (c-time
      (interval
       ?hands-close-crumbs
       (relation
        (?hands-close-crumbs (:e)
         ?get-bread-crumbs))))
     (object hands)
     (attribute (proximity bread-crumbs))
     (visual
      (detector hands-close-crumbs)
        (relation hands-close-crumbs (:e)
         DET-hands-close-crumbs)
      (static
       (position-type :proximity)
       (location bread-crumbs)))))
  ))))

;; the physical obtainment of basil
;; geting some basil

  (instrument
   (ptrans
   (c-time
  (interval
   ?get-basil
   (relation
    (?get-basil (:s :d) ?chef-is-mixing)
    (?get-basil (:b :m)
     ?mixed-ingredients)
    (?get-basil (:b :m)
     ?ingredients-in-bowl)
    (?get-basil (:b :m) ?physical-mix)
    (?get-basil (:bi) ?pick-up-bowl)
    (?get-basil (:bi) ?bowl-off-hands)
    (?get-basil (:s :d) ?bowl-in-hands)
    (?get-basil (:b) ?put-down-bowl)
    (?get-basil (:b) ?bowl-off-hands2))))
     (actor chef)
     (object (some basil))
     (from (inside basil))
     (to (inside bowl))

  (instrument
  (have
   (c-time
    (interval
     ?basil-in-container
     (relation
      (?basil-in-container (:o :di)
       ?chef-is-mixing)
      (?basil-in-container (:b :m :di)
       ?mixed-ingredients)
      (?basil-in-container (:b :m :di)
       ?ingredients-in-bowl)
      (?basil-in-container (:b :m :di)
       ?physical-mix)
      (?basil-in-container (:di)
       ?pick-up-bowl)
      (?basil-in-container (:di)
       ?bowl-off-hands)
      (?basil-in-container (:o :di)
       ?bowl-in-hands)
      (?basil-in-container (:b :di)
       ?put-down-bowl)
      (?basil-in-container (:b :di)
       ?bowl-off-hands2)
      (?basil-in-container (:di :fi)
       ?get-basil))))
   (object bread-crums)
   (attribute (contained basil))))
  (instrument
  (move
   (c-time (interval ?get-basil))
   (actor chef)
   (object hands)
   (visual
    (kinematic
     (motion-type :translational
       :repetitive)
     (path-geometry ((location basil)
         (location bowl)))))

;; instrument for moving the hands
  (instrument
    (have
     (c-time
      (interval
       ?hands-close-basil
       (relation
        (?hands-close-basil (:e)
         ?get-basil))))
     (object hands)
     (attribute (proximity basil))
     (visual
      (detector hands-close-basil)
       (relation hands-close-basil (:e)
        DET-hands-close-basil)
      (static
       (position-type :proximity)
       (location basil)))))
  ))))))
```

Action Frame Representation for *“Wrapping Chicken”*

```
;;====================================
;; "chef wraps chicken with a plastic
;;  bag"
```

```
;;=====================================

(do-something
  (c-time
   (interval ?wrap-chicken
    (relation
     (?wrap-chicken (:b :bi)
      ?mix-main))))

  (actor chef)

;; the result of wraping is physical
;;  contact and containment
  (result
   (to
    (have
     (c-time
  (interval
   ?chicken-wrapped
   (relation
    (?chicken-wrapped (:oi)
     ?wrap-chicken))))
    (object chicken)
    (attribute
     (group (contained plastic-bag))
     (phys-cont plastic-bag)))))

;; getting a plastic-bag
  (instrument
   (grasp
    (c-time
     (interval
  ?get-plastic-bag
  (relation
   (?get-plastic-bag (:b :m :o :s :d)
    ?wrap-chicken)
   (?get-plastic-bag (:b)
    ?chicken-wrapped))))
    (actor chef)
    (object (one-of plastic-bag))
    (to (location hands))
    (from
      (location plastic-bag-container))

;; instrument of grasping the
;;   plastic-bag: moving the hands

  (instrument
   (move
  (c-time
   (interval
    ?reach-for-plastic-bag
    (relation
     (?reach-for-plastic-bag (:d)
      ?wrap-chicken)
     (?reach-for-plastic-bag (:s)
      ?get-plastic-bag))))
  (actor chef)
  (object hands)
  (visual
   (kinematic
    (motion-type :translational)
    (path-geometry ((location ?unknonw)
      (location
       plastic-bag-container)))))

;; instrument for moving the hands to get
;;    the plastic-bag

  (instrument
   (have
    (c-time
     (interval
      ?hands-close-bag-cont
      (relation
       (?hands-close-bag-cont (:f)
         ?reach-for-plastic-bag))))
    (object hands)
    (attribute
      (proximity plastic-bag-container))
    (visual
     (detector DET:hand-close-pbag-box)
     (relation DET:hand-close-pbag-box
      (:e) hands-close-bag-cont)
     (static
      (position-type :proximity)
      (location
        plastic-bag-container)))))))

;; result of grasping the plastic-bag

  (result
     (from
  (c-time
   (interval
    ?plastic-bag-off-hands
    (relation
     (?plastic-bag-off-hands (:m :o)
      ?get-plastic-bag))))
  (object plastic-bag)
  (attribute (physical-contact hands)))
    (to
  (c-time
   (interval
    ?plastic-bag-in-hands
    (relation
     (?plastic-bag-in-hands (:mi :oi)
      ?get-plastic-bag)
     (?plastic-bag-in-hands (:o :fi)
      ?chicken-wrapped)
     (?plastic-bag-in-hands (:mi)
      ?plastic-bag-off-hands))))
  (object plastic-bag)
  (attribute
   (not (physical-contact hands)))))))

;; the physical opening of the
;;  plastic-bag

  (instrument
   (move
    (c-time
     (interval
  ?open-plastic-bag
  (relation
   (?open-plastic-bag (:s :d)
    ?wrap-chicken)
   (?open-plastic-bag (:b)
    ?chicken-wrapped)
   (?open-plastic-bag (:d :f)
```

```
    ?plastic-bag-in-hands))))
    (actor chef)
    (object hands)
     (visual
  (detector DET:chaos-front-trunk1)
   (relation DET:chaos-front-trunk1 (:e)
    open-plastic-bag)
  (kinematic
   (motion-type :chaotic)
   (location
    (front-of (trunk-of chef))))))
    (instrument
     (have
  (c-time (interval ?open-plastic-bag))
  (object hands)
  (attribute
   (physical-contact plastic-bag)))))

;; getting chicken

  (instrument
   (grasp
    (c-time
     (interval
  ?get-chicken
  (relation
   (?get-chicken (:d) ?wrap-chicken)
   (?get-chicken (:b :m)
    ?chicken-wrapped)
   (?get-chicken (:bi :mi :oi)
    ?get-plastic-bag)
   (?get-chicken (:bi :mi)
    ?open-plastic-bag))))
    (actor chef)
    (object (some chicken))
    (to (location hands))
    (from (location chicken-container)))

;; instrument for getting chicken
   (instrument
    (move
     (c-time
  (interval
   ?reach-for-chicken
   (relation
    (?reach-for-chicken (:d)
     ?wrap-chicken)
    (?reach-for-chicken (:s)
     ?get-chicken))))
     (actor chef)
     (object hands)
     (visual
  (kinematic
   (motion-type :translational)
   (path-geometry ((location ?unknonw)
       (location chicken-container))))))

;; instrument for moving the hands to get
;;  the chicken

  (instrument
  (have
  (c-time
   (interval
    ?hand-close-chix-cont
     (relation
      (?hand-close-chix-cont (:f)
       ?reach-for-chicken))))
  (object hands)
  (attribute (proximity chicken-container))
  (visual
   (detector hand-close-chix-cont)
    (relation DET:hand-close-chix-co (:e)
     hand-close-chix-cont)
   (static
    (position-type :proximity)
    (location chicken-container))))))

;; result of grasping the chicken

  (result
   (from
     (c-time
  (interval
   ?chicken-off-hands
   (relation
    (?chicken-off-hands (:m :o)
     ?get-chicken))))
     (object chicken)
     (attribute
      (not (physical-contact hands))))
    (to
     (c-time
  (interval
   ?chicken-in-hands
   (relation
    (?chicken-in-hands (:mi :oi)
     ?get-chicken)
    (?chicken-in-hands (:o :fi)
     ?chicken-wrapped)
    (?chicken-in-hands (:mi)
     ?chicken-off-hands))))
     (object chicken)
     (attribute
      (physical-contact hands)))))

;; the physical wrapping
   (instrument
    (move
     (c-time
  (interval
   ?actual-wrapping
   (relation
    (?actual-wrapping (:e :s :d :f)
     ?wrap-chicken)
    (?actual-wrapping (:s :o)
     ?chicken-wrapped)
    (?actual-wrapping (:bi :mi)
     ?get-plastic-bag)
    (?actual-wrapping (:bi :mi)
     ?get-chicken)
    (?actual-wrapping (:bi :mi)
     ?open-plastic-bag)
    (?actual-wrapping (:d :f)
     ?chicken-in-hands)
    (?actual-wrapping (:d :f)
     ?plastic-bag-in-hands))))
     (actor chef)
     (object hands)
     (visual
```

```
  (detector DET:chaos-front-trunk2)
   (relation DET:chaos-front-trunk2 (:e)
    actual-wrapping)
  (kinematic
   (motion-type :chaotic)
   (location
    (front-of (trunk-of chef))))))

;; instrument of physical wrapping
  (instrument
   (have
  (c-time (interval ?actual-wrapping))
  (object hands)
  (attribute
   (physical-contact
    (group plastic-bag chicken))))))


;; putting down the chicken and the
;;  plastic-bag

  (instrument
   (grasp
   (c-time
  (interval
   ?put-down-wrap-chix
   (relation
    (?put-down-wrap-chix (:bi :mi :oi :f)
     ?wrap-chicken)
    (?put-down-wrap-chix (:d)
     ?chicken-wrapped)
    (?put-down-wrap-chix (:bi :mi)
     ?actual-wrapping)
    (?put-down-wrap-chix (:oi :mi :f)
     ?chicken-in-hands)
    (?put-down-wrap-chix (:oi :mi :f)
     ?plastic-bag-in-hands))))

     (actor chef)
     (object (group chicken plastic-bag))
     (from (location hands))

;; instrument for putting down the bowl

  (instrument
  (move
   (c-time
    (interval
     ?depart-from-wrap-chix
     (relation
      (?depart-from-wrap-chix (:f)
       ?put-down-wrap-chix))))
   (actor chef)
   (object hands)
   (visual
    (kinematic
     (motion-type :translational)
     (path-geometry
      ((location
       (group chicken plastic-bag))
        (location ?unknown))))))

;; instrument for moving the hands
;;  proximity to static wrapped chicken

  (instrument
   (have
    (c-time
     (interval
      ?hands-close-wrap-chix
      (relation
       (?hands-close-wrap-chix
        (:s) ?depart-from-wrap-chix))))
    (object hands)
    (attribute
     (proximity (group chicken plastic-
bag)))
    (visual
     (detector DET:hands-clo-wrap-chix)
      (relation DET:hands-clo-wrap-chix
       (:e) hands-close-wrap-chix)
     (static
      (position-type :proximity)
      (location
       (group chicken plastic-bag)))))))

;; result from putting down the wrapped
;;   chicken: chicken off hands

  (result
  (from
   (have
    (c-time
     (interval
      ?wrap-chix-in-hands
      (relation
       (?wrap-chix-in-hands (:oi :mi)
        ?wrap-chicken)
       (?wrap-chix-in-hands (:oi :mi)
        ?chicken-in-hands)
       (?wrap-chix-in-hands (:oi :mi)
        ?plastic-bag-in-hands))))
    (object (group chicken plastic-bag))
    (attribute
     (physical-contact hands))))
  (to
   (have
    (c-time
     (interval
      ?wrap-chix-off-hands
      (relation
       (?wrap-chix-off-hands (:oi :mi)
        ?put-down-wrap-chix)
       (?wrap-chix-off-hands (:mi)
        ?wrap-chix-in-hands))))
    (object (group chicken plastic-bag))
    (attribute
     (not (physical-contact hands)))

    (visual
     (detector no-motion-wrap-chix)
      (relation DET:no-motion-wrap-chix
       (:s) wrap-chix-off-hands)
     (kinematic
      (motion-type :none)))))))))
```

D.Grammar of Interval Scripts

```
script: list_of_declarations ;

list_of_declarations: declaration '.'
| list_of_declarations declaration '.' ;

declaration: | cpp_dec | interval_dec | start_dec | stop_dec
| state_dec | reset_dec | data_dec | condition | constraint
| duration ;

cpp_dec: DECLARE '[>' <c++code> '<]';

interval_dec: <name> '=' visible interval_specs ;

visible: | DISPLAY ;

interval_specs: '{' list_of_declarations '}'
| TIMER '(' number ',' number ')'
| CYCLE interval
| SEQUENCE list_of_interval ;

list_of_intervals: interval  | list_of_intervals ',' interval ;

interval: <name> ;

condition: WHEN expression TRYTO list_of_simple_execs ;

list_of_simple_execs: simple_exec
| list_of_simple_execs ';' simple_exec ;

simple_exec: START list_of_intervals
| STOP list_of_intervals
| FORGET list_of_intervals ;

list_of_execs: exec | list_of_execs ';' exec ;

exec: START list_of_intervals
```

```
| STOP list_of_intervals
| RESET list_of_intervals
| SETSTATE pnfexpression
| SETDURATION '(' number ',' number ')'
| EXECUTE CCODE
| IF expression list_of_execs ifelse ENDIF ;

ifelse: | ELSE list_of_execs ;

constraint: BETTER-IF list_of_intervals time_relation list_of_intervals ;

time_relation: time_relation_tag
| time_relation OR time_relation_tag ;

time_relation_tag: EQUAL | MEET | I-MEET | BEFORE | I-BEFORE
| AFTER | DURING | I-DURING | OVERLAP | I-OVERLAP | SSTART
| I-START | FINISH | I-FINISH | ONLY-DURING | NOT-DURING
| DIFFERENT | PRECEDE | I-PRECEDE ;

start_dec: START ':' list_of_execs ;

stop_dec: STOP ':' list_of_execs ;

reset_dec: RESET ':' list_of_execs ;

data_dec: DATA ':' '[>' <c++code> '<]' ;

state_dec: STATE ':' list_of_execs ;

pnfexpression: timetag | interval | CURRENTSTATE | EXPANDEDSTATE;

timetag: PAST | NOW | FUTURE | UNDETERMINED | PAST-OR-NOW
| PAST-OR-FUTURE | NOW-OR-FUTURE | P__ | _N_ | __F | PNF
| PN_ | _NF |P_F ;

expression: expression AND or_expression | or_expression ;

or_expression: or_expression OR primary | primary ;

primary:  pnfexpression | pnfexpression IS timetag
| ISCONTAINED pnfexpression timetag | TRUE | '(' expression ')'
| NOT primary | BOOLEXPRESSION '[>' <c++code> '<]' ;

duration: DURATION  '(' number ',' number ')' ;

number: <number> | '[>' <c++code> '<]' ;
```

References

[1] *Director's User Manual*. MacroMind Inc. 1990.

[2] S. Agamapolis and V. M. Bove Jr. “Multilevel Scripting for Responsive Multimedia”, *IEEE Multimedia,* vol. 4 (4), pp. 40-50. 1997.

[3] J. F. Allen. “Maintaining Knowledge about Temporal Intervals”, *Communications of the ACM,* vol. 26 (11), pp. 832-843. 1983.

[4] J. F. Allen. “Towards a General Theory of Action and Time”, *Artificial Intelligence,* vol. 23, pp. 123-154. 1984.

[5] J. F. Allen. “Time and Time Again: The Many Ways to Represent Time”, *International Journal of Intelligent Systems,* vol. 6 (4), pp. 341-355. 1991.

[6] J. F. Allen and G. Ferguson. “Actions and Events in Interval Temporal Logic”, *Journal of Logic and Computation,* vol. 4 (5), pp. 531-579. 1994.

[7] J. F. Allen and P. J. Hayes. “Moments and Points in an Interval-Based Temporal Logic”, *Computational Intelligence,* vol. 5 (4), pp. 225-238. 1989.

[8] K. Amaya, A. Bruderlin, and T. Calvert. “Emotion from Motion”, *Proc. of Graphics Interface'96,* Toronto, Canada. 1996.

[9] E. Andre and T. Rist. “Coping with Temporal Constraints in Multimedia Presentation Planning”, *Proc. of AAAI'96,* Portland, Oregon, pp. 142-147. 1996.

[10] N. I. Badler, C. B. Phillips, and B. L. Webber. *Simulating Humans: Computer Graphics Animations and Control*. Oxford University Press, Oxford, England. 1993.

[11] B. Bailey, J. A. Konstan, R. Cooley, and M. Dejong. “Nsync - A Toolkit for Building Interactive Multimedia Presentations”, *Proc. of ACM Multimedia'98,* Bristol, England, pp. 257-266. 1998.

[12] P. Baptiste and C. L. Pape. “A Theoretical and Experimental Comparison of Constraint Propagation Techniques for Disjunctive Scheduling”, *Proc. of IJCAI'95,* Montreal, Canada, pp. 600-606. August. 1995.

[13] E. Barba and N. Savarese. *Dictionary of Theatre Anthropology: The Secret Art of the Performer*. Routledge, London, England. 1991.

[14] J. Bates, A. B. Loyall, and W. S. Reilly. "An Architecture for Action, Emotion, and Social Behavior", *Proceedings of the Fourth European Workshop on Modeling Autonomous Agents in a Multi-Agent World*, S. Martino al Cimino, Italy. July. 1992.

[15] B. B. Bederson and A. Druin. "Computer Augmented Environments: New Places to Learn, Work and Play", *Advances in Human-Computer Interaction*, vol. 5. Ablex, Norwood, New Jersey. 1995.

[16] B. Blumberg. *Old Tricks, New Dogs: Ethology and Interactive Creatures*. Ph.D. Thesis. Media Arts and Sciences Program: Massachusetts Institute of Technology, Cambridge, Massachusetts. 1996.

[17] B. M. Blumberg and T. A. Galyean. "Multi-Level Direction of Autonomous Agents for Real-Time Virtual Environments", *Proc. of SIGGRAPH'95*. 1995.

[18] A. Bobick, S. Intille, J. Davis, F. Baird, C. Pinhanez, L. Campbell, Y. Ivanov, A. Schutte, and A. Wilson. "The KidsRoom: A Perceptually-Based Interactive and Immersive Story Environment", *to appear in Presence: Teleoperators and Virtual Environments*. 1999.

[19] A. Bobick and C. Pinhanez. "Using Approximate Models as Source of Contextual Information for Vision Processing", *Proc. of the ICCV'95 Workshop on Context-Based Vision*, Cambridge, Massachusetts, pp. 13-21. July. 1995.

[20] A. F. Bobick. "Movement, Activity, and Action: The Role of Knowledge in the Perception of Motion", *Phil. Trans. Royal Society London B*, vol. 352, pp. 1257-1265. 1997.

[21] A. F. Bobick and R. C. Bolles. "The Representation Space Paradigm of Concurrent Evolving Object Descriptions", *IEEE PAMI*, vol. 14 (2), pp. 146-156. 1992.

[22] A. F. Bobick and Y. Ivanov. "Action Recognition Using Probabilistic Parsing", *Proc. of CVPR'98*, Santa Barbara, California, pp. 196-202. 1998.

[23] A. F. Bobick and C. S. Pinhanez. "Controlling View-Based Algorithms Using Approximate World Models and Action Information", *Proc. of CVPR'97*, Puerto Rico, USA, pp. 955-962. June. 1997.

[24] A. F. Bobick and A. D. Wilson. "A State-Based Approach to the Representation and Recognition of Gesture", *IEEE PAMI*, vol. 19 (12). 1997.

[25] A. Borning. "The Programming Language Aspects of ThingLab, A Constraint-Oriented Simulation Laboratory", *ACM Transactions on Programming Languages and Systems*, vol. 3 (4). 1981.

[26] M. Brand. "Understanding Manipulation in Video", *Proc. of 2nd International Conference on Face and Gesture Recognition (FG'96)*, Killington, VT, pp. 94-99. 1996.

[27] M. Brand, N. Oliver, and A. Pentland. “Coupled Hidden Markov Models for Complex Action Recognition”, *Proc. of CVPR'97*, Puerto Rico, USA, pp. 994-999. 1997.

[28] S. Brand. *How Buildings Learn*. Viking, New York. 243 pages. 1994.

[29] C. Bregler and J. Malik. “Tracking People with Twists and Exponential Maps”, *Proc. of CVPR'98*, Santa Barbara, California, pp. 8-15. June. 1998.

[30] K. M. Brooks. “Do Story Agents Use Rocking Chairs? The Theory and Implementation of One Model for Computational Narrative”, *Proc. of the ACM Multimedia'96*, pp. 1-12. November. 1996.

[31] R. Brooks. “A Robust Layered Control System for a Mobile Robot”, *IEEE Journal of Robotics and Automation*, vol. RA-2 (1), pp. 14-23. 1986.

[32] R. Brooks. “Elephants Don't Play Chess”, *Robotics and Automation Systems*, vol. 6, pp. 3-15. 1990.

[33] R. Brooks. “Intelligence without Reason”, *Artificial Intelligence*, vol. 47, pp. 139-159. 1991.

[34] M. C. Buchanan and P. T. Zellweger. “Automatic Temporal Layout Mechanisms”, *Proc. of ACM Multimedia'93*, Ahaheim, California, pp. 341-350. August. 1993.

[35] L. W. Campbell, D. A. Becker, A. Azarbayejani, A. F. Bobick, and A. Pentland. “Invariant Features for 3-D Gesture Recognition”, *Proc. of the Second International Conference on Automatic Face and Gesture Recognition (FG'96)*, Killington, Vermont, pp. 157-162. 1996.

[36] J. Cassell, C. Pelachaud, N. Badler, and M. Steedman. “Animated Conversation: Rule-Based Generation of Facial Expression, Gesture & Spoken Intonation for Multiple Conversational Agents”, *Proc. of SIGGRAPH'94*, Orlando, Florida, pp. 413-420. July, 24-29. 1994.

[37] D. Chapman. *Vision, Instruction, and Action*. The MIT Press, Cambridge, Massachusetts. 1991.

[38] E. Charniak and D. McDermott. *Introduction to Artificial Intelligence*. Addison-Wesley, Reading, Massachusetts. 1985.

[39] M. H. Coen. “Building Brains for Rooms: Designing Distributed Software Agents”, *Proc. of IAAI'97*, Providence, Connecticut, pp. 971-977. August. 1997.

[40] T. Darrell and A. Pentland. “Space-Time Gestures”, *Proc. of CVPR'93*, pp. 335-340. 1993.

[41] G. Davenport and L. Friedlander. “Interactive Transformational Environments: Wheel of Life”, in *Contextual Media: Multimedia and Interpretation*, E. Barrett and M. Redmond (eds.). The MIT Press, Cambridge, Massachusetts. pp. 1-25. 1995.

[42] C. Davies and J. Harrison. “Osmose: Towards Broadening the Aesthetics of Virtual Reality”, *ACM Computer Graphics: Virtual Reality*, vol. 30 (4). 1998.

[43] J. W. Davis and A. Bobick. “The Representation and Recognition of Human Movement Using Temporal Templates”, *Proc. of CVPR'97*, pp. 928-934. June. 1997.

[44] J. W. Davis and A. F. Bobick. “Virtual PAT: a Virtual Personal Aerobics Trainer”, *Proc. of Workshop on Perceptual User Interfaces (PUI'98)*, San Francisco, California, pp. 13-18. November. 1998.

[45] T. Dean and D. McDermott. “Temporal Data Base Management”, *Artificial Intelligence,* vol. 32, pp. 1-55. 1987.

[46] R. Dechter. “From Local to Global Consistency”, *Artificial Intelligence,* vol. 55 (1), pp. 87-107. 1992.

[47] R. Dechter, I. Meiri, and J. Pearl. “Temporal Constraint Networks”, *Artificial Intelligence,* vol. 49 (1-3), pp. 225-233. 1991.

[48] R. Dechter and J. Pearl. “Directed Constraint Networks”, *Proc. of IJCAI'91*, Sydney, Australia, pp. 1164-1170. 1991.

[49] C. Elliott, G. Schechter, R. Yeung, and S. Abi-Ezzi. “TBAG: A High Level Framework for Interactive, Animated 3D Graphics Applications”, *Proc. of SIGGRAPH'94*, Orlando, Florida, pp. 421-434. July 24-29. 1994.

[50] R. E. Fikes and N. J. Nilsson. “STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving”, *Artificial Intelligence,* vol. 2, pp. 189-205. 1971.

[51] C. Freksa. “Temporal Reasoning Based on Semi-Intervals”, *Artificial Intelligence,* vol. 54 (1-2), pp. 199-227. 1992.

[52] T. A. Galyean. *Narrative Guidance of Interactivity*. Ph.D. Thesis. Media Arts and Sciences Program: Massachusetts Institute of Technology, Cambridge, Massachusetts. 1995.

[53] D. M. Gavrila and L. S. Davis. “3-D Model Based Tracking of Human Upper Body Movement: a Multi-View Approach”, *Proc. of the IEEE-PAMI International Symposium on Computer Vision*, Coral Gables, Florida, pp. 253-258. November. 1995.

[54] A. Geist, A. Beguelim, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine: A User's Guide and Tutorial for Networked Parallel Computing*. The MIT Press, Cambridge, Massachusetts. 1994.

[55] M. Gelfond and V. Lifschitz. “Representing Action and Change by Logic Programs”, *Journal of Logic Programming,* vol. 17 (2,3,4), pp. 301-323. 1993.

[56] A. Gerevini and L. Schubert. “Efficient Algorithms for Qualitative Reasoning about Time”, *Artificial Intelligence,* vol. 74 (2), pp. 207-248. 1995.

[57] W. E. L. Grimson, C. Stauffer, R. Romano, and L. Lee. “Using Adaptive Tracking to Classify and Monitor Activities in a Site”, *Proc. of CVPR'98*, Santa Barbara, California, pp. 22-29. June. 1998.

[58] P. A. Hall. “Equivalence Between AND/OR Graphs and Context-Free Grammars”, *CACM*, vol. 21, pp. 444-445. 1973.

[59] R. Hamakawa and J. Rekimoto. “Object Composition and Playback Models for Handling Multimedia Data”, *Proc. of ACM Multimedia'93*, Ahaheim, California, pp. 273-281. August. 1993.

[60] G. G. Hendrix. “Modeling Simultaneous Actions and Continuous Processes”, *Artificial Intelligence,* vol. 4 (3), pp. 145-180. 1973.

[61] X. D. Huang, Y. Ariki, and M. A. Jack. *Hidden Markov Models for Speech Recognition*. Edinburgh University Press. 1990.

[62] E. Hyvönen. “Constraint Reasoning Based on Interval Arithmetic: The Tolerance Propagation Approach”, *Artificial Intelligence*, vol. 58 (1-3), pp. 71-112. 1992.

[63] S. Intille and A. Bobick. “A Framework for Recognizing Multi-Agent Action from Visual Evidence”, *Proc. of AAAI'99*, Orlando, Florida. July. 1999.

[64] H. Ishii and B. Ullmer. “Tangible Bits: Towards Seamless Interfaces between People, Bits, and Atoms”, *Proc. of CHI'97*, Atlanta, Georgia, pp. 234-241. March. 1997.

[65] D. Israel, J. Perry, and S. Tutiya. “Actions and Movements”, *Proc. of IJCAI'91*, Sydney, Australia. pp. 1060-1065. 1991.

[66] Y. Ivanov, A. Bobick, and J. Liu. “Fast Lighting Independent Background subtraction”, *Proc. of the IEEE Workshop on Visual Surveillance (VS'98)*, Bombay, India, pp. 49-55. January. 1998.

[67] Y. Ivanov, C. Stauffer, A. Bobick, and E. Grimson. “Video Surveillance of Interactions”, *Proc. of the CVPR'99 Workshop on Visual Surveillance*, Fort Collins, Colorado. November. 1998.

[68] R. Jackendoff. *Semantics and Cognition*. The M.I.T. Press, Cambridge, MA. 1983.

[69] R. Jackendoff. *Semantic Structures*. The M.I.T. Press, Cambridge, MA. 1990.

[70] T. Jebara and A. Pentland. “Action Reaction Learning: Automatic Visual Analysis and Synthesis of Interactive Behavior”, *Proc. of the International Conference on Computer Vision Systems (ICVS'99)*, Las Palmas, Gran Canaria, Spain, pp. 273-292. January 13-15. 1999.

[71] M. Johnson, A. Wilson, C. Kline, B. Blumberg, and A. Bobick. “Sympathetic Interfaces: Using a Plush Toy to Direct Synthetic Characters”, *Proc. of CHI'99*, Pittsburgh, Pennsylvania. May.1999.

[72] N. Johnson and D. C. Hogg. “Learning the Distribution of Object Trajectories for Event Recognition”, *Proc. of British Machine Vision Conference*. 1995.

[73] K. Johnstone. *IMPRO: Improvisation and Theatre*. Methuen Drama, England. 1981.

[74] M. Jourdan, N. Layaida, C. Roisin, L. Sabry-Ismail, and L. Tardif. “Madeus, an Authoring Environment for Interactive Multimedia Documents”, *Proc. of ACM Multimedia'98*, Bristol, England, pp. 267-272. September 12-16. 1998.

[75] K. M. Kahn and A. G. Gorry. “Mechanizing Temporal Knowledge”, *Artificial Intelligence,* vol. 9 (2), pp. 87-108. 1977.

[76] K. I. Kakizaki. “Generating the Animation of a 3D Agent from Explanation Text”, *Proc. of ACM Multimedia'98*, Bristol, England, pp. 139-144. 1998.

[77] J. K. Kalita. *Natural Language Control of Animation of Task Performance in a Physical Domain*. Ph.D. Thesis. Department of Computer and Information Science: University of Pennsylvania, Philadelphia, Pennsylvania. 1991.

[78] H. A. Kautz and P. B. Ladkin. “Integrating Metric and Qualitative Temporal Reasoning”, *Proc. of AAAI'91*, pp. 241-246. July. 1991.

[79] M. W. Krueger. *Artificial Reality II*. Addison-Wesley. 1990.

[80] V. Kumar. “Algorithms for Constraint-Satisfaction Problems: a Survey”, *AI Magazine,* vol. 13, pp. 32-44. 1992.

[81] Y. Kuniyoshi and H. Inoue. “Qualitative Recognition of Ongoing Human Action Sequences”, *Proc. of IJCAI'93*, pp. 1600-1609. 1993.

[82] P. Ladkin and A. Reinefeld. “Effective Solution of Qualitative Interval Constraint Problems”, *Artificial Intelligence,* vol. 57 (1), pp. 105-124. 1992.

[83] S. K. Langer. *Feeling and Form*. Charles Scribner's Sons, New York, New York. 1953.

[84] B. Laurel, R. Strickland, and R. Tow. “Placeholder: Landscape and Narrative in Virtual Environments”, *ACM Computer Graphics Quarterly*, vol. 28 (2). 1994.

[85] J. Lobo, G. Mendez, and S. R. Taylor. “Adding Knowledge to the Action Description Language A”, *Proc. of AAAI'97*, Providence, Rhode Island, pp. 454-459. July. 1997.

[86] R. E. Lovell and J. D. Mitchell. “Using Human Movement to Control Activities in Theatrical Environments”, *Proc. of Third International Conference on Dance and Technology*. 1995.

[87] A. B. Loyall and J. Bates. “Hap: A Reactive, Adaptive Architecture for Agents,” Carnegie Mellon University, Pittsburgh, Pennsylvania Technical Report CMU-CS-91-147, June. 1991.

[88] T. Machover. “Hyperinstruments: a Progress Report,” M.I.T. Media Laboratory Technical Report. January. 1992.

[89] T. Machover. “Brain Opera”, in *Memesis: The Future of Evolution*. Ars Electronica Editions, Linz, Austria. 1996.

[90] A. K. Mackworth. “Consistency in Networks of Relations”, *Artificial Intelligence,* vol. 8 (1), pp. 99-118. 1977.

[91] R. Maconie. “Four Criteria of Electronic Music”, in *Stockhausen on Music*, R. Maconie (ed.). Marion Boyars, New York, New York. pp. 88-111. 1989.

[92] A. B. Maddox and J. Pustejovsky. “Linguistic Descriptions of Visual Event Perceptions”, *Proc. of the Ninth Annual Cognitive Science Society Conference*, Seattle, Washington, pp. 442-454. 1987.

[93] P. Maes. “Agents that Reduce Work and Information Overload”, *Communications of the ACM*, vol. 37 (7), pp. 31-40. 1995.

[94] P. Maes, T. Darrell, B. Blumberg, and A. Pentland. “The ALIVE System: Full-Body Interaction with Autonomous Agents”, *Proc. of the Computer Animation'95 Conference*, Geneva, Switzerland. April. 1995.

[95] P. Maes, Y. Lashkari, and M. Metral. “Collaborative Interface Agents”, in *Readings in Agents*, M. N. Huhns and M. P. Singh (eds.). Morgan Kaufmann Publishers. 1997.

[96] J. Malik and T. O. Binford. “Reasoning in Time and Space”, *Proc. of IJCAI'83*, Karlsruhe, Federal Republic of Germany, pp. 343-345. 1983.

[97] R. Mann, A. Jepson, and J. Siskind. “Computational Perception of Scene Dynamics”, *Proc. of Fourth European Conference in Computer Vision*. 1996.

[98] K. Mase and R. Kadobayashi. “Meta-museum: a Supportive Augumented Reality Environment for Knowledge Sharing”, *International Conference on Virtual Systems and Multimedia*. September. 1996.

[99] M. J. Mataric. “Behavior-Based Control: Examples from Navigation, Learning, and Group Behavior”, *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9 (2-3), pp. 323-336. 1997.

[100] B. Maubrey. “Audio Jackets and Other Electroacustic Clothes”, *Leonardo*, vol. 28 (2), pp. 93-97. 1995.

[101] J. McCarthy and P. Hayes. “Some Philosophical Problems from the Standpoint of Artificial Intelligence”, in *Machine Intelligence 4*, B. Meltzer, D. Michie, and M. Swann (eds.). Edinburgh University Press, Edinburgh, Scotland. pp. 463-502. 1969.

[102] I. Meiri. “Combining Qualitative and Quantitative Constraints in Temporal Reasoning”, in *Proc. of AAAI'91*. pp. 260-267. 1991.

[103] I. Meiri. “Combining Qualitative and Quantitative Constraints in Temporal Reasoning”, *Artificial Intelligence*, vol. 87 (1-2), pp. 343-385. 1996.

[104] G. A. Miller and P. N. Johnson-Laird. *Language and Perception*. Belknap Press, Cambridge, Massachusetts. 1976.

[105] M. Minsky. “A Framework for Representing Knowledge”, in *The Psychology of Computer Vision*, P. Winston (ed.). McGraw-Hill, New York, New York. pp. 211-277. 1975.

[106] M. Minsky. *The Society of Mind*. Simon & Schuster, New York, New York. 1985.

[107] R. Mohr and T. C. Henderson. “Arc and Path Consistency Revisited”, *Artificial Intelligence,* vol. 28 (2), pp. 225-233. 1986.

[108] U. Montanari. “Networks of Constraints: Fundamental Properties and Applications to Picture Processing”, *Information Sciences,* vol. 7, pp. 95-132. 1974.

[109] J. Murray. *Hamlet on the Holodeck: the Future of Narrative in Cyberspace*. The Free Press, Simon & Schuster, New York, New York. 1997.

[110] B. A. Nadel. “Constraint Satisfaction Algorithms”, *Computational Intelligence,* vol. 5, pp. 188-224. 1989.

[111] H.-H. Nagel. “A Vision of 'Vision and Language' Comprises Action: An Example from Road Traffic”, *Artificial Intelligence Review,* vol. 8, pp. 189-214. 1995.

[112] R. Nakatsu, N. Tosa, and T. Ochi. “Interactive Movie System with Multi-person Participation and Anytime Interaction Capabilities”, *Proc. of ACM Multimedia'98,* Bristol, England, pp. 129-137. September 12-16. 1998.

[113] B. Neumann. “Natural Language Description of Time-Varying Scenes”, in *Semantic Structures: Advances in Natural Language Processing*, D. L. Waltz (ed.). Lawrence Erlbaum Associates. pp. 167-206. 1989.

[114] B. Neumann and H. Novak. “Event Models for Recognition and Natural Language Description of Events in Real-World Image Sequences”, *Proc. of IJCAI'83*, pp. 724-726. 1993.

[115] D. Newtson, G. Engquist, and J. Bois. “The Objective Basis of Behavior Units”, *Journal of Personality and Social Psychology,* vol. 35 (12), pp. 847-862. 1977.

[116] N. J. Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, New York, New York. 1971.

[117] S. Oettermann. *The Panorama: History of a Mass Medium*. Zone Books, New York. 407 pages. 1997.

[118] N. Olivier, B. Rosario, and A. Pentland. “A Bayesian Computer Vision System for Modeling Human Interactions”, *Proc. of the International Conference on Computer Vision Systems (ICVS'99),* Las Palmas, Gran Canaria, Spain, pp. 254-272. January 13-15. 1999.

[119] J. A. Paradiso. “New Instruments and Gestural Sensors for Musical Interaction and Performance”, *Journal of New Music Research*. 1998.

[120] R. Pausch, T. Burnette, A. C. Capeheart, M. Conway, D. Cosgrove, R. DeLine, J. Durbin, R. Gossweiler, S. Koga, and J. White. “A Brief Architectural Overview of Alice, a Rapid Prototyping System for Virtual Reality”, *IEEE Computer Graphics and Applications*. 1995.

[121] R. Pausch, J. Snoddy, R. Taylor, S. Watson, and E. Haseltine. “Disney's Alladin: First Steps Toward Storytelling in Virtual Reality”, *Proc. of SIGGRAPH'96*, pp. 193-203. August. 1996.

[122] A. Pentland. “Smart Rooms”, *Scientific American,* vol. 274 (4), pp. 68-76. 1996.

[123] A. Pentland. “Wearable Intelligence”, *Scientific American Presents Exploring Intelligence,* vol. 9 (4). 1998.

[124] K. Perlin. “Real Time Responsive Animation with Personality”, *IEEE Transactions on Visualization and Computer Graphics,* vol. 1 (1), pp. 5-15. 1995.

[125] K. Perlin and A. Goldberg. “Improv: A System for Scripting Interactive Actors in Virtual Worlds”, *Proc. of SIGGRAPH'96*. August. 1996.

[126] K. Perlin and A. Goldberg. “Improvisational Animation”, *Proc. of The Society for Computer Simulation International Conference on Virtual Worlds and Simulation (VWSIM'99),* San Francisco, California. January 17-20. 1999.

[127] R. W. Picard. *Affective Computing*. The MIT Press, Cambridge, Massachusetts. 292 pages. 1997.

[128] C. S. Pinhanez. “Computer Theater,” M.I.T. Media Laboratory Perceptual Computing Section, Technical Report #378. May. 1996.

[129] C. S. Pinhanez. “Computer Theater”, *Proc. of the Eighth International Symposium on Electronic Arts (ISEA'97)*, Chicago, Illinois. September. 1997.

[130] C. S. Pinhanez and A. F. Bobick. “Approximate World Models: Incorporating Qualitative and Linguistic Information into Vision Systems”, *Proc. of AAAI'96*, Portland, Oregon, pp. 1116-1123. August. 1996.

[131] C. S. Pinhanez and A. F. Bobick. “Computer Theater: Stage for Action Understanding”, *Proc. of the AAAI'96 Workshop on Entertainment and AI/A-Life*, Portland, Oregon, pp. 28-33. August. 1996.

[132] C. S. Pinhanez and A. F. Bobick. “PNF Calculus: A Method for the Representation and Fast Recognition of Temporal Structure,” M.I.T. Media Laboratory Perceptual Computing Section, Technical Report #389. September. 1996.

[133] C. S. Pinhanez and A. F. Bobick. “Intelligent Studios: Modeling Space and Action to Control TV Cameras”, *Applications of Artificial Intelligence,* vol. 11, pp. 285-305. 1997.

[134] C. S. Pinhanez and A. F. Bobick. “PNF Propagation and the Detection of Actions Described by Temporal Intervals”, *Proc. of the DARPA Image Understanding Workshop*, New Orleans, Louisiana. May. 1997.

[135] C. S. Pinhanez and A. F. Bobick. “Human Action Detection Using PNF Propagation of Temporal Constraints”, *Proc. of CVPR'98*, Santa Barbara, California, pp. 898-904. June. 1998.

[136] C. S. Pinhanez and A. F. Bobick. "'It/I': A Theater Play Featuring an Autonomous Computer Graphics Character," M.I.T. Media Laboratory Perceptual Computing Section, Technical Report #455. January. 1998.

[137] C. S. Pinhanez, K. Mase, and A. F. Bobick. "Interval Scripts: A Design Paradigm for Story-Based Interactive Systems", *Proc. of CHI'97*, Atlanta, Georgia, pp. 287-294. March. 1997.

[138] S. Pinker. *Learnability and Cognition*. The M.I.T. Press, Cambridge, MA. 1989.

[139] R. Polana and R. Nelson. "Low Level Recognition of Human Motion", *Proc. of IEEE Workshop on Motion of Non-Rigid and Articulated Objects*, Austin, Texas, pp. 77-82. November. 1994.

[140] L. R. Rabiner and B. H. Juang. "An Introduction to Hidden Markov Models", *IEEE ASSP Magazine*, vol. 3 (1), pp. 4-16. 1986.

[141] R. Raskar, G. Welch, M. Cutts, A. Lake, and L. Stesin. "The Office of the Future: A Unified Approach to Image-Based Modeling and Spatially Immersive Displays", *Proc. of SIGGRAPH'98*, Orlando, Florida, pp. 179-188. July. 1998.

[142] M. Reaney. "Virtual Scenography: The Actor, Audience, Computer Interface", *Theatre Design and Technology*, vol. 32 (1), pp. 36-43. 1996.

[143] T. Richards. *At Work with Grotowski on Physical Actions*. Routledge, London, England. 1993.

[144] C. J. Rieger III. "Conceptual Memory and Inference", in *Conceptual Information Processing*. North-Holland. pp. 157-288. 1975.

[145] K. Rohr. "Towards Model-Based Recognition of Human Movements in Image Sequences", *CVGIP: Image Understanding*, vol. 59 (1), pp. 94-115. 1994.

[146] S. Rosenschein and L. Kaelbling. "The Synthesis of Machines with Probable Epistemic Properties", *Proc. of Conf. on Theoretical Aspects of Reasoning about Knowledge*, Los Altos, California, pp. 83-98. 1986.

[147] R. Rowe. *Interactive Music Systems*. The MIT Press, Cambridge, Massachusetts. 1993.

[148] E. D. Sacerdoti. *A Structure for Plans and Behavior*. Elsevier North-Holland, New York. 1977.

[149] R. C. Schank. "Conceptual Dependency Theory", in *Conceptual Information Processing*. North-Holland. pp. 22-82. 1975.

[150] R. C. Schank, N. M. Goldman, C. J. Rieger III, and C. K. Riesbeck. *Conceptual Information Processing*. North-Holland. 1975.

[151] R. Schechner. *Performance Theory*. Routledge, London, England. 1988.

[152] J. Schlenzig, E. Hunter, and R. Jain. “Recursive Identification of Gesture Inputs Using Hidden Markov Models”, *Proc. of the Second Annual Conference on Applications of Computer Vision*, pp. 187-194. 1994.

[153] E. Schwartz and D. Godfrey. *Music since 1945: Issues, Materials, and Literature*. Schirmer Books, New York, New York. 1993.

[154] K. C. Selcuk, B. Prabhakaran, and V. S. Subrahmanian. “CHIMP: A Framework for Supporting Distributed Multimedia Document Authoring and Presentation”, *Proc. of ACM Multimedia'96*, Boston, Massachusetts, pp. 329-339. November. 1996.

[155] C. Shaw, M. Green, J. Liang, and Y. Sun. “Decoupled Simulation in Virtual Reality with the MR Toolkit”, *ACM Transactions on Information Systems*, vol. 11 (3), pp. 287-317. 1993.

[156] Y. Shoham. “Temporal Logics in AI: Semantical and Ontological Considerations”, *Artificial Intelligence*, vol. 33 (1), pp. 89-104. 1987.

[157] K. Sims. “Evolving Virtual Creatures”, *Proc. of SIGGRAPH'94*, Orlando, Florida, pp. 15-34. July 24-29. 1994.

[158] J. M. Siskind. *Naive Physics, Event Perception, Lexical Semantics, and Language Acquisition*. Ph.D. Thesis. Dept. of Electrical Engineering and Computer Science: Massachusetts Institute of Technology, Cambridge, Massachusetts. 1992.

[159] J. M. Siskind. “Grounding Language in Perception”, *Artificial Intelligence Review*, vol. 8, pp. 371-391. 1994.

[160] J. R. Slagle. “A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculus”, in *Computers and Thought*, E. A. Feigenbaum and J. Feldman (eds.). McGraw-Hill, New York, New York. pp. 191-203. 1963.

[161] C. Sommerer and L. Mignonneau. “Art as a Living System”, *Leonardo*, vol. 30 (5). 1997.

[162] F. Sparacino, K. Hall, C. Wren, G. Davenport, and A. Pentland. “Improvisational Theater Space”, *Proc. of the Sixth Biennial Symposium for Arts and Technology*, New London, Connecticut, pp. 207-208. March. 1997.

[163] L. Stark and K. Bowyer. “Achieving Generalized Object Recognition through Reasoning about Association of Function to Structure”, *IEEE PAMI*, vol. 13 (10), pp. 1097-1104. 1991.

[164] L. Stark and K. Bowyer. “Functional Context in Vision”, *Proc. of the ICCV'95 Workshop on Context-Based Vision*, Cambridge, Massachusetts, pp. 63-74. July. 1995.

[165] T. Starner and A. Pentland. “Real-Time American Sign Language Recognition from Video Using Hidden Markov Models”, *Proc. of the IEEE-PAMI International Symposium on Computer Vision*, Coral Gables, Florida, pp. 265-270. November. 1995.

[166] L. A. Stein and L. Morgenstern. “Motivated Action Theory: a Formal Theory of Causal Reasoning”, *Artificial Intelligence,* vol. 71, pp. 1-42. 1994.

[167] S. Strassman. “Semi-Autonomous Animated Actors”, *Proc. of AAAI'94*, Seattle, Washington, pp. 128-134. August. 1994.

[168] C. Sul, K. Lee, and K. Wohn. “Virtual Stage: A Location-based Karaoke System”, *IEEE Multimedia,* vol. 5 (2). 1998.

[169] D. Terzopoulos and K. Waters. “Analysis and Synthesis of Facial Image Sequences Using Physical and Anatomical Models”, *IEEE PAMI,* vol. 15 (6), pp. 569-579. 1993.

[170] N. M. Thalmann and D. Thalmann. *Synthetic Actors in Computer Generated 3D Films.* Springer-Verlag, Berlin, Germany. 1990.

[171] R. Thibadeau. “Artificial Perception of Actions”, *Cognitive Science,* vol. 10, pp. 117-149. 1986.

[172] N. Tosa, H. Hashimoto, K. Sezaki, Y. Kunii, T. Yamada, K. Sabe, R. Nishino, H. Harashima, and F. Harashima. “Network-Based Neuro-Baby with Robotic Hand”, *Proc. of IJCAI'95 Workshop on Entertainment and AI/Alife*, Montreal, Canada. August. 1995.

[173] N. Tosa and R. Nakatsu. “For Interactive Virtual Drama: Body Communication Actor”, *Proc. of 7th International Symposium on Electronic Art*, Rotterdam, The Netherlands. September. 1996.

[174] N. Tosa and R. Nakatsu. “Interactive Poem System”, *Proc. of ACM Multimedia'98*, Bristol, England, pp. 115-118. September 12-16. 1998.

[175] G. van Rossum, J. Jansen, K. Mullender, and D. Bulterman. “CMIFed: a Presentation Environment for Portable Hypermedia Documents”, *Proc. of ACM Multimedia'93*, California. 1993.

[176] P. van Beek. “Reasoning about Qualitative Temporal Information”, *Artificial Intelligence,* vol. 58 (1-3), pp. 297-326. 1992.

[177] M. Vilain and H. Kautz. “Constraint Propagation Algorithms for Temporal Reasoning”, *Proc. of AAAI'86*, Philadelphia, Pennsylvania, pp. 377-382. 1986.

[178] M. Vilain, H. Kautz, and P. v. Beek. “Constraint Propagation Algorithms for Temporal Reasoning: A Revised Report”, in *Readings in Qualitative Reasoning About Physical Systems*, D. S. Weld and J. d. Kleer (eds.). Morgan Kaufmann, San Mateo, California. pp. 373-381. 1990.

[179] M. Weiser. “The Computer for the Twenty-First Century”, *Scientific American*, pp. 94-100. 1991.

[180] W. Wilks. “A Preferential, Pattern-Seeking Semantics for Natural Language Inference”, *Artificial Intelligence,* vol. 6 (1), pp. 53-74. 1975.

[181] A. Wilson and A. F. Bobick. “Learning Visual Behavior for Gesture Analysis”, *Proc. of the IEEE-PAMI International Symposium on Computer Vision*, Coral Gables, Florida, pp. 229-234. November. 1995.

[182] A. Wilson, A. F. Bobick, and J. Cassell. “Temporal Classification of Natural Gesture and Application to Video Coding”, *Proc. of CVPR'97,* Puerto Rico, USA, pp. 948-954. 1997.

[183] A. Wilson and P. Wilson. *Theme Parks, Leisure Centers, Zoos, and Aquari*. Longman Scientific and Technical, Essex, United Kingdom. 1994.

[184] S. Wirag. “Modeling of Adaptable Multimedia Documents”, *Proc. of the European Workshop on Interactive Distributed Multimedia Systems and Telecommunications Services*, Darmstadt. September. 1997.

[185] L. Wolford and R. Schechner. *The Grotowski Sourcebook*. Routledge, London, England. 1997.

[186] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. “Pfinder: Real-Time Tracking of the Human Body”, *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 19 (7), pp. 780-785. 1997.

[187] J. Yamato, J. Ohya, and K. Ishii. “Recognizing Human Action in Time-Sequential Images Using Hidden Markov Model”, *Proc. of CVPR'92*, pp. 379-385. 1992.

[188] H. Zettl. *Television Production Handbook*, 4th ed. Wadsworth Publishing, Belmont, California. 607 pages. 1984.