Princess Cassie: An Embodied Cognitive Agent in a Virtual World

Vikranth B. Rao

Department of Computer Science and Engineering

State University of New York at Buffalo

Submitted to the

Advanced Honors Program at the

State University of New York at Buffalo

in partial fulfillment of requirements for graduation with

Advanced Honors

Advisors:

Dr. Stuart C. Shapiro

Josephine Anstey

April, 2004

Abstract

This thesis presents the development of an embodied cognitive agent called Princess Cassie and her implementation as a cognitive actor in a Virtual Reality drama. This work effectively bridges the gap between, and builds upon previous work carried out on cognitive agent modeling and development and work carried out on the development of dramatic agents that are richly expressive, believable and that portray characters with emotional depth. Princess Cassie is expected to serve as an ideal test framework, upon which an effective model of cognitive agency for dramatic and socially interesting agents can be built. Several issues central to the development of such embodied cognitive agency have been considered and implemented in this implementation. Princess Cassie's capabilities include: an ability to identify and process a rich array of perceptual information - both about the world and about its own embodiment; true multi-modal behavior involving several channels of asynchronous communication between the mind and the body; adherence to a tight script for performance; a sophisticated model for interrupt and contingency handling; achievement of goals by the formulation and execution of plans and ability to dissemble effectively in the face of unpredictability in the scene.

The development and subsequent implementation of Princess Cassie has allowed us to experiment with the various ways that an embodied cognitive agent can be used to effectively enhance a VR experience for a human user. This work has been carried out as part of a larger project which has the goal of eventually constructing an entire dramatic story around a human user, in which all the actors are computational agents sophisticated enough to draw dramatic responses from the user. The agent presented in this thesis therefore represents a first step towards accomplishing this goal.

Acknowledgements

I would like to thank my advisors Dr. Stuart C. Shapiro and Josephine Anstey without whose support, encouragement and advice this work would never have gotten done. I would like to especially thank Dr. Shapiro for being so generous with his time and support and for all the advice and encouragement he has extended me over the past few years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Inter-disciplinary research in the fields of cognitive science and computer science, in particular in the sub-discipline of artificial intelligence, has led to a great deal of interest in the area of cognitive robotics.

> Research in Cognitive Robotics is concerned with the theory and implementation of robots that reason, act and perceive in changing, incompletely known, unpredictable environments. [From the description of the AAAI 1998 Fall Symposium on Cognitive Robotics]

This document describes the development of an embodied cognitive robot, *Princess Cassie*, and her implementation as a cognitive actor in a Virtual Reality (VR) Drama. The work has been carried out under the guidance and supervision of Dr. Stuart C. Shapiro and Josephine Anstey at the University at Buffalo, and is part of a larger-project aimed at the "enhancement of VR with intelligent agents" [26].

The Virtual Reality (VR) Drama in which the agent is implemented is being developed by Josephine Anstey and her colleagues in the Department of Media Study at the University at Buffalo, using Ygdrasil [16], a VR authoring tool. Ygdrasil, developed by Dr. David Pape, one of Anstey's colleagues at the University at Buffalo, is "based on the OpenGL Performer scene-graph, and provides a framework for extension; application-specific modules (plug-ins) may be added to define behaviors for objects or characters." [4]. The work builds upon their previous experience developing large scale virtual environments and interactive fiction for VR [1, 3, 17] . The Virtual Reality drama is expected to serve as an interactive medium for storytelling and drama for human users to be immersed in. During the course of the drama, the user encounters

many challenges and interacts with various actors, that are expected to dramatically involve the user and engage him psychologically. The actors encountered by the user are all expected to be artificial cognitive agents [4]. In terms of agent development, this implementation imposes the requirements of multi-modal behavior, adherence to a tight narrative act, believability, contingency handling, natural language interaction with a human user, the ability to achieve goals by formulating and executing plans, reasoning and inference about knowledge and the ability to dissemble in the face of unpredictable behavior on the part of the user. In order for these requirements to be satisfied, several areas of interest in the development of embodied cognitive agency have been addressed and resolved in this implementation.

Specifically, Princess Cassie is expected to serve as an ideal test framework, upon which an effective model of cognitive agency can be built. This work on agent development, is built upon and extends previous work on developing autonomous agents and cognitive robots at the University at Buffalo by Shapiro and his colleagues in the SNePS Research Group (SNeRG) . SNeRG has been involved in building several implementations of a cognitive agent called "Cassie" (from which the current agent derives its name) [20, 22, 28], based on the broad framework of the Grounded Layered Architecture with Integrated Reasoning (GLAIR) [27]. GLAIR, also developed by Shapiro and his colleagues, is a generalized architecture for the modeling and development of cognitive agents. This document also describes work carried out on the development and implementation of modalities in Princess Cassie, which might be used towards the development of a generalized multi-modal cognitive architecture to be known as *M-GLAIR* [26] This document therefore describes the initial steps taken in what might ultimately become M-GLAIR.

The rest of this document provides a description of the development and implementation of Princess Cassie. In Chapter 2, I provide background for the current work in terms of agent development. In Chapter 3, I describe the GLAIR cognitive architecture. In Chapter 3, I also describe the implementation of modalities in Princess Cassie and discuss how this might be used to extend GLAIR. In Chapter 4, I describe the development of Princess Cassie as a cognitive actor. In Chapter 5, I discuss further considerations for agent development, which are fundamental to an effective implementation of the agent. In Chapter 6, I describe the implementation of Princess Cassie in the VR drama, in terms of the performance of her role. Finally, in Chapter 7, I conclude this document.

Chapter 2

Background

2.1 Introduction

Agent development, and the specialized area of cognitive robot development, has enjoyed significant attention in both the Artificial Intelligence community and the dramatic arts community. Several examples of agent development are readily available in the literature. These agents have used a wide variety of cognitive architectures, logical programming languages and intentional acting systems and have been implemented in varied applications - computational and dramatic. However, work on agent development in each of the disciplines of AI and dramatic art has traditionally been concerned with different areas of agent development stemming from differing motivations. Work in the AI community has traditionally been concerned with development of autonomy of behavior and with the modeling of cognitive agency in artificial agents. Work in the dramatic art community, on the other hand, has been concerned with the development of agents that are richly expressive and believable, and that are responsive to and can interact with human users. The current effort is an effort to bridge the gap between the two approaches to agent development through active collaboration between the two communities and by building upon previous work carried out in both communities. It is hoped that through the addition of expressiveness, believability and depth of character to a sophisticated autonomous artificial agent modeled along cognitive lines, truly remarkable and socially useful agents can be developed. This chapter will seek to provide appropriate background to this effort.

2.2 Cognitive Architectures

Princess Cassie is an embodied cognitive agent that implements cognitive agency as observed in human agents. Cognitive agency in human beings has long been studied as part of the field of Cognitive Science. Cognitive Science, an inter-disciplinary field spanning the disciplines of psychology, philosophy, linguistics, computer science, neuroscience and anthropology, is concerned with the study of the various parts that together result in cognition in human agents. The focus of cognitive science study therefore includes the study of such components of cognition as language, memory, perception, learning, acting, reasoning and information processing. Further, this field is also concerned with the development of computational models or architectures for artificial agents incorporating a range of these cognitive behaviors - reflecting a long held belief that the best form of study is modeling and replication of observed processes.

Several cognitive architectures have been proposed for the development and implementation of cognitive robots and agents. These include Soar [13] , GOLOG [14], Event Calculus [21] and GLAIR [27]. The current agent is based on the Grounded Layered Architecture with Integrated Reasoning (GLAIR). In an effort to properly place GLAIR among other comparable architectures, I provide a brief description of some other comparable architectures and programming languages in this section. Owing to the almost restrictively brief descriptions provided, the reader is encouraged to refer to appropriate sources for a more detailed description. I also discuss some of the advantages inherent in using GLAIR, rather than another comparable architecture. In Chapter 3, I will provide a comprehensive description and specification of the GLAIR architecture.

Soar is a cognitive architecture based on production systems and problem solving. At the core of the Soar architecture is an effort to develop a unified theory of cognition that tries “to find a set of computationally-realizable mechanisms and structures that can answer all the questions [one] might want to ask about cognitive behavior” [13]. The architecture is therefore an effort to develop and formalize a “fixed base of tightly-coupled mechanisms” [18] underlying intelligent behavior as engendered in human agents. Soar is an architecture that has become extremely popular and many agents in both the AI and dramatic art communities have been developed based on this architecture.

GOLOG is a logic programming language for agents implemented in dynamic domains. It is a cognitive robot language that allows for the description and maintenance of an explicit representation of the agent's world, and is based on a "formal theory of action".

> [GOLOG's] interpreter automatically maintains an explicit representation of the dynamic world being modeled, on the basis of user supplied axioms about the preconditions and effects of actions and the initial state of the world [14].

An agent developed using GOLOG is thus able to able to reason about actions in the world and maintain beliefs about the consequences and effects of acting in a dynamic world.

Event Calculus represents more of a formal acting system than a complete cognitive architecture. It is a "logic-based formalism for representing actions and their effects" [21]. Event Calculus therefore provides a formal approach to the task of developing and maintaining an explicit representation of actions and their effects in the world, using standard logical methods. It includes support for "actions with indirect effects, actions with non-deterministic effects, concurrent actions, and continuous change" [21].

The GLAIR architecture, on which the current agent is based, is similar in approach to the GOLOG and Event Calculus robot programming languages. However, neither GOLOG nor Event Calculus, provides a complete architecture for cognitive modeling, whereas GLAIR is a complete architecture for cognitive modeling, based on both logical reasoning and an acting language. Specifically, GLAIR is based on the Semantic Network Processing System, which is a suite of languages for knowledge representation and reasoning, intentional acting, rational inference and natural language interaction. By being based on SNePS, GLAIR "facilitates agent beliefs about acts, plans and what the agent, itself, has done, and reasoning about those beliefs" [26].

2.3 The SNePS system

Princess Cassie has been built using SNePS and Lisp, both in terms of GLAIR architecture being based on SNePS and the agent being specified in SNePS. SNePS is a propositional semantic network processing

system with built in functionality for acting, natural language interaction and belief revision. At its core, it is a system for building, manipulating and maintaining a semantic network of nodes where each node represents an entity that an agent based on SNePS is aware of. The system includes the following packages and languages:

SNeRE is the SNePS Rational Engine and is "a package that allows the smooth incorporation of acting into SNePS-based agents [29]. In a SNePS network, any action that needs to be performed by the agent is represented by an action arc pointing to an act node. SNeRE is able to recognize these act nodes as special nodes that correspond to actions that need to be performed. SNeRE includes an action executive and is built around an action queue. When an agent intends to perform one or multiple actions, each action is added to the action queue and the executive schedules the execution of each action in sequence.

SNIP is the SNePS Inference Package that allows for reasoning and inference to be carried out on information represented in a semantic network. Reasoning and inference can be carried out in a network by the definition and addition of specific rules using special rule nodes. Inference can be carried out through both rule-based and path-based inference on information represented in the network [29].

SNePSUL is the SNePS User Language. It "is the standard command language for using SNePS" [29]. It provides a Lisp-like interface for the construction, manipulation and maintenance of semantic networks, and for the information contained within them. The complete functionality provided by the SNePS system can be harnessed using SNePSUL and this can be done even from within Lisp code.

SNePSLOG provides a logic programming interface to SNePS similar to traditional logic programming languages such as PROLOG. It provides the complete functionality provided by SNePSUL with the added convenience of using standard predicate logic notation. SNePSLOG is particularly interesting for the purpose of this document, as it is the language in which all SNePS terms, plans, and constructs have been specified in the remainder of this thesis. A description of the syntax and semantics of the SNePS terms and constructs is presented in section 2.3.1 below so the reader can easily understand and parse any plans he might encounter in subsequent chapters of this document.

SNaLPS is the SNePS Natural Language Processing System. SNaLPS makes it possible for any agent built using SNePS, to exhibit natural language generation and comprehension.

SNeBR is the SNePS Belief Revision system. SNeBR "recognizes when a contradiction exists in the network, and interacts with the user whenever it detects that the user is operating in a contradictory belief space" [29].

2.3.1 SNePSLOG syntax and semantics

SNePSLOG is a logic programming interface to SNePS and as such uses a standard predicate logic notation. The syntax and semantics for the constructs that are interesting in the context of the current work are specified below (excerpted from descriptions in [29]).

Believe, *Disbelieve*, *snsequence*, *snif*, *do-all* and *do-one* are all represented in the SNePS semantic network as act nodes which can be intentionally performed.

Believe(X1) where X1 must be a proposition node. This causes X1 to be asserted and forward inference is done with X. If the negation of X had been asserted, then the negation is removed from the context under current consideration.

Disbelieve(X1) where X1 must be a proposition node. This causes X1 to be removed from the context under current consideration.

snsequence (X1, X2) or snsequence_n (X1, X2 ... Xn) where X1, X2 ... Xn are act nodes, causes the acts X1, X2 ... Xn to be performed in sequence.

do-all (X1) where X1 is a set of one or more act nodes, causes all of the act nodes to be performed in some arbitrary order.

do-one (X1) where X1 is a set of one or more act nodes, causes an arbitrary one of the act nodes to be performed.

snif (X1) where X1 is a set of guarded acts, and a guarded act is either of the form if (Y1 , Y2), or of the form else(elseact), where Y1 is a proposition node and Y2 and elseaact are act nodes. snif chooses

at random one of the guarded acts whose condition is asserted, and performs its act. If none of the conditions is asserted and the else clause is present, the elseact is performed.

The constructs *wheneverdo* and *whendo* are techniques for delayed act initiation. By using these constructs it is possible to entertain intentions to carry out certain acts only when specific propositions hold true in the network.

wheneverdo (X1, Y1) or whendo (X1 ,Y1): If a node of the form M:{wheneverdo (X1, Y1)} or of the form M:{whendo (X1, Y1)}, where X1 is a proposition node and Y1 is an act node, is in the network, and forward inference causes both M and X1 to be asserted, then Y1 is performed. The difference between wheneverdo and when do is that if the proposition p is disbelieved at any time and then believed again, the act controlled by whenever will be performed again while the act controlled by when will not.

GoalPlan, *ActPlan*, *AgentAct*, *MemberClass*, *AgentState*, *AgentLocation* and *ObjectEvent* are all standard SNePSLOG predicates that correspond to relations between entities in the semantic network. The syntax and semantics for each of these predicates is as follows:

GoalPlan(X, Y): The plan for achieving Goal X is to perform the act or set of acts represented by Y.

ActPlan(X, Y): The plan for performing the Act X is to perform the act or set of acts represented by Y.

AgentAct(X, Y): Agent X performed the action Y.

MemberClass(X, Y): X is a member of class Y.

AgentState(X, Y): Agent X is in state Y.

AgentLocation(X, Y): Agent X is in location Y.

ObjectEvent(X, Y): Event Y occurs with X as its object.

2.4 Related Agents

2.4.1 SNePS-based agents

Princess Cassie builds upon and extends previous work on autonomous agent and cognitive robot development carried out at the University at Buffalo, by members of the SNePS Research Group. There have been several software implementations and one hardware implementation of an embodied cognitive agent called "Cassie" [20, 22, 27, 28], based on the broad framework of the GLAIR architecture and on the SNePS system. Two such implementations are especially relevant to the development and implementation of Princess Cassie, because much of the work that will be described in this thesis has been inspired by and builds upon these implementations. These implementations are briefly described in this section with the intent of providing the reader with a better understanding of origins of, and motivations for, the current work.

The Foveal Extra-Vehicular Activity Helper-Retriever(FEVAHR) [22, 27]: This implementation of Cassie involved an embodied hardware implementation of Cassie and was developed for NASA, to serve as an artificially intelligent computational agent capable of exhibiting Foveal vision in performing certain tasks. FEVAHR Cassie, as this implementation is known, is the immediate pre-cursor to Princess Cassie and many of the mechanisms and routines developed for this implementation have been used in some form or the other, in Princess Cassie. The capabilities of Fevahr Cassie included "input and output in fragments of English; reasoning; performance of primitive and composite acts; and [Foveal] vision" [22], all of which proved useful for Princess Cassie. FEVAHR Cassie, based on GLAIR, also included mechanisms for seamless grounding of higher level mental entities, represented in its knowledge space, with the lower level routines that corresponded to these entities. The ability to carry out efficient grounding is essential for any embodied agent. Therefore, the mechanisms developed for the FEVAHR Cassie implementation to accomplish this have proven very useful during the development of Princess Cassie. FEVAHR Cassie also included a personal sense of time and routines for low-level bodily awareness.

Crystal Cassie [20]: This implementation of Cassie is more recent and the work on the development and implementation of Crystal Cassie is currently ongoing. This work has been carried out almost simultaneously with work on the development and implementation of Princess Cassie. However at the start

of the Princess Cassie project, there had been quite a lot of work accomplished already, and this work has greatly contributed to the development of Princess Cassie. Crystal Cassie is an implementation of Cassie in which Cassie's body and the world are simulated in Crystal Space, an environment for building 3-D games [20], just as Princess Cassie's body and the world are simulated in Ygdrasil. Crystal Cassie has been developed as part of an effort to develop a computational theory of identifying perceptually indistinguishable objects. The capabilities of Crystal Cassie, also based on GLAIR, have included identification and processing of a range of perceptual cues, vision, comprehension and generation of sentences in a fragment of English and the performance of a suite of actions, based on a small suite of primitive actions, through the intervention of a reasoning system.

These previous SNePS-based implementations have produced sophisticated cognitive agents that are capable of exhibiting a high degree of autonomy in behavior. However, neither of them have been built with the view of being richly expressive characters in a drama - characters that are able to respond appropriately and believably to human users. There are further shortcomings to these agents - like the lack of independent modalities in FEVAHR Cassie and the resultant lock-step behavior that this produces or the fact that extensive contingency handling has not been necessary for either of the agents. My work in the current implementation of Cassie as an embodied cognitive agent/actor in the VR drama has therefore been concerned with the development of artificial cognitive agents by extending these previous agents and also with the development of appropriate additions to the current GLAIR architecture. My work has also involved augmenting computational mechanisms with mechanisms for rich expressiveness, believability, and depth of character for the agents by building upon work carried out on developing agents for VR applications (described in section 2.4.2 below).

2.4.2 Agents for VR applications

Several research teams in the dramatic arts community have tried to enhance interactive stories and VR settings with artificially intelligent agents. The overriding motivations in these implementations has been the development of rich expressiveness, believability and depth of character, in combination with artificial intelligence mechanisms for their embodiment. Two such implementations are especially interesting because they provide concrete examples of efforts to enhance interactive fiction applications with artificial agents. A

brief description of the agents developed is provided in this section in an effort to introduce the user to the dramatic arts end of the spectrum in agent development.

The Mission Rehearsal Exercise [MRE] Project [10] provides an example of work that has made significant progress in enhancing an interactive fictional experience with intelligent agents. In this project, three artificial agents have been developed based on the Soar cognitive architecture (see section 2.2 for a brief description and appropriate sources for further reading on Soar). It is a training simulation developed for the Army and is based on a peace-keeping scenario. The scenarios of this project have been devised by Hollywood script writers and the purpose of the project is to develop simulations that are "compelling and effective as tools for learning" [10]. The agents play the parts of army soldiers or members of the local populace in the simulated world, and each agent is expected to realistically and believably serve as a training tool for a trainee being trained with the system. Each agent is therefore able to interact believably in "face-to-face dialogue" with the trainee, create moral dilemmas for the trainee and psychologically involve him in real life situations.

The Facade Project is an attempt to create a "fully-realized interactive drama - a dramatically interesting virtual world inhabited by computer-controlled characters, within which the user experiences a story from a first person perspective" [15]. This project therefore is quite similar to the current effort of developing *The Trial, the Trail,* which will be inhabited by computer-controlled agents such as Princess Cassie. The agents in Facade have been developed using a reactive planning language called ABL, a derivative of another language Hap. The fact that is truly interesting about this project is that:

> ABL [is a language that] has been designed specifically for authoring believable agents - characters which express rich personality, and which play roles in an interactive, dramatic story world [15].

Agents developed for this project have all been developed with the primary motivation of being believable, richly expressive and having a depth of character that makes for compelling social simulations.

Both the above projects are extremely interesting to our current work, because both of them present compelling example of work in enhancing interactive fiction with artificial agents. The agents developed for

these projects are very similar in dramatic emphasis to Princess Cassie. For example, the agents of the Façade project, having been developed using a specialized believability-driven language, provide insights into mechanisms that produce richly expressive and socially compelling agents. The agents of the MRE project have the added advantage of being based on a cognitive architecture, thereby making many of the routines used in their development extremely useful to Princess Cassie's development.

However, as mentioned earlier, these projects are still only primarily motivated by dramatic content and the development of agents that are socially compelling. It is only by being inspired by and building upon the projects presented in this chapter, can a truly interesting agent be developed that is both computationally sophisticated and socially compelling. The rest of this document provides a description of Princess Cassie, which has been an effort to bridge the gap in agent development between the AI and dramatic art communities.

Chapter 3

GLAIR and an extension

3.1 GLAIR

The current agent has been built based on the broad framework of the Grounded Layered Architecture with Integrated Reasoning (GLAIR). GLAIR is an architecture that allows for the modeling of agents that are cognitively motivated and for the development of agents that are intelligent and autonomous [27]. GLAIR is composed of several levels, each of which is concerned with the implementation of a separate layer of a cognitive model. A significant part of GLAIR is an approach "to anchoring the abstract symbolic terms that denote an agent's mental entities in the lower-level structures used by the embodied agent to operate in the real (or simulated) world." [27]. GLAIR therefore provides a framework and structure around which a cognitive agent can be modeled. In the past, several previous hardware and software implementations of cognitive agents have been built based on this framework, by Shapiro and his colleagues. Although GLAIR was not developed as part of the current project - it was developed by Shapiro and his colleagues - an adequate understanding of the architecture and its details is essential for understanding the agent that is the subject of this document. A description of the details of this architecture follows in this section.

Specifically, GLAIR consists of three layers - The Knowledge Layer (KL), the Perceptuo-Motor Layer (PML), and the Sensori-Actuator Layer (SAL). In previously implemented agents, the PML has been further sub-divided into three layers called PMLa, PMLb and PMLc respectively. The mind of a cognitive agent based on GLAIR is encompassed in the KL and the routines that translate the mental notions into embodied

functions are present in the top two layers of the PML - the PMLa and the PMLb. The bottom layer of the PML - PMLc - and the SAL are involved in the specific embodiment of the agent. The current document is concerned with the part of the architecture that is concerned with the development of a mind for an agent. Therefore, the work described is restricted to work on the KL, the PMLa and PMLb. The PMLc and the SAL have been implemented by Anstey and her colleagues. The reader is referred to [4] for a description. The architecture is illustrated in Figure 3.1.

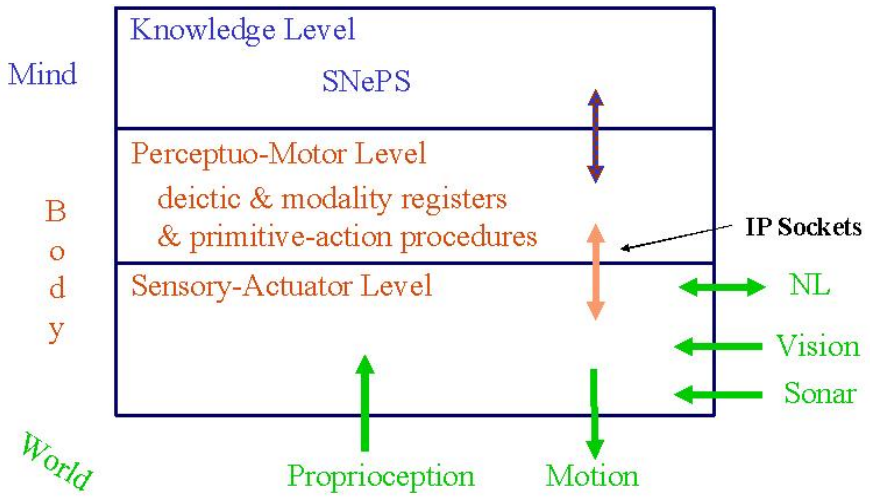

Figure 3.1: The GLAIR architecture [25]

1. The Knowledge Layer The Knowledge Layer (KL) is the highest level of the architecture and is the level in which intentional and "conscious" [27] reasoning takes place. It is also the most important layer of the architecture for the purpose of the discussions that follow in this document. It is the layer in which all reasoning, inference, planning and intentional processing takes place, and much of the work on the development of the current agent has been carried out in this layer. The KL is implemented with the SNePS Knowledge Representation and Reasoning system. The KL contains the mental representation of all the things that are in the agent's sphere of influence, including itself. All the entities and concepts about the world, that an agent implemented based on GLAIR is aware of, are represented as nodes in this mental representation. The mental representation built up by the agent, can therefore be conceptualized as a "knowledge space" [27], accessible by the agent that is

composed of a network of nodes and relations connected together using SNePS logical constructs. The knowledge space is available to all future reasoning, inference and natural language interaction. Further, any lower-level structures used by the agent's embodiment can also access the knowledge space allowing for efficient symbol grounding.

2. The Perceptuo-Motor Layer The Perceptuo-Motor Layer (PML) contains the "physical-level representation of objects" [7, 8, 9], that the agent is aware of. The PML is concerned with translating all mental notions into appropriate embodied functions. It is also the layer that contains the specification of "well-practiced" [27] behaviors that the agent is intuitively aware of. These behaviors are considered primitive at the level of the KL and can be executed without any conscious thought about the details of execution. Further, an agent built based on GLAIR, is explicitly aware of these behaviors and is able to reason and carry out inference about their execution. Traditionally, in previous GLAIR-based agents the PML has been further subdivided into three levels.

   PMLa: This is the highest level of the PML and it is the only level of the PML that communicates with the KL and has access to the agent's knowledge space. All the primitive act definitions are specified in this level and it is also in this level that incoming perceptions are added to the knowledge space. However, this level has no way to communicate with the embodiment of the agent. Whenever, an act needs to be performed, this level communicates with the PMLb, the next lower level, and when the embodiment perceives something in the world, the perception is passed to this level from the PMLb.

   PMLb: This is the middle level of the PML and is the level in which all translations between mental notions and embodied functions takes place. It is the lowest level of the architecture that is involved in the implementation of a mind for the agent - the reader is reminded that the mind of an agent based on GLAIR is encompassed in the KL and the routines that translate mental notions into bodily functions are present in PMLa and PMLb. This level has the ability to communicate both with PMLc, the lowest PML level, and PMLa, the highest level. At this level, all mental intentions are converted into embodied functions by generating appropriate calls that are sent through the PMLc to the embodiment. Further, all perceptions that come through the

PMLc from the embodiment are translated into a format that can be used in the knowledge space and this is transmitted to the PMLa , where the perceptions add to the agent's knowledge.

PMLc: This is the lowest level of the PML and is the level that is concerned with communicating with the SAL, in which the details of the embodiment are specified. The specification of this level depends upon the specific embodiment of the agent as the purpose of this level is to provide communication channels between the mental and physical parts of the agent.

3. The Sensori-Actuator Layer The Sensori-Actuator Layer (SAL) is the layer which controls all the physical parts (real or simulated) of the agent implementation, such as sensors, actuators and motors. This layer therefore contains the specifications of all routines that control the simulated Virtual Reality agent's presence in the world.

3.2 An implementation of Modalities

The GLAIR architecture allows for the development of sophisticated cognitive agents that are capable of exhibiting a high level of autonomy of behavior. However, it does have a few shortcomings. For example, any cognitive agent needs to be able to perform acts and process perceptions, in several different modalities at the same time, depending on the constraints imposed upon the agent by the world. However, the GLAIR architecture does not have any in-built support for such behavior. In previous agents developed based on GLAIR, acting and perceptions have been carried out along singular channels resulting in highly inflexible behavior. In the current implementation, since the driving motivation is to develop a cognitive actor, it was desirable to implement modalities which could be used in conjunction with the GLAIR architecture to produce true multi-modal behavior. A description of the design and implementation of modalities is presented in this section.

Princess Cassie is inherently composed of two sets of processes. One set is concerned with the mind of the agent (Set M) and is implemented in Lisp and SNePS. The other set is concerned with the embodiment of the agent (Set W) and is implemented in Ygdrasil. The two sets of processes are connected to each other using standard IP sockets [19] with the world acting as the server accepting and interacting with clients.

Set M, which we are concerned about, is composed of 6 different processes, each of which implements a different cognitive modality - both acting and perceptual. These include:

Modality Perception : This is a perceptual modality and is a one way stream of communication from the body to the mind of the agent. All perceptions about the state of the world are transmitted via this modality to the agent's mind.

Modality Self-Perception: This is also a perceptual modality and is a one way stream of communication from the body to the mind of the agent. However, this perceptual modality is concerned with perceiving the actions of the agent itself. All perceptions of changes in the state of the world caused by the agent are expected to be transmitted via this modality. At the current state of development, only perceptions of speech acts executed by the agent are transmitted, but this can very easily be extended to include all kinds of actions.

Modality Speech: This is an acting modality and is a one way stream of communication from the mind of the agent to its body. This modality is concerned with transmitting intentions for carrying out speech acts in the world, from Set M to Set W.

Modality Animation: This is another acting modality and is a one way stream of communication from the mind of the agent to its body. This modality is concerned with transmitting intentions for carrying out animated acts in the world from Set M to Set W.

Modality Natural Language In: This is a natural language modality and is a one way stream of communication from the body to the mind of the agent. This modality is concerned with identifying and transmitting to Set M, any natural language spoken by the user or by other agents in the scene.

Modality Natural Language Out: This is another natural language modality and is a one way stream of communication from the mind of the agent to its body. This modality is concerned with transmitting natural language strings, generated by the agent, to Set W, so they can be said by the embodiment of the agent.

During the development of Princess Cassie, each modality has been implemented in a separate thread, acting as an independent process, connected to Set W as a distinct client and using a separate IP socket.

When the agent is initialized, a new process is spawned for each modality, using Lisp's multi-processing functionality. Each process in turn, tries to bind with a port on the server machine - running the agent's body and the world - by querying for and receiving a port designation. In Set W, this port designation corresponds to the port number that the processes handling the embodiment of that modality communicate with. The modalities have therefore been implemented to run completely independent of each other and can send and receive strings from the world asynchronously. Further, in Set W, the processes that handle each of the modalities, are completely discrete, ensuring true asynchrony. This approach fits in very well with our efforts to model cognitive agency as the agent's cognitive abilities seem to emerge out of these distinct interactions with the world, along several modalities independent of one another. The various interactions between the mind and the embodiment of the agent are illustrated in figure 3.2.

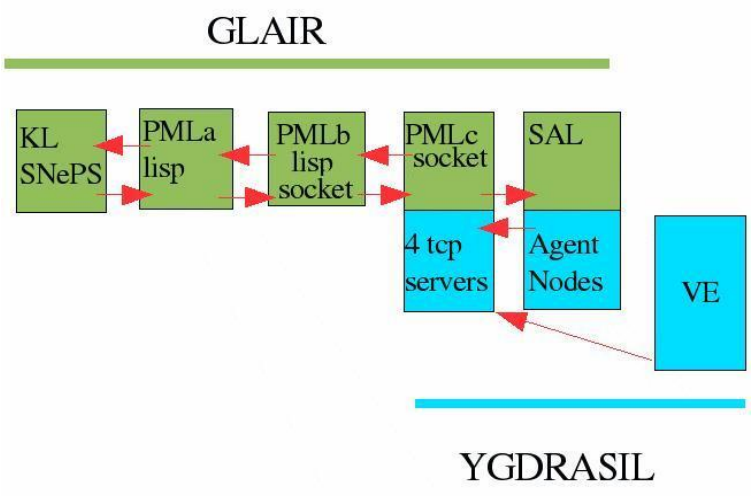

Figure 3.2: The interactions between the GLAIR layers and the world [2]

Finally, it is important to note that, the sockets serve both as streams of communication between the agent's mind and its body, and as effective barriers between the two sets of processes, allowing Set M to be implemented in a different environment without changing anything, or Set W to implement a different agent mind without major change.

3.3 Extending GLAIR

The design and implementation of the different modalities in the Princess agent makes it possible for the Princess agent to exhibit true multi-modal behavior. However, another detail that needs to be discussed before the discussion in this chapter can be complete, is one about how the implementation of modalities fits in conceptually, with the GLAIR architecture. In the current implementation, the addition of the different modalities has been accomplished in an ad-hoc manner with the different parts of the implementation of the modalities being carried out at different levels of the architecture, as required (see Figure 3.2). The initialization and binding of the individual process for each modality in Set M with the corresponding processes in Set W, is carried out at the PMLa level, so that they are accessible to the primitive action routines that use them to transmit intentions for the execution of speech and animation acts. However, the actual routines that carry out the initialization and binding repose at the PMLb level. Finally, the set of sockets and the set of processes in Set W, that the sockets bind to, together correspond to the PMLc level.

This description is completely ad-hoc in terms of the architecture because this is not a part of the architecture itself. It is entirely possible that another GLAIR-based agent might implement modalities in another way, as is the case in Crystal Cassie [20] (see 2.4.1). It is therefore visualized that as future work, by the larger group concerned with the development of "enhancement of VR with intelligent agents" [26], GLAIR will be extended to include functionality that would allow the design and specification of each modality as part of a generalized "suite of modalities".

> We, therefore, need to do a careful design of a modality both as a modular concept and a modular piece of software, so that an actor-agent can be constructed by specifying a suite of modalities, and one or more acts that are to be performed in each. We propose to make a suite of modalities a formal part of the Modal GLAIR (MGLAIR) cognitive architecture. [26]

It is further visualized that the design and implementation of modalities that I have discussed in this section, will possibly be the first steps in what will ultimately become the suite of modalities that are to be a part of the MGLAIR architecture.

Chapter 4

Agent Development

4.1 The Princess Environment

The agent is embodied in a virtual reality drama that is designed to be an interactive fictional experience for a human user. This work follows previous attempts to develop interactive dramatic experiences with intelligent agents.The drama, *The Trial, The Trail*, is designed to engage the user in not just a "first-person perspective", but also "first-person experience". "The basic conceit of the drama is a traditional quest or journey narrative" [4]. The user of the drama, henceforth referred to as "the User", is immersed in it and is a character that embarks on a journey through the narrative. Along the way the user encounters various distractions and challenges, and the user's psychological profile and dramatic response is assessed from these interactions.

One of the distractions faced by the user is a scene in which he encounters a Princess in a dilapidated factory. The role of this Princess is played by the cognitive agent/actor that is the subject of this paper. Henceforth, this actor agent will be referred to as "Princess Cassie" or as the "Princess". The layout of the scene is illustrated in Figure 4.1.

The scene starts with the user entering the factory and finding the Princess caged in a castle. From that point on, the Princess tries to get the user to free her from her cage, portraying the archetypical "damsel in distress". Opposing the user are villains, initially responsible for caging the Princess, who periodically

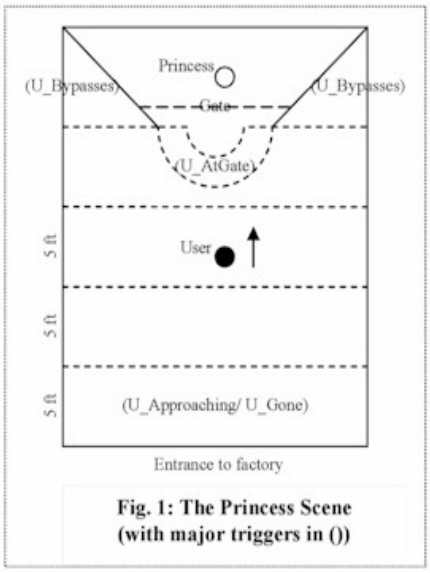

Figure 4.1: The Princess Scene (with major triggers in ())

threaten her and ultimately take her away. The presence of the villains, combined with periodic threats, is meant to emphasize the vulnerability of the Princess and to compel the user to help her escape. The scene contains several zones of narrative, depending on the actions of the user and his willingness to help the Princess. Several triggers are embedded in the scene to signal key events. These might include spatial information about the user or the villains, temporal information about the state of the narrative or behavioral information about the user and the villains.

As the user moves through the scene acting upon it, he trips various triggers which result in changes in the world's state. These changes in state are then passed on to the Princess agent as perceptual cues through perceptual modalities. Further, other information such as the villains threatening the Princess or the gate opening is also perceived by the Princess. This information is used by the Princess to advance her narrative in the scene and to gauge the behavior of the user. Ultimately, regardless of what the user does, the scene ends with the Princess being carried away by the villains of the scene, and the user being ridiculed for trying in vain to be a hero.

4.2 Princess Cassie's Identifiable Perceptions

Our discussion of the actual development of Princess Cassie needs to begin with her perceptual abilities. The agent, at the most basic level of behavior, is in a constant perceive-reason-act (if required) loop. The primary purpose in the implementation of Princess Cassie is to model a human-like cognitive agent. However, her specific implementation is in a definitely non-human simulated virtual world. This implementation in a simulated world allows the theoretical potential for the agent to have a "god's eye" view of the world around her and therefore have the capability of perceiving things not in her immediate perceptual vicinity. Such perceptual capability would not be in line with the motivations for the current effort. In the current implementation, Princess Cassie must be capable of perceiving only those perceptions that an ordinary human agent would be capable of perceiving. For this reason it is necessary, during the development of the agent, to specify exactly what perceptions the agent will be able to identify. A specification of the set of perceptions that she is capable of identifying and processing is presented in this section.

Princess Cassie's identifiable perceptions include two kinds of perceptions - perceptions of the world around her and perceptions of what she herself is doing. She perceives the world around her by identifying and processing a set of perceptual cues that are passed back to her from the world through a perceptual modality. These are specified and described in Table 4.1.

Princess Cassie is also capable of identifying and processing a complete set of self-perceptual cues, containing information about what her embodiment is doing. The inclusion of this capability is a direct consequence of an interesting observation of performance in human agents. When a human agent speaks a sentence, how is he aware of having spoken it? I believe that on some levels, the agent is aware of his speech through actually hearing it, or through the actual perception of the spoken speech. The same can be said of other animated acts and the actual performance of them. This ability to sense completion of intended acts, is extremely useful for entertaining knowledge and beliefs about successful execution of intentions and actions, advancement through plans, and achievement of goals (see Section 5.2 for a discussion).

Similarly, when Princess Cassie successfully completes execution of an action in the world, the information corresponding to the change of state that is manifested in the world, is transmitted back to the Princess'

- U_Approaching: The User is approaching the Princess (is within 10ft of the Princess).
- U_AtGate: The User is at the gate behind which the Princess is caged.
- U_Bypasses: The User has bypassed the gate and is in one of the corners to the left or right of the gate. (see Figure 4.1 for an illustration)
- U_Agrees2Help: The User agrees to help by pressing a button on his controller.
- U_Searching: The User is searching for a way to open the gate so he can free the Princess.
- U_Leaving: The User leaves the area (goes more than 10ft away from the Princess).
- U_Returns: The User returns after he seemed to leave the scene.
- U_Gone: The User has left the scene.
- BGs_Start_Threatening: The Bad Guys have started threatening the Princess.
- BGs_Stop_Threatening: The Bad Guys have stopped threatening the Princess.
- BGs_Coming: The Bad Guys are on their way to carry away the Princess.
- BGs_Grab: The Bad Guys grab the Princess.
- U_Using1Hand: The User is using only one hand in searching for a way to open the gate.
- U_NotUsingHands: The User is not searching for a way to open the gate.

Table 4.1: Princess Cassie's Identifiable Perceptions

mind through a self-perceptual stream - akin to the human user's actual perception of the spoken speech. This information then becomes knowledge about the execution of the act, through the addition of the information to the "knowledge space" [23, 29] of the Princess. The knowledge can in turn be used for reasoning and for drawing inferences about future acts and event. Thus, just as in the case of the human agent we are trying to model, Princess Cassie entertains beliefs about what she has successfully performed, after performing it.

4.3 Acting

Our discussion of the development of the Princess agent continues with a discussion of her ability to act in the world she is embodied in. Any human-like, rational cognitive agent needs to have the ability to schedule and carry out intentional actions, based on her beliefs, so she can carry out her plans or achieve her

goals. An obvious observation in human agents, is that they are capable of performing some actions which are very intuitive acts, and which require no thought about the specific details (moving a hand, speaking etc.). Additionally, they are also capable of carrying out actions that are more complex and which would require some thought about the specific details (greeting a person, searching for something etc.). Finally every human agent is capable of carrying out certain acts which are internal to his embodiment and which are not manifested as any physical act (thinking, memorizing etc.). The requirement for acting ability and the observation of acting ability in human agents has been reflected in the design of acting ability for the Princess agent.

Princess Cassie is capable of acting across two action modalities and is also capable of carrying out acts which are internal to her embodiment. As in the case of human agents, Princess Cassie is able to carry out both intuitive and complex intentional acts. In the following discussion, the terms “intuitive”, “internal” and “complex” refer to the kinds of actions described in the discussion on possible human agent actions above. A description of both kinds of actions in the Princess Implementation follows in subsections 4.3.1 and 4.3.2.

4.3.1 Primitive acts

The specification and design of intuitive acts is part of the general cognitive architecture - GLAIR - upon which the current agent is based. In GLAIR, such intuitive acts are referred to as “primitive acts” [27] and these acts are defined as established routines in the Perceptuo-Motor Level of the architecture. These routines provide specifications for basic behaviors that can be executed without any conscious reasoning or thought about the details. Further, the architecture also makes it possible for the agent to be aware of these behaviors, reason about them and intentionally perform them. At the current stage of development, all primitives are directed towards the user. For example, the look primitive act causes the agent to look at the user. In the future, it is visualized that the specification for each primitive will include a parameter that allows the agent to direct the primitive at any entity in the world that it is aware of. The complete set of primitive actions that the Princess agent is capable of is specified and described here.

(Say “something”): The say primitive act is a speech act that is handled by sending the string that needs

to be spoken across the speech modality to the embodiment of the agent. When executed, the string represented by “something” is said by the agent in the world.

Attention: The Attention primitive act is handled by passing the word attention to the embodiment of the agent. When executed, the agent pays attention to the user.

Jump: The Jump primitive act is handled by passing the word jump to the embodiment of the agent. When executed, the agent jumps.

Wave: The Wave primitive act is handled by passing the word wave to the embodiment of the agent. When executed, the agent waves at the user.

Look: The Look primitive act is handled by passing the word look to the embodiment of the agent. When executed, the agent looks at the user.

Grabbed: When executed, the agent expresses a reaction to being forcibly grabbed by the villains in the scene.

Interact: The Interact primitive act is handled by passing the word interact to the embodiment of the agent. When executed, the agent expresses interest in the user’s actions.

Cower: The Cower primitive act is handled by passing the word cower to the embodiment of the agent. When executed, the agent cowers in fear (See Figure 4.2).

DoNothing: This primitive action represents a “no-operation” and is useful for the Princess to be able to do nothing at various times during the performance of her role as an actor.

GoOffStage: This primitive action results in the agent shutting down all her processes. When executed, the agent stops performance.

The set of internal acts that Princess Cassie is capable of, is also a special set of primitive acts. These internal acts are all established routines that are not passed across to the embodiment of the agent, rather, they are restricted to the mind of the agent (See Section 5.2.2 for a discussion of how they are actually used). Like all primitive acts, Princess Cassie is aware of these behaviors, is able to reason about them and intentionally perform them. These include:

Figure 4.2: The Princess agent executing a cower act [2]

Memorize: The Memorize primitive act enables the agent to memorize a previously carried out action.

Recall: The Recall primitive act enables the agent to recall a previously memorized action.

WM: The WM primitive act is used by the agent implementation to subconsciously store previous actions into a temporary buffer. Every intuitive primitive action that is not an internal act has the side-effect of calling this act.

4.3.2 Complex acts

Primitive acts are the smallest units of acting available to the Princess agent. These acts provide for restrictively simple, "well-practiced routines" [27] for behavior and acting. Further, each primitive act only applies to its corresponding modality, and if plans were built up for the agent with these primitive acts as the building blocks, the agent would be able to act in only one modality at any instant of time. Since our driving motivation in the current project is to model a human-like cognitive actor, this would not fit in well with our requirements. This is because, for a true cognitive agent, it is desired that the agent be able to perform acts in several different modalities at the same time, depending upon the constraints imposed upon the agent by the world. Such behavior is often observed in human agents during the performance of "complex" acts. For example, the agent should be able to execute the look primitive across the animation modality, while

executing the say primitive with the message “Hello, Hello” across the speech modality, while greeting the user. Performing the say primitive after the look primitive would not be useful, as she might be looking at something totally different at that time instant.

The implementation of independent modalities (see Chapter 3.2 for a discussion of this implementation) in the Princess agent makes it possible for the Princess agent to exhibit such multi-modal acting. Since Princess Cassie has been implemented as a cognitive actor that has to act according to a specific script, a set of complex acts has been specified for her, based on the script. Each complex acts is a plan of action using a set of primitives, across multiple modalities, which are executed either at the same time, or in sequence, or at random depending upon the requirements of the script. It is these complex acts, and not the primitive acts, that serve as the building blocks upon which the plans for her performance are built. In the remainder of this document, these complex acts will be referred to as “primitive performatives” since they serve as the primitives at the level of the performance. The primitive performatives are specified and described below (using SNePSLOG [24, 29]). In the following complex act plans, as mentioned in the section on primitive actions (Section 4.3.1), all actions are directed at the user.

The performative **sayhello** combines the say primitive, across the speech modality, and the look primitive, across the animation modality. When executed, the agent looks at the user and says “Hallo, Hallo”.

```
ActPlan(sayhello,
        do-all({look,
                say(Hallo, Hallo)})))
```

The performatives **callover**, **thankful**, **beckon**, **beckoncloser canttalkloudly** and **introbabelamb** combine the say and wave primitives. When executed, the agent waves at the user and says the appropriate speech string.

```
ActPlan(callover,
        do-all({wave,
                say(We’re all over here)}))

ActPlan(thankful,
        do-all({say(Thank goodness you’ve come),
                wave}))
```

```
ActPlan(beckon,
        do-all({say(Please come nearer),
                wave}))

ActPlan(beckoncloser,
        do-all({say(Even nearer),
                wave}))

ActPlan(canttalkloudly,
        do-all({say(They'll hear me if I talk too loudly),
                wave}))

ActPlan(introbabelamb,
        do-all({say(This is my babe, and here is my little lambkin),
                wave}))
```

The performatives **explthatcaged**, **accepthelp**, **explhow2help**, **duringsearch**, **usingonehand** and **notusinghand** combine the say and interact primitives into different action plans. When executed, the agent pays attention to and responds to the user and says the appropriate speech string.

```
ActPlan(explthatcaged,
        do-all({interact,
                say(We're locked in here)}))

ActPlan(accepthelp,
        do-all({say(You can open the gate from your side.),
                interact}))

ActPlan(explhow2help,
        do-all({say(You have to touch one of the globes in front of it.),
        interact}))

ActPlan(duringsearch,
        do-all({do-one({say(I know they open it by touching one of
                                                        the globes.),
                        say(I don't think its that one.),
                        say(I'm not sure which globe it is.)}),
                interact}))

ActPlan(usingonehand,
        do-all({do-one({say(Put your hands on different globes.),
                        say(Try using both hands.)}),
                interact}))
```

```
ActPlan(notusinghand,
        do-all({do-one({say(Touch the globes.),
                        say(You have to touch the globes.)}),
                interact}))
```

The performatives **duringleaving**, **callback**, **stopleaving**, **feeldoomed**, **exprpanic**, **exprjoy** and **givethanks** combine the say and jump primitives into different action plans. When executed, the agent jumps urgently and says the appropriate speech string.

```
ActPlan(duringleaving,
        do-all({do-one({say(Please come back!),
                        say(OH NO! Please don't leave us! Please help!)}),
                jump}))

ActPlan(callback,
        do-all({say(Please come back!),
                jump}))

ActPlan(stopleaving,
        do-all({say(OH NO! Please don't leave us! Please help!),
                jump}))

ActPlan(feeldoomed,
        do-all({say(We're doomed! Doomed!),
                jump}))

ActPlan(exprpanic,
        do-all({say(Oh! They're coming!),
                jump}))

ActPlan(exprjoy,
        do-all({say(Hooray Hooray!),
                jump}))

ActPlan(givethanks,
        do-all({jump,
                say(Thank the Lord!)}))
```

The performatives **scream**, **franticforhelp**, **preventabandon**, **pleadfrhelp** and **exprdanger** are single modality performatives, acting across the speech modality. When executed, the agent says the appropriate speech string. In this case, one might wonder what the advantage of using such performatives

is, when all they do is call one single-modality primitive act. One might reasonably argue for the use of the say primitive itself rather than renaming it as a performative and then using that performative in the plan. The answer is that although these are implemented as single act performatives at the current state of development, such a specification of these performatives allows for easy extension into the multi-modal form. Further this design also allows for conformity for plan specification, allowing the plans to be completely composed of performatives (see Chapter 6 for a description of the agent's plans).

```
ActPlan(scream,
        say(AGGGRHH))

ActPlan(franticforhelp,
        say(Help Help))

ActPlan(preventabandon,
        say(Don't abandon us - please))

ActPlan(pleadfrhelp,
        say(Please will you help us escape?))

ActPlan(exprdanger,
        say(We're in terrible danger))
```

The performative **duringthreat** randomly picks between an attention and a cower primitive act and is a single-modality performative across the animation modality. When executed, the agent either shows attention to the user or cowers in fright.

```
ActPlan(duringthreat,
        do-one({attention,
                cower}))
```

The performative **exprpain** combines the grabbed and say primitives. When executed, the agent is grabbed by the villains of the scene and she screams in pain.

```
ActPlan(exprpain,
        do-all({grabbed,
                say(AHHHHH)}))
```

The performative **resptothreat** combines the say and cower primitives. Further, the speech string is randomly chosen from a set of choices. When executed, the agent says a randomly picked speech string and she cowers in fright.

```
ActPlan(resptothreat,
        do-all({do-one({say(What are they planning?),
                        say(Oh - my poor baby and lambie),
                        say(I hope they kill us quickly),
                        say(They'll kill us),
                        say(I'm so scared)}),
                cower}))
```

Chapter 5

Further considerations in Agent Development

5.1 Introduction

In Chapter 4, I presented a description and specification of the basic building blocks which were used to build the Princess agent. In the actual implementation of the Princess agent as a cognitive actor in the VR drama, all these building blocks are structured into a complex framework of goals, plans and intentions that together specify the narrative arc that the agent needs to perform through the course of her part in the scene. This structure of goals, plans and intentions can be conceptually understood as a description of the scene script in a language that is understood by the agent. In chapter 6, I will present a specification and description of the actual plans that are used by the agent in the performance of her role. However, for a proper understanding of the plans, a description and discussion of several considerations fundamental to our current effort of modeling cognitive agency, is essential. This is especially important because in the final performance plans, the various building blocks covered in the previous chapter are augmented with specific mechanisms that reflect these considerations. This chapter is concerned with describing and specifying these mechanisms.

5.2 A discussion of Time and a notion of Memory

Previous implementations of agents similar to Princess Cassie have included a personal sense of time [11, 22, 27], so that the agent can have beliefs about what she has done in the past as well as what she is currently doing. A personal sense of time is extremely useful for a cognitive agent, because having such a sense of time provides a reference metric which can be used by the agent to orient itself in performance time. Knowledge about past completion of acts, goals and intentions allow the agent to further its own act and goal plans. Further, the agent is also able to use this knowledge in reasoning and inference [11].

In the current implementation, however, I have not included a personal sense of time in the design of the Princess agent. I believe that such a formal representation of time for an agent such as the Princess, which is quite basic in the scheme of things, is unnecessary. For the most part, the Princess agent has no requirement which makes it necessary for her to entertain beliefs about past acts. This lack of personal time leads us to several issues that have to be considered at the developmental level, before the Princess agent can be effectively implemented. A description of these issues is provided in this section.

5.2.1 An issue of synchronization

The first issue was one of synchronization between the mind of the agent and its physical embodiment in the Virtual world. Most of Princess Cassie's plans for performance in the scene, contain intentions for the performance of multiple acts in succession. Now, in an acting system developed using the SNeRE Rational Engine, the intention to execute an act corresponds to the addition of the action to a queue of actions waiting to be executed. In a traditional agent, in which both the mind and the body are implemented on a single system, each act in the queue is carried out by executing the lower-level routine in the SAL, which when executed will result in the act being performed [29]. When the routine corresponding to the current act has finished execution, the SNeRE executive initiates the similar execution of the next act in the queue. However, in the current agent, due to the separation between the mind and the body, the SAL and the lower-level routines are on a separate machine across a socket. At the level of the mind, the lowest level is the function that sends an appropriate string across a specific socket, and the mind has no conception of the actual performance of the acts it intends to perform. In the current agent, sending a request across a socket

for the execution of an act, is indistinguishable from the successful completion of the act. Therefore, during the execution of plans, the mind is able to process intentions and send requests for execution of actions across the sockets much faster than the virtual embodiment can perform them. The lack of a personal sense of time further exacerbates this issue, because the agent is unable to read anything into the lack of completion of previous intended acts. As far as an agent without a personal sense of time is concerned, there is no past and there have been no past acts. Unchecked, this would result in a second or third act interrupting the first one and the actual act performed being a non-deterministic choice between them.

It was clear from an examination of this issue, that there needed to exist a mechanism at the level of the mind, which ensured that no subsequent act could be initiated unless there existed sufficient belief that the current act had been completed. There have been attempts to implement such mechanisms in other agent development languages such as Hap (or ABL), by the specification of each step in each plan within an explicit success/failure framework [15], with no subsequent step being executed if the current step cannot be successfully executed. It is however unclear if the agents specified using ABL are implemented on a single machine or, as is the case in the Princess implementation, on multiple machines. If ABL-based agents are indeed implemented on multiple machines with a separation between the mind and the body, then I would think that the attempt will fail since a similar problem would arise for the same reasons. In the current agent, such a mechanism was implemented using a notion of a Cascade [12]. A cascade is a sequence of acts in which no act in the sequence may be performed before the goals for which the earlier acts in the sequence are performed are achieved. For Princess Cassie, a notion of this was implemented by including explicit checks at every step of every performance plan for the completion of the previous step. The reader is now reminded of the self-perception modality (described in Section 4.2) which allowed the Princess agent to entertain beliefs about the acts she has successfully performed, after performing them, through the actual perception of the execution of the action in the world. The usefulness of the knowledge represented by these beliefs now becomes clear, because this knowledge can be used to explicitly check the completion of a step in a performance plan. Thus, at every step of a performance plan, the Princess checks, through self-perception, if she has completed execution of the previous step, and if so, she continues with the execution of the current plan. A fragment of a performance plan which illustrates this mechanism is shown below (using SNePSLOG [24, 29]).

```
wheneverdo(AgentAct(b2,approaching),
           snsequence(sayhello(b2),
                      believe(wheneverdo(AgentAct(b1,
                                                  say(Hallo,Hallo)),
                                         callover))))
```

In this plan fragment, the term (b1) refers to the Princess agent and the term (b2) refers to the user. When Princess Cassie perceives a person (the user) approaching her, she performs the performative **sayhello** which involves her looking at the person and saying the speech string “Hallo, Hallo”. Now, the interesting part of this example is that the next step of the plan - her performance of the performative **callover** - is executed only after she perceives that she has said the words “Hallo, Hallo”. At the current state of development, she only checks her speech acts for completion, but this can very easily be extended to include explicit checks across every modality.

5.2.2 An issue of Memory

Princess Cassie is basic enough not to warrant a full-blown sense of time. However, during the development of the agent, it quickly became quite apparent that she did need to have at least a short-term sense of what she has just done. The issue at hand was, therefore, one of developing a short term memory for the Princess agent without having the luxury of a personal sense of time. The issue of synchronization discussed in Section 5.2.1, could also have benefited from such a short term memory. However, development of short term memory was more directly required to handle the case in which Princess Cassie’s performance of a plan or the achievement of a goal is interrupted by a perception that requires immediate handling. (for example: If the villains (”Bad Guys”) threaten her at any time in the scene, she needs to stop whatever she is doing, show fear of the Bad Guys, and then continue what she was doing.) Such a performance would entail that she have the ability to remember what she was doing before she started handling the interrupt, so that she can continue with what she was doing after handling the interrupt.

In addressing this issue, two structures were added to the internal specification of the mind of the agent. These structures - called *WM* and *STM* - correspond, in a cognitive sense, to a conception of a Working Memory and Short Term Memory in human agents [5, 6]. In the agent specification, *WM* is a buffer,

while *STM* is a standard last-in-first-out stack. When executing an embodied act across the speech or animation modalities, Princess Cassie stores the speech act or the animation act that she has just performed in *WM* by executing the WM primitive. Each time a new act is performed the buffer is over-written with the new information.

Now during Princess Cassie's performance as an actor, should she perceive anything that causes her to interrupt her current execution path and execute a different plan, she performs a sequence of internal acts which enable her to handle the interrupting or contingency case and return to the original plan after that. This is best illustrated by considering the plans specified below (using SNePSLOG [24, 29]).

```
wheneverdo(AgentAct(b3,start_threatening),
           snsequence_3(memorize,
                        believe(AgentState(b1,contingency)),
                        duringthreat))

wheneverdo(AgentAct(b3,stop_threatening),
           snsequence(resptothreat,
                      believe(wheneverdo(AgentAct(b1,
                                                  say(I'm so scared)),
                              recall)))))
```

In this example, when the villains start threatening the princess, she reacts by first performing the memorize primitive which has the effect of pushing the information in the *WM* buffer onto the *STM* stack. She then believes that she is in a state of contingency and carries out the **duringthreat** primitive which involves her being silent during the threat and paying attention to the bad guys. Next, when she perceives that the villains have finished threatening her, she carries out the performative **resptothreat** which involves her showing fear of the villains. Finally, when she perceives that she has performed appropriate interrupt handling by saying "I'm so scared", she performs the recall primitive which pops the memorized information from the *STM* stack and pipes this into her self-perception modality, allowing her to remember where she was. A point to note in this process is that the Princess recalls what she last did in the original plan and not what she was about to do. This might seem counter-intuitive, but is easily explained by remembering that each performance plan is a cascade of intentions in which each step is performed only if there exists sufficient belief that the previous step has been completed. When a contingency occurs during the performance of a plan,

the plan is interrupted at a particular step and subsequent steps in the cascade cannot be performed. After handling the interrupt, by recalling what she last did in the original plan and piping this information into the self-perception modality, the Princess causes the continuation of the original performance plan. The actual process involved in stopping the original performance plan when an interrupt happens, and subsequently continuing the plan when the interrupt has been handled is described and discussed in detail in section 5.4 below. The specific implementation of the notion of memory that has been presented in this section, will also be presented and discussed in section 5.4. For now, all that I hope to help the reader understand, is that when an interrupt occurs in the scene, the notion of memory described in this section is used by the agent to memorize her place in the scene narrative where she is interrupted, and subsequently to recall by self-perception that original place after the interrupt has been appropriately handled.

5.3 Higher Level Plans and Princess Cassie's Beliefs about them

As described in section 4.3.2 and specified in section 6.1, the plans for the performance of the Princess agent's role have the performatives as the basic building blocks. To refresh the memory of the reader, each performative is a plan of action which combines one or more primitive actions across one or more modalities. The use of these performatives allows for the kind of flexible, multi-modal behavior that we would expect a cognitive agent to exhibit. The advantages of using these performatives are further maximized in the current implementation by structuring the performatives into a complex system of goals, plans and intentions that are cascaded together (see Section 6.1). The ability to extend the use of these performatives in this manner, combined with the basic flexibility and multi-modal behavior that the performatives make possible, makes them an ideal unit of behavior. However, the use of performatives in the current agent architecture, which considers the primitive actions to be the basic building block, is not trivially accomplished and several issues need to be considered.

In the current agent architecture - which is directly based upon GLAIR - Princess Cassie is consciously aware of her primitive actions and is able to entertain beliefs and reason about them [27]. This is possible because in the current architecture, the primitive actions - which are defined as routines in the agent specification - form a part of the agent's "consciousness of its body".

> The PML [the middle layer of the GLAIR architecture] also contains routines for well-practiced behavior, including those that are primitive acts at the KL, and other subconscious activities that ground Cassie's consciousness of its body and surroundings. [27]

Cassie-like agents can therefore "think" about the execution of their primitive actions, in the sense that the execution of primitive actions causes beliefs to be asserted automatically in their knowledge space. In the implementation of Princess Cassie, however, primitive actions have been buried under multiple layers of abstraction in the form of performatives and higher level multiple-performative plans. Without explicit specification, no automatic mechanisms existed in the agent architecture by which Princess Cassie would believe that these higher-level plans and performatives were done. This was an issue that needed rectification because in a real unconstrained world, when interacting with an unpredictable user, it is quite possible that she would need to change plans, handle contingencies or abort plans, depending on what the user is doing. The ability to exhibit such behavior, in turn, was contingent on her ability to entertain appropriate beliefs about higher level structures.

To better illustrate this problem, let us consider the case where Princess Cassie is engaged in executing a plan which has the goal of getting the user to the target area in front of the gate (see Figure 4.1 for an illustration of the scene and Section 6.1 for the exact plan specification). In the ideal scenario, the goal of getting the user to the target area in front of the gate would only be achieved after she has finished saying and doing everything she needs to do, to achieve it. However, the goal state is dependent on the actions of an autonomous agent - the user - whose actions do not have to correspond to the Princess agent's performance. In user tests, it was observed that it is often the case that the user, seeing the Princess in the distance, will most likely just head in the direction of the gate and reach the target area much before the Princess has finished doing and saying all that she needs to say to get him there. In the version of the agent in which this problem had not yet been resolved, this then led to the scenario where Princess Cassie was simultaneously executing two different plans - one for explaining her predicament to the user since he is at the target (see Section 6.1 for plan specification), and the other for still trying to coax the user to the gate - because she had no beliefs about the intermediate steps in the high-level plans. Princess Cassie initially started executing the first plan, because at the time of instantiation of the plan, the goal of getting the user to the target had not

been achieved. Since she had no intermediate beliefs, she had no reason to stop execution of the cascaded steps in the plan, when the goal got asserted midway through the plan.

This problem was resolved in the Princess Agent by including mechanisms at every step of every performance plan, which enabled the agent to intentionally check her beliefs and goals. Should a goal for a plan she is executing become asserted at any time, these mechanisms allow her to realize that she no longer needs to continue the plan and she stops execution of the plan, by executing a DoNothing primitive. A fragment of a performance plan , which includes this mechanism, is shown below (using SNePSLOG [24, 29]).

```
wheneverdo(AgentAct(b1,
                    say(Hallo,Hallo)),
           snif({if(AgentAct(b1,greet(b2)),
                    nothing),
                 else(callover)}))
```

This is a fragment of a plan which has the goal of greeting the user and is executed when Princess Cassie perceives a person - the user - approaching. The point to note in this example is the SNeRE construct snif [29]. As specified in this fragment, after the execution of the step which involves the Princess saying “Hallo, Hallo”, the snif construct checks if the goal proposition of Princess Cassie greeting the user has been asserted, and if so, the Princess stops execution of the plan by carrying out the performative **nothing**, which in turn calls the DoNothing primitive. Only if the goal proposition is not asserted, does the agent execute the next performative - **callover**. In future work, it is envisioned that a construct will be added to the rational acting language, which would allow this to be carried out implicitly without explicit checks at every step of every plan.

5.4 Prioritized Acting

The description of the mechanisms developed for handling high-level plans, presented in section 5.3 applies to all performance plans in the Princess Agent implementation. However, plans for contingency or interrupt handling are special cases and the description provided does not adequately cover such plans. Contingency or interrupt handling plans are special plans because they have no goals which can be used to check for completion. These plans are purely reactive in nature and as such also need to be considered as plans with

the highest priority. Further, the agent needs to treat these plans as only temporary interruptions, after which she needs to resume whatever performance plan she was in the process of executing before the interrupt occurred. This means that the initial plan which the contingency handling plan interrupts should not just be abandoned, as was the case with the performance plans (see Section 5.3 for a complete discussion of the mechanisms developed for handling performance plans).

In the current agent, interrupt handling is carried out using a notion of memory. In section 5.2.2, I presented a discussion of the details of interrupt handling in the Princess agent which involved her interrupting her normal execution of the scene narrative, handling the interrupt, and then recalling by self-perception, her prior place in the scene. That description while being theoretically valid for interrupt handling by itself, requires further elaboration. To illustrate the need for further elaboration, let us consider, once again, the case where Princess Cassie is engaged in carrying out the plan which has the goal of getting the user to the target area in front of the gate. Let us further consider that a contingency of the "Bad Guys" threatening occurs sometime midway through the execution of the plan. Following the discussion in section 5.2.2, Princess Cassie carries out a memorize primitive to memorize her place in the scene narrative, and begin handling the interrupt. However, unless specified, there is no mechanism that prevents the original plan (the plan for getting the user to the gate), from continuing while she was handling the interrupt. Since the plans are specified as cascaded chains of actions, the next step of the cascade in the original performance plan would continue after the current step was completely executed. Princess Cassie had no high level knowledge about the state of her performance which prevented her from executing both the regular performance plan and the contingency plan simultaneously. Such behavior was obviously undesirable.

Now let us consider a human actor in the same situation. Since the driving motivation in the current project is to model one, an observation of how human agents carry out interrupt handling would be useful. It is my belief that a human agent, when presented with a contingency, has the ability to entertain a high level belief that he is in a state of contingency while the contingency is being handled. Such a state would be considered special, would have the highest priority for execution and no regular plans of performance would be executable while in this state. I have used a similar notion in resolving this problem in the Princess Cassie implementation To better understand the exact problem resolution implemented, let us consider the

following interrupt-handling plan (specified using SNePSLOG [24, 29]).

```
wheneverdo(AgentAct(b2,leaving),
           snsequence_4(memorize,
                        believe(AgentState(b1,contingency)),
                        duringleaving,
                        believe(wheneverdo(AgentAct(b2,returns),
                                recall)))),
```

As described in section 5.2.2, upon encountering a contingency, the Princess agent memorizes her place in the scene narrative and starts handling the interrupt. However, as illustrated in this plan, the first step in the interrupt handling routine is to assert the belief that she is in a state of contingency. This belief that she is in a state of contingency is a high level belief that the agent can subsequently use to limit the original performance plan from continued execution, by executing the **nothing** performative at that level. A complete performance plan, which includes this mechanism, is shown below (using SNePSLOG [24, 29]).

```
all(p)(MemberClass(p,person)
=>ActPlan(explainpred(p),
   believe(
    wheneverdo(AgentLocation(b2,AtGate),
               snsequence_5(believe(AgentAct(b1,greet(b2))),
                            introbabelamb,
                            believe(
                             wheneverdo(AgentAct(b1,
                                                 say(This is my babe,
                                                     and here is my
                                                     little lambkin)),
                                        snif({if(AgentState(b1,contingency),
                                                 nothing),
                                              if(AgentAct(b1,
                                                          explpredicament(b2)),
                                                 nothing),
                                              else(explthatcaged)}))),
                            believe(
                             wheneverdo(AgentAct(b1,
                                                 say(We're locked in here)),
                                        snif({if(AgentState(b1,contingency),
                                                 nothing),
                                              if(AgentAct(b1,
                                                          explpredicament(b2)),
                                                 nothing),
                                              else(exprdanger)}))),
                            believe(
                             wheneverdo(AgentAct(b1,
                                                 say(We're in terrible
                                                             danger)),
                                        snif({if(AgentState(b1,contingency),
                                                 nothing),
                                              if(AgentAct(b1,
                                                          explpredicament(b2)),
                                                 nothing),
                                              else(pleadfrhelp)}))))))))
```

In this plan, when the Princess perceives that the agent is at the gate in the scene, she believes that she has greeted the user and then performs the introbabelamb performative. She then performs a sequence of three believe actions which apparently seem to just set up a parallel set of wheneverdo waits. However, what is important to note in this plan is that the performance of the **introbabelamb** performative involves the Princess saying the string "This is my babe and here is my little lambkin", the performance of the **explthatcaged**

performative involves the Princess saying the string “We’re locked in here” and the performance of the **exprdanger** performative involves the Princess saying the string “We’re in terrible danger”. Keeping this in mind and reading the above plan again now makes it clear that although it seems that the three wheneverdo statements are in parallel, it is really the case that each wheneverdo waits for the performance of the previous step in a top-to bottom reading of the plan. Now this is important to our current discussion of prioritized acting because, when the snif construct returns true that the agent is in a state of contingency at any one step in the plan, the next step of the plan cannot be executed because the princess does not say the speech string in the current step. This results in the wheneverdo not being triggered for that next step and the Princess effectively stops executing the entire performance plan.

In the interrupt handling routine (illustrated above), which does continue, the Princess performs the **duringleaving** performative to handle the interrupt of the user leaving thereby trying to get the user to return. If the user then returns, she recalls by self-perception her prior place in the original plan and disbelieves that she is in a state of contingency, since she has now finished handling the contingency. This allows her to continue executing the original plan where she left off since the snif condition for the agent being in a state of contingency will return false (see performance plan fragment in this section) and the wheneverdo for the next step in the plan will now be triggered.

Chapter 6

Agent Implementation

6.1 Role Performance

The primary motivation for the development of Princess Cassie has been the development of an embodied cognitive agent, designed to act the role of an actor in a virtual reality immersive drama. In chapter 4, I provided a description of the development of such an agent in terms of the specific parts that make up the whole of the agent. In chapter 5.1, I provided a discussion of the various issues that needed to be considered for an effective implementation of an embodied cognitive agent. A reading of those chapters would leave the reader with an understanding of all the different basic components and mechanisms that needed to be considered or developed for the agent to work. However, the mere development of these components and mechanisms does not result in an embodied cognitive agent. Consider the example of a child's set of building blocks. The set of building blocks, together with the mechanisms for putting the blocks together, carries no significance by itself, it only carries a certain potential. Only when the building blocks are put together into some meaningful whole, does any significant structure emerge. The various components described in chapter 4 are akin to the individual building blocks and the mechanisms of chapter 5 are akin to the mechanisms for putting the blocks together. The Princess actor will be the meaningful whole that will be built out of them.

Princess Cassie is designed to be a believable cognitive actor that is capable of drawing the user into psychological and social trials. She is an actor in every sense of the word that is applicable to human agents.

By this I mean that Princess Cassie needs to be able to react to cues in the scene, be able to interact with and respond to other actors, and be able to follow a script for performance. This ability would mean that she would have specific goals, intentions and desires - corresponding to the narrative arc of the scene - that she would try to achieve or bring about through the course of her performance in the scene. This section is concerned with providing an overview of her role as a cognitive actor in the factory/castle scene and her performance of that role.

The Princess' performance starts with the user entering the factory and ends with the user leaving the factory, before or after the Princess has been spirited away by the villains of the scene. To understand the performance of the Princess, consider a human actor in a similar situation in a real-life drama. Prior to the start of the scene, the human actor has been handed a script, which he has memorized. It is my belief that the human actor then develops a structured series of plans and intentions about how he will enact his performance. The Princess actor also follows a similar process with the difference being that in the current version of the implementation, the plans and intentions have been specified for the agent in pre-processing, based on the storyboard script. In future work, it is envisioned that mechanisms will be developed by which the Princess will be able to accept a script for the scene and build up her own plans and intentions on how to achieve the goal state from a particular start state.

The Princess actor therefore has holistic knowledge about the script, in the form of plans, intentions and goals that need to be achieved to bring about the plans or that use the intentions. In the current implementation, these plans and intentions have been structured into a complex framework of plans, intentions, goals and performatives, with the performatives being the smallest unit of performance. This structure is specified and described further below (using SNePSLOG [24, 29]).

At the highest level of planning, she is aware of a series of goals which, when achieved, will bring about the narrative arc specified by the script. Princess Cassie's highest level plan involves achieving the goals of greeting the user and getting him to a target area in front of the gate (**gettogate**), formalized as AgentLocation(b2, AtGate), explaining her predicament to the user and getting him to agree to help her (**explpredicament**), formalized as AgentAct(b2, agreestohelp) and finally explaining to the user how the

gate can be opened (**explhow2opendoor**), formalized as AgentAct(b2,searching). In this plan, as in all subsequent plans in this section, the atomic term (b1) refers to the Princess agent and the atomic term (b2) refers to the User.

```
snsequence_3(achieve(AgentLocation(b2,AtGate)),
             achieve(AgentAct(b2,agreestohelp)),
             achieve(AgentAct(b2,searching)))
```

Each goal can be achieved by executing certain plans, which are based on the script. Executing the plan **getstotarget** has the effect of achieving the goal of greeting the user and getting him to the gate, executing the plan **explainpred** has the effect of achieving the goal of explaining the Princess' predicament to the user and getting him to help her and executing the plan **explaindoor** has the effect of achieving the goal of explaining to the user how the gate can be opened. The Princess is therefore aware of a plan to achieve every goal that she needs to achieve to bring about the narrative arc of the scene.

```
all(p)(MemberClass(p,person)
=> GoalPlan(AgentLocation(b2,AtGate), getstotarget(p)))

all(p)(MemberClass(p,person)
=> GoalPlan(AgentAct(b2,agreestohelp), explainpred(p)))

all(p)(MemberClass(p,person)
=> GoalPlan(AgentAct(b2,searching),explaindoor(person)))
```

In the above structure, one might wonder what the advantage of using the twin levels of goals and plans is, when each goal is achieved by the execution of a single plan. At the highest level, one might reasonably argue for the performance of just a sequence of plans comprising of the **getstotarget**, **explainpred** and **explaindoor** plans, thus eliminating the need to achieve the higher level goals of greeting and getting the user to the gate, explaining her predicament to the user and explaining to the user how he can open the gate. In response to this argument, the reader is reminded of the discussion in section 5.3, in which I presented a mechanism that enabled Princess Cassie to entertain beliefs about high-level plans. Through the specification of goals for various parts of her performance, the agent has access to information that she can use to entertain beliefs regarding higher-level entities such as plans. For example, all the parts of the **getstotarget** plan, have the achievement of the goal of greeting and getting the user to the gate as their

motivation. As described in section 5.3, should this goal become asserted midway through the plan, the Princess is able to reason that there is no need for the further execution of the plan, allowing her to terminate execution of that plan.

Each plan above is in turn a chained cascade or sequence of performatives. Recall that the performative is the smallest unit of performance in the Princess Cassie implementation. The use of goals and plans can therefore be best understood as levels of abstraction. Any such level of abstraction would have to ultimately work down to the smallest unit, which is the performative. Each plan is specified and further described below.

1. The getstotarget plan: When executed this plan causes the agent to first greet the user using the **sayhello** and **callover** performatives and then to get the user to the target area in front of the gate using the **thankful**, **beckon**, **beckoncloser** and **canttalkloudly** performatives. When the user enters the factory, the Princess says "Hallo, Hallo" and looks at the user by performing the **sayhello** primitive. At this point, after she has transmitted intentions from her mind to her body to carry out the say and look primitives across the speech and animation modalities, as described in section 5.2.1, she waits for a perception that she has said the words "Hallo, Hallo". Only after she perceives that she has said "Hallo, Hallo" does she try to perform the next step of her plan. At this point, it is important to understand the precise structure of this plan so the reader can understand how Princess Cassie's plans are implemented. From a reading of the plan, it is apparently the case that after the Princess performs the **sayhello** primitive, she carries out five different believe statements in sequence which set up five different wheneverdo waits which could be executed in parallel. However, as previously described in section 5.4, just as the first wheneverdo statement waits for the Princess to have said "Hallo, Hallo" before executing the **callover** primitive, the next one waits for the Princess to have said the speech component of the **callover** primitive and the each next wheneverdo waits for a perception of the speech component of the performative controlled by the current wheneverdo. Thus, the duringgreet plan truly implements the notion of a cascade, discussed in section 5.2.1, by which each subsequent step in this plan is only executed after receiving a perception of the execution of the previous step. The steps of this plan also implement the mechanisms described in sections 5.3 and 5.4. After perceiving that she

has completed execution of the current act, Princess Cassie performs explicit checks to check if she is in a state of contingency or if the goal for which the current plan is being executed - getting the user to the gate - has been achieved. If either of the two conditions is satisfied, she stops execution of the plan by performing the performative **nothing**. Only if both the checks return false, does she perform the next step in the plan. This plan therefore provides a concrete example of the implementation of the various parts and mechanisms for agent development, described in chapters 4 and 5.

```
all(p)(MemberClass(p,person)
=> ActPlan(duringgreet(p),
    believe(
     wheneverdo(AgentAct(b2,approaching),
                snsequence_6(sayhello(b2),
                             believe(
                              wheneverdo(AgentAct(b1,say(Hallo,Hallo)),
                                         snif({if(AgentState(b1,contingency),
                                                  nothing),
                                               if(AgentLocation(b2,AtGate),
                                                  nothing),
                                               else(callover)}))))
                             believe(
                              wheneverdo(AgentAct(b1,
                                                  say(We're all over here)),
                                         snif({if(AgentState(b1,contingency),
                                                  nothing),
                                               if(AgentLocation(b2,AtGate),
                                                  nothing),
                                               else(thankful)}))),
                             believe(
                              wheneverdo(AgentAct(b1,
                                                  say(Thank goodness
                                                       you've come)),
                                         snif({if(AgentState(b1,contingency),
                                                  nothing),
                                               if(AgentLocation(b2,AtGate),
                                                  nothing),
                                               else(beckon)}))),
                             believe(
                              wheneverdo(AgentAct(b1,say(Please come
                                                         nearer)),
                                         snif({if(AgentState(b1,contingency),
                                                  nothing),
                                               if(AgentLocation(b2,AtGate),
                                                  nothing),
                                               else(beckoncloser)}))),
                             believe(
                              wheneverdo(AgentAct(b1,say(Even nearer)),
                                         snif({if(AgentState(b1,contingency),
                                                  nothing),
                                               if(AgentLocation(b2,AtGate),
                                                  nothing),
                                               else(canttalkloudly)}))))
)))))
```

2. The explainpred plan: When executed this plan causes the agent to perform the part of the script that is concerned with explaining to the user her predicament (of being caged by the villains) in the scene, and trying to convince him to help her. It waits for the user getting to the target area in front of the gate. Since the goal of greeting the user has been achieved at this point, the Princess asserts the belief that she has greeted the user, as the first step in this plan. The plan then involves the cascaded performance of the **introbabelamb**, **explthatcaged**, **exprdanger**, and **pleadfrhelp** performatives, using the mechanisms described in sections 5.2.1, 5.3 and 5.4, and previously implemented in the **getstotarget** plan. The description of the implementation of the **getstotarget** plan is also relevant here.

```
all(p)(MemberClass(p,person)
=>ActPlan(explainpred(p),
   believe(
    wheneverdo(AgentLocation(b2,AtGate),
               snsequence_5(believe(AgentAct(b1,greet(b2))),
                            introbabelamb,
                            believe(
                             wheneverdo(AgentAct(b1,
                                                 say(This is my babe,
                                                     and here is my
                                                     little lambkin)),
                                        snif({if(AgentState(b1,contingency),
                                                 nothing),
                                              if(AgentAct(b2,agreestohelp),
                                                 nothing),
                                              else(explthatcaged)}))),
                            believe(
                             wheneverdo(AgentAct(b1,
                                                 say(We're locked in here)),
                                        snif({if(AgentState(b1,contingency),
                                                 nothing),
                                              if(AgentAct(b2,agreestohelp),
                                                 nothing),
                                              else(exprdanger)}))),
                            believe(
                             wheneverdo(AgentAct(b1,
                                                 say(We're in terrible
                                                     danger)),
                                        snif({if(AgentState(b1,contingency),
                                                 nothing),
                                              if(AgentAct(b2,agreestohelp),
```

```
                                                    nothing),
                                              else(pleadfrhelp)}))))))))
```

3. The explaindoor plan: When executed this plan causes the agent to perform the part of the script that is concerned with explaining to the user, how the gate can be opened. This plan waits for the Princess' perception of the user agreeing to help the princess. The Princess asserts the belief that she has explained her predicament to the user, as the first step in this plan. The rest of the plan involves the cascaded execution of the **accepthelp** and **explhow2help** performatives, augmented by the mechanisms described in sections 5.2.1, 5.3 and 5.4.

```
all(p)(MemberClass(p,person)
=>ActPlan(explaindoor(p),
   believe(
    wheneverdo(AgentAct(b2,agreestohelp),
               snsequence_3(believe(AgentAct(b1,explpredicament(b2))),
                            accepthelp,
                            believe(
                            wheneverdo(AgentAct(b1,
                                                say(You can open the gate
                                                        from your side.)),
                                       snif({if(AgentState(b1,contingency),
                                                nothing),
                                             if(AgentAct(b2, searching),
                                                nothing),
                                             else(explhow2help)}))))))))
```

4. The coachabtdoor plan: When executed this plan causes the agent to perform the part of the script that is concerned with perceiving how the user is going about trying to open the gate, and periodically providing suggestions till he succeeds in opening the gate. This plan is a special plan in that it is not executed to achieve any goal in the highest-level meta plan, rather, it is meant to be a very reactive plan designed to just help the user. This plan is triggered by the Princess' perception of the user searching for a way to open the gate. The central element in this plan is the snif construct which enables the Princess to check exactly what the User is doing, before making an appropriate suggestion - performing the **notusinghand** performative if he is not using his hands to search, performing the **usingonehand** performative if he is using just one hand and performing the **duringsearch** performative if the user is using both hands and making a concerted effort to free her.

```
all(p)(MemberClass(p,person)
=>ActPlan(coachabtdoor(p),
   believe(
    wheneverdo(AgentAct(b2,searching),
               snsequence(believe(AgentAct(b1,explhow2opendoor(b2))),
                          snif({if(AgentAct(b2,usingbothhands),
                                   duringsearch),
                                if(AgentAct(b2,using1hand),
                                   usingonehand),
                                if(AgentAct(b2,notusinghands),
                                   notusinghand)})))))
```

Finally, each of the above plans is built as a chain of intentions which are executed contingent upon receiving appropriate perceptual cues from the world of the agent's embodiment. The ability to do so is extremely essential and the reason for this is obvious from an observation of human actors. Although the human actor has developed complete plans for performance, from the script, these plans are executed with regard to the state of the world. The actor waits for cues from the world (or from other actors who are a part of the world), and upon receiving one, he performs an appropriate action. In the plan specifications described above, this is implemented using the "wheneverdo" and "whendo" constructs. The reader is referred to [23, 24, 29] for a description of these constructs. In the current discussion it is sufficient to say that these constructs allow the agent to execute a particular action - or performative - only when a particular condition is satisfied in the world or a particular proposition, about the state of the world, is asserted.

6.2 Believability and Contingency Handling

Any agent that needs to interact with an unconstrained human user needs to have sophisticated contingency and interrupt handling capabilities. Princess Cassie's performance plans (described above in section 6.1) trace a fixed arc of scene progress, along the lines of the script. However, although Princess Cassie is constrained to follow this scene script, the user has no such constraints imposed upon him - he is unaware of the existence of a script in the first place. Therefore, should the user decide to follow an arc through the scene that does not correspond to what Princess Cassie expects, she should be able to dissemble effectively and handle any possible behavior of the agent. Of course, one could work the other way and design the

world in such a way that it does not allow the user any freedom to diverge from the script arc. However, this would not be in line with the driving motivations for this project as a whole, which are to allow the user sufficient autonomy and freedom, for him to be dramatically involved in and psychologically engaged by the story.

The effort in the current project to draw psychological responses from the user also imposes the requirement of believability upon the agent. Her effective portrayal of the archetypical "damsel in distress" is essential for the user to be drawn into the belief that she is a fellow agent in a real predicament. Further, the interactions with the user need to be flexible and dynamic enough to evoke natural responses from the user - as opposed to the kind of unnatural, disengaged responses usually elicited by computational programs or agents.

Princess Cassie entertains plans and intentions that allow her to handle possible contingencies and interrupting scenarios and which allow her to dissemble effectively and believably in the face of unpredictable behavior on the part of the user. Specifically, as described in sections 5.2.2 and 5.4, when contingencies occur, Princess Cassie has built in mechanisms which allow her to seamlessly handle contingent and interrupting scenarios without breaking her performance. The contingency plans are specified and described below (using SNePSLOG [24, 29]). In these plans, the atomic unit (b1) represents the Princess agent, (b2) represents the User, (b3) represents the "Bad Guys" or the villains of the scene, and (b4) represents the gate.

1. A coupled set of plans is used to handle the contingency in which the villains threaten Princess Cassie. These plans use the performatives **duringthreat** and **resptothreat** and the primitives memorize and recall.

```
wheneverdo(AgentAct(b3,start_threatening),
           snsequence_3(memorize,
                        believe(AgentState(b1,contingency)),
                        duringthreat))

wheneverdo(AgentAct(b3,stop_threatening),
           snsequence(resptothreat,
                      believe(wheneverdo(AgentAct(b1,
                                                  say(I'm so scared)),
                              recall)))
```

When the Princess perceives that the villains have started threatening her, she believes that she is in a state of contingency and performs the **duringthreat** primitive. As described in section 5.4, by entertaining the high-level belief that she is in a state of contingency, the agent is able to interrupt the original performance plan while handling the interrupt. When the princess perceives that the villains have stopped threatening her, she performs the memorize primitive so that the information pertaining to the point in the original plan where she was interrupted, can be saved for future use (see section 5.2.2 for a discussion of how this is done). She then responds to the bad guys threatening her by performing the **resptothreat** performative and after perceiving the completion of this performative, she recalls the point in the original plan where she was interrupted, by executing the recall primitive (see section 5.2.2 for a discussion of the mechanism involved).

One point that might seem ambiguous to the reader might be the reason for the agent responding to the bad guys only after they stop threatening her. This might seem counter-intuitive. However, the script for the agent's performance mandates that she maintain silence while the bad guys are threatening her, and respond to the threat only after they stop threatening her. This mandate is reflected in the plans above.

2. The plan which is used for handling the contingency in which the User decides to leave the scene and not help the Princess escape, or one in which the user just randomly leaves the scene is specified below. This plan uses the performative **duringleaving**.

```
wheneverdo(AgentAct(b2,leaving),
           snsequence_5(memorize,
                        believe(AgentState(b1,contingency)),
                        duringleaving,
                        believe(wheneverdo(AgentAct(b2,returns),
                                recall)),
                        believe(wheneverdo(AgentAct(b2,gone),
                                goOffStage)))))))
```

In this plan, when the Princess perceives that the User is leaving the target area in front of the gate, she asserts a belief that she is in a state of contingency, memorizes the point in the original plan

where she was interrupted, by executing the memorize primitive and then performs the **duringleaving** performative to handle the contingency. If the user returns to the target area, the Princess recalls the point in the original plan where she was interrupted by executing the recall primitive and continues her role performance. Otherwise, if the agent continues to leave and leaves the scene, then the agent terminates her performance by executing the GoOffStage primitive.

3. Although the gate opening is desirable in terms of the narrative arc, the Princess' perception of the gate opening is still handled as a contingent case. This is because the event of the gate opening can happen at any time during the Princess performance based on the actions of the user. Therefore, all the princess can do is react to the gate opening when it does. This plan uses the performatives **givethanks** and **exprjoy** in a cascaded sequence. Another special case in this plan is that after the contingency is handled, there is no need to return to the original plan because the villains of the scene carry away the Princess and the scene ends, as soon as the gate starts opening.

```
wheneverdo(ObjectEvent(b4,Opens),
           snsequence_3(believe(AgentState(b1,contingency)),
                        givethanks,
                        believe(whendo(AgentAct(b1,
                                        say(Thank the Lord!)),
                                exprjoy))))
```

Chapter 7

Conclusion

Princess Cassie is an embodied cognitive agent, based on the Grounded Layered Architecture with Integrated Reasoning (GLAIR), and built using Lisp and the Semantic Network Processing System (SNePS), a suite of languages for Knowledge Representation and Reasoning, intentional rational acting, inference and reasoning, belief revision, logical programming and natural language processing. The work on the development of Princess Cassie extends and builds upon previous work on embodied cognitive agent development carried out by Stuart Shapiro and his colleagues in the SNePS Research Group at the University at Buffalo. This work has been carried out as part of a larger collaborative effort aimed at the "enhancement of VR with intelligent agents" [26].

Princess Cassie, at her current state of development, allows for effective simulation and serves as an ideal test framework for the development of embodied cognitive agency. As I have described in this document, the development of the agent has included the implementation and addition of a suite of modalities to the existing GLAIR architecture, allowing for truly realistic behavior through asynchronous interactions across several modalities. The agent is capable of perceiving and identifying a wide range of perceptual cues from the world around her and is also able to perceive and process her own actions in the world. Princess Cassie also possesses a varied repertoire of actions that she is able to schedule and carry out intentionally, based on her beliefs, in order to accomplish her plans or achieve her goals. These include primitive actions - purely reactive actions that can be carried out without any conscious thought or reasoning about the details, internal

acts which are mental acts that are not externally manifested by the embodiment and finally complex acts or performatives that are multi-modal plans for action or behavior.

In this document, I have also shown an implementation of Princess Cassie as a cognitive actor in a Virtual reality drama. In the drama, *The Trial The Trail*, Princess Cassie plays the role of a trapped Princess that tries to engage a human user immersed in the drama. As part of this implementation, she serves as an actor in every sense of the word that is applicable to human agents. Princess Cassie is capable of entertaining a complex framework of specific goals, intentions and desires - corresponding to the the narrative arc of the scene - that she needs to achieve or bring about, through the course of her performance in the scene. She is able to wait for and react to cues in the scene, interact with other agents and is able to follow the script for performance. She is also capable of carrying out a wide range of sophisticated contingency and interrupt handling routines that allow her to dissemble effectively in the face of unpredictability in the scene, resulting from the user's actions or otherwise.

Several areas of interest in the development of embodied cognitive agency have been addressed and resolved in the current implementation. The issue of effective synchronization of Princess Cassie's actions in the world, has been resolved without resorting to the implementation of a personal sense of time. The development of Princess Cassie has also included the development of a notion of Short Term Memory and Working Memory in a cognitive sense. Using such a notion of memory, the Princess Agent is capable of effectively carrying out interrupt and contingency handling, continue execution of past plans and effectively carry out her performance in the face of unpredictability. Finally, the development of Princess Cassie has included the development of routines and techniques that allow her to reason, infer and process modular plans and intentions that are specified at a level higher than that specified by the architecture's details.

In its entirety, Princess Cassie provides an example of a cognitive agent that is capable of exhibiting a high degree of sophistication and autonomy. She is capable of true multi-modal behavior, adherence to a tight narrative arc, is believable, is capable of contingency handling, is able to achieve goals by formulating and executing plans, is able to reason and infer about knowledge and she is able to dissemble in the face of unpredictability.

Bibliography

[1] J. Anstey. Writing a story in virtual reality. In M. Hocks and M. Kendrick, editors, *Eloquent Images*, pages 283–304. MIT Press, Cambridge, Massachusetts, 2003.

[2] J. Anstey. Agent development home page [online], Cited March 31 2004. Available from World Wide Web: http://www.ccr.buffalo.edu/anstey/VR/TRAIL/agent/.

[3] J. Anstey, D. Pape, and D. Sandin. The thing growing: Autonomous characters in virtual reality interactive fiction. *IEEE Virtual Reality 2000*, 2000.

[4] J. Anstey, D. Pape, S. C. Shapiro, and V. Rao. Virtual drama with intelligent agents. In H. Thwaites, editor, *Hybrid Reality: Art, Technology and the Human Factor, Proceedings of the Ninth International Conference on Virtual Systems and Multimedia (VSMM)*, pages 521–528. International Society on Virtual Systems and Multimedia, 2003.

[5] A. D. Baddeley. Working memory: The interface between memory and cognition. *Journal of Cognitive Neuroscience*, 4(3):281–288, Summer 1992.

[6] A. D. Baddeley. Short-term and working memory. In E. Tulving and F. I. Craik, editors, *The Oxford handbook of memory*, pages 77–92. Oxford University Press, London, 2000.

[7] S. Coradeshi and A. Saffiotti. Forward. In S. Coradeshi and A. Saffiotti, editors, *Anchoring Symbols to Sensor Data in Single and Multiple Robot Systems: Papers from the 2001 AAAI Fall Symposium, Technical Report FS-01-01*, page vii. AAAI Press, Menlo Park, CA, 2001.

[8] S. Coradeshi and A. Saffiotti. Perceptual anchoring of symbols for action. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 407–412, San Francisco, CA, 2001. Morgan Kaufman.

[9] S. Coradeshi and A. Saffiotti. An introduction to the anchoring problem. *Robotics and Autonomous Systems*, 43(2-3):85–96, May 2003.

[10] R. W. Hill, J. Gratch, S. Marsella, J. Rickel, W. Swartout, and D. Traum. Virtual humans in the mission rehearsal exercise system. *KI special issue on Embodied Conversational Agents*, 2003.

[11] H. O. Ismail. *Reasoning and Acting in Time*. PhD thesis, The State University of New York at Buffalo, August 2001.

[12] H. O. Ismail and S. C. Shapiro. Cascaded acts: Conscious sequential acting for embodied agents. Technical Report 99-10, Department of Computer Science and Engineering, The State University of New York at Buffalo, Buffalo, NY, November 1999.

[13] J. Lehman, J. Laird, and P. Rosenbloom. A gentle introduction to soar, an architecture for human cognition. In D. Scarborough and S. Sternberg, editors, *Methods, Models and Conceptual Issues*, volume 4 of *An Invitation to Cognitive Science*. MIT Press, Cambridge, MA, 1998.

[14] H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. B. Scherl. GOLOG: A logic programming language for dynbamic domains. *Journal of Logic Programming*, 19(20):1–25, 1994.

[15] M. Mateas and A. Stern. A behavior language for story-based believable agents. In K. Forbes and M. E.-N. Seif, editors, *Working Notes of Artificial Intelligence and Interactive Entertainment*, AAAI Spring Symposium Series. AAAI Press, Menlo Park, CA, 2002.

[16] D. Pape. *Future Generation computers*, chapter Ygdrasil - a framework for composing shared Virtual Worlds. Elsevier Press, 2003.

[17] D. Pape, J. Anstey, B. Carter, M. Roussou, and T. Portlock. Virtual heritage at igrid 2000. In *The Internet Global Summit(INET01)*. 2001.

[18] P. S. Rosenbloom, J. E. Laird, and A. Newell. *Introduction to The Soar Papers: Readings on Integrated Intelligence*. MIT Press, Cambridge, MA, 1993.

[19] J. F. Santore. Multiprocessing, semaphores and networking with acl. SNeRG Technical Note 33, Department of Computer Science and Engineering and SNePS Research Group, The State University of New York at Buffalo, Buffalo, NY, 2002.

[20] J. S. Santore and S. C. Shapiro. Crystal Cassie: Use of a 3-d gaming environment for a cognitive agent. In R. Sun, editor, *Papers of the IJCAI 2003 Workshop on Cognitive Modeling of Agents and Multi-Agent Interactions*, pages 84–91, Acapulco, Mexico, August, 9 2003. IJCAI.

[21] M. P. Shanahan. The event calculus explained. In M. J. Wooldridge and M. Veloso, editors, *Artificial Intelligence Today: Recent Trends and Developments*, number 1600 in Lecture Notes in Artificial Intelligence, pages 409–430. Springer-Verlag, Berlin, 1999.

[22] S. C. Shapiro. Embodied cassie. In *Cognitive Robotics: papers from the 1998 AAAI Fall Symposium*, pages 136–143. AAAI Press, Menlo Park, California, October 1998. Technical Report FS-98-02.

[23] S. C. Shapiro. Sneps: A logic for natural language understanding and commonsense reasoning. In Iwánska and S. C. Shapiro, editors, *Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language*, pages 175–195. AAAI Press/ MIT Press, Menlo Park, CA, 2000.

[24] S. C. Shapiro. FevahrCassie: A description and notes for building FevahrCassie-like agents. SNeRG Technical Note 35, Department of Computer Science and Engineering, The State University of New York at Buffalo, Buffalo, NY, September 2003.

[25] S. C. Shapiro. Cassie as a self-aware SNePS/GLAIR agent. Panel Presentation at the DARPA Workshop on Self-Aware Computer Systems, April 27-28 2004. Available from World Wide Web: http://www/cse.buffalo.edu/ shapiro/Talks.

[26] S. C. Shapiro and J. Anstey. Virtual reality and intelligent agents for social simulations and interactive storytelling. Proposal for NSF Funding, 2004.

[27] S. C. Shapiro and H. O. Ismail. Symbol anchoring in a grounded layered architecture with integrated reasoning. *Robotics and Autonomous Society*, 43:97–108, 2003.

[28] S. C. Shapiro, H. O. Ismail, and J. F. Santore. Our dinner with cassie. In *Working Notes for the AAAI 2000 Spring Symposium on Natural Dialogues with Practical Robotic Devices*, pages 57–61, Menlo Park, CA, 2000. AAAI.

[29] S. C. Shapiro and The SNePS Implementation Group. *SNePS 2.6 User's Manual*. Department of Computer Science and Engineering, University at Buffalo, The State University of New York at Buffalo, Buffalo, NY, 2002. Available from World Wide Web as: http://www.cse.buffalo.edu/sneps/Manuals/manual26.ps.