Collaborative Narrative Generation in Persistent Virtual Environments

by Neil Madden, BSc

Thesis submitted to The University of Nottingham
for the degree of Doctor of Philosophy, January 2009

Abstract

This thesis describes a multi-agent approach to generating narrative based on the activities of participants in large-scale persistent virtual environments, such as massively-multiplayer online role-playing games (MMORPGs). These environments provide diverse interactive experiences for large numbers of simultaneous participants. Involving such participants in an overarching narrative experience has presented challenges due to the difficulty of incorporating the individual actions of so many participants into a single coherent storyline. Various approaches have been adopted in an attempt to solve this problem, such as guiding players to follow pre-designed storylines, or giving them goals to achieve that advance the storyline, or by having developers (*'dungeon masters'*) adapt the narrative to the real-time actions of players. However these solutions can be inflexible, and/or fail to take player interaction into account, or do so only at the collective level, for groups of players.

This thesis describes a different approach, in which embodied *witness-narrator agents* observe participants' actions in a persistent virtual environment and generate narrative from reports of those actions. The generated narrative may be published to external audiences, e.g., via community websites, Internet chatrooms, or SMS text messages, or fed back into the environment in real-time to embellish and enhance the ongoing experience with new narrative elements derived from participants' own achievements.

The design and implementation of this framework is described in detail, and compared to related work. Results of evaluating the framework, both technically, and through a live study, are presented and discussed.

Acknowledgments

I'd like to thank, first and foremost, my supervisor, Brian Logan, who has provided much useful support and guidance during my PhD research. It is fair to say that this thesis would not exist at all if not for his tireless encouragement. Thanks also to Steve Benford for much useful feedback and advice.

I would also like to thank the School of Computer Science for funding my research and providing office space and equipment for my work. I am grateful to the Mixed Reality Lab for providing employment and interesting projects when my funding ran out.

Of course, I could not have completed this work without the help and support of my friends and family. I'd particularly like to thank Catherine Preston, Johanna Madden, and Katy Brown, and my other friends and house-mates over the years. Finally, I'd like to thank my parents and the rest of my family for their support and encouragement.

Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

1.1 Motivation

The last few years have seen the creation of a large number of persistent virtual environments for entertainment, e.g., massively-multiplayer online role-playing games (MMORPGs). These environments provide diverse interactive experiences for very large numbers of simultaneous participants. However, their sheer scale and the activities of other participants makes it difficult to involve players in an overarching narrative experience. One of the main attractions of such environments is the ability to interact with other human players. Such interaction precludes the possibility of an omniscient narrator who 'tells a story' which structures the user's experience, as much of this experience is driven by the (unknowable) thoughts and feelings of other players. A desire to integrate more user-driven storytelling elements into games in general can be found expressed in several recent articles by game developers. For instance, in [110], Neil Sorens argues that "*...designers can and should do more to exploit these player-generated stories.*" The article discusses in particular so-called sandbox games, in which the player has a largely free hand in how the game unfolds. Examples include games such as *The Sims* or *Civilization*. The general argument, however, applies to other game genres such as MMORPGs: games should do more to emphasise the narrative elements of gameplay and highlight the achievements of players in story form. Various approaches have been adopted in an attempt to solve these problems, such as guiding players to follow pre-designed storylines [123], or giving them goals to achieve that advance the storyline, or by having developers (*'dungeon masters'*) adapt the narrative to the real-time actions of players. However these solutions can be inflexible, work for only single players, and/or fail to take player interaction into

account or do so only at the collective level, for groups of players.

This thesis describes a different approach, in which embodied *witness-narrator agents* observe participants' actions in a persistent virtual environment and generate narrative from reports of those actions [111]. The generated narrative may be published to external audiences, e.g., via community websites, Internet chatrooms, or mobile telephone text messages, or fed back into the environment in real-time to embellish and enhance the ongoing experience with new narrative elements derived from participants' own achievements. Such in-world narration may enhance the enjoyment of participants, and being talked about is a way of building a reputation and progressing in the community of players. The possibility of appearing in a report (e.g., when doing well in a game) can help to motivate players, and the narrated events can, in turn, influence the participants' future activities thus helping to drive events in the environment. The latter may be particularly important in, e.g., MMORPGs, where the quests and challenges are periodically reset.

The approach we have adopted is based on the use of a multi-agent team, including both embodied and easily recognisable in-game agents that observe and report on the activities of participants, and external commentator agents that take the narrative generated by in-game agents and translate it into a format suitable for publication to an external audience (which may or may not include the actual participants). Such publication channels could include automatically generated web pages, Facebook-style social networks, RSS or Atom [1] newsfeeds, instant messaging services, or mobile telephone short-messaging services (SMS). In this thesis we describe in depth only a simple web-based interface, using existing web publishing software, but the design of the system can be adapted to other output media with minimal changes.

The back-bone of the framework is a large formal ontology of events and activities that are typical of persistent role-playing games. We motivate the development of this ontology, and show how it has been implemented, while also describing how it can be adapted and extended for use in new game environments. The ontology draws on elements from narrative theory and formal theories of intentional action to provide a flexible basis for representing a wide range of different activities and translating these into simple narratives that incorporate the activities of individual participants. In this way the thesis directly addresses the motivating question of how to acknowledge the accomplishments of individuals in a way that allows them to feel part of an ongoing narrative experience. We do not however address the larger question of how players' actions can be incorporated

into a single coherent narrative, but rather show how individual actions can be used to generate overlapping narrative strands. These strands could then be used as raw material in any attempt to overlay a more comprehensive approach to narrative within a MMORPG.

A key design decision in the approach described is the focus on player participation in the narrative generation process. This is the primary reason for having agents embodied in the game world, rather than using a hidden and omniscient process. This allows players to know when they are being observed and to adjust their behaviour accordingly. Beyond this, players are also able to interact directly with the agents, to tell them to stop observing them, or to focus the attentions of the agents on particular activities that they would like to see reported in the generated narratives. These aspects give players a degree of control over the story generation process, which we term *collaborative narrative generation* for this reason. This collaboration not only increases the likelihood that the narratives will be more interesting to the intended target audience, but also provide important controls over what could otherwise be a potentially disruptive activity. We feel that such control is essential for responsible reporting in what may be a public and long-running game experience.

1.2 Research Objectives and Contributions

The research objectives that are addressed in this thesis are as follows:

1. To develop a comprehensive framework for recognising, representing and reasoning about events and activities in diverse persistent virtual environments;
2. To show how existing theories of knowledge representation, story generation, and multi-agent coordination can be adapted to coverage of such online environments;
3. To demonstrate that this framework can be applied to a typical commercial role-playing game, supporting a reasonable number of simultaneous participants;
4. To evaluate the performance characteristics of the approach and whether it is feasible to apply to large-scale game environments;
5. To evaluate whether the use of narrative increases participation in the game;
6. To evaluate how the presence of narrative agents affects the gameplay, either positively or negatively, and to explore the nature of this interaction.

These research objectives are addressed in detail in the thesis text, and in the main evaluation chapter. The research approach builds in part upon earlier work by Daniel Fielding investigating reporting in the *Unreal Tournament* game [41]. The current work expands upon that earlier work in a number of different ways: firstly, the scope of the current work is much larger, encompassing a wide variety of different potential game environments. Secondly, the underlying theory and representational power of the system has been greatly enhanced, and given a more formal footing, building on theories of knowledge representation and reasoning. Finally, the approach to the problem has been expanded to consider narratives, rather than just reported actions.

1.3 Overview of Thesis

The task of generating narrative from events occurring in a persistent virtual environment involves a number of different areas of research. The relevant background literature is reviewed in Chapter 2 and Chapter 3, before describing in depth the design and implementation of the framework that constitutes the original research of the thesis in chapters 4 to 6. Finally, we evaluate the research, with respect to the research objectives, in Chapter 7, and finish with some general conclusions and pointers to future research directions in Chapter 8. The outline of the thesis is as follows:

Chapter 2: Background: Generating Narrative reviews the literature related to the core task of recognising activities and generating narrative. Firstly, we review a number of theories of narrative to see what elements a framework for narrative generation must include. Then we look in depth at a number of approaches to knowledge representation and reasoning, and how these can be applied to representing the objects and characters in a game world, and also the dynamic and temporal aspects of the events that occur. Finally, we conclude with a look at existing approaches to narrative prose generation.

Chapter 3: Background: Multi-Agent Systems then reviews the literature related to multi-agent systems, which forms the basis for the design and implementation of the framework. This chapter briefly describes the multi-agent approach to software engineering before describing the key theories that underly the approach, and the various concrete agent architectures that have been proposed. We conclude the chapter with

a look at multi-agent system coordination strategies and approaches to teamwork.

Chapter 4: Collaborative Narrative Generation provides the high-level design and specification of the system, and describes in detail the problems it is intended to solve;

Chapter 5: Ontology of Role-Playing Games describes in depth the formal ontology of role-playing games that forms the core of the system, and shows how it relates to the background research in chapters 2 and 3;

Chapter 6: Multi-Agent Implementation then details how the reference design was implemented and integrated with the test environment.

Chapter 7: Evaluation describes the evaluation of the software; and,

Chapter 8: Conclusions finishes with some general conclusions.

1.3.1 *Neverwinter Nights*

To evaluate the results of the research described in this thesis, a suitable test environment was chosen that represents a realistic application of the technology, while being feasible to integrate in the time available. The computer role-playing game *Neverwinter Nights* developed by Bioware Corporation[1] was chosen for a number of reasons. Firstly, the game provides for medium-scale environments of reasonable complexity, supporting a variety of different interactions between players and non-player characters. The game has also been developed to support multi-player, persistent game worlds, where up to around 64 simultaneous players can be supported in worlds of several hundred distinct regions. Furthermore, such persistent world severs can be joined by means of portals that can teleport a player to a region on a different server, effectively extending the scope of the environment. While these worlds are still considerably smaller than current commercial massively-multiplayer games such as *World of Warcraft*, the size of the environment is significantly larger than many other multi-player games and is sufficiently large to provide a good test of the technologies that have been developed. Importantly, *Neverwinter Nights* (NWN) also provides tools for users to create their own game worlds ('modules') and has a simple integrated scripting language that allows a certain amount of third-party observa-

[1] `http://nwn.bioware.com`

tion of creatures and objects in the environment, and the events they are involved in, which is necessary for the task of generating narrative from the environment to be possible at all.

A number of other possibilities were considered before settling on *Neverwinter Nights*. Firstly, we considered applying the technology to an existing commercial MMORPG. However, no existing game at the time of writing provided a public interface for such research to be feasible. While we did contact a number of publishers of such games to discuss the possibility, we received no response. One large-scale persistent virtual environment that does provide some level of user-created content and could be a potential test-bed for this research is the popular *Second Life* collaborative virtual environment[2]. However, after evaluating this environment it was discovered that there was a lack of necessary structure to the interactions in the environment, and a lack of facilities for detecting and recognising third-party events and activities at even quite a basic level. It is difficult, for instance, on perceiving an object in the environment to determine what that object is supposed to represent, and there is almost no support for determining what actions participants are performing at a level that would be needed for convincing narrative generation. *Neverwinter Nights*, on the other hand, provides a much more structured experience for players and provides a reasonable level of detail for agents perceiving the environment: objects can be identified, and meaningful actions can be perceived. It was also felt that NWN, as a successful commercial role-playing game, also provides an environment that is more typical of current technology.

[2] `http://secondlife.com`

CHAPTER 2

BACKGROUND: GENERATING NARRATIVE

This chapter presents a review of the literature related to the various areas involved in generating narrative from events in persistent virtual environments. The first question we must answer in such an endeavour is what constitutes a 'narrative'? How can a narrative be constructed? What are the key elements of such a narrative? In Section 2.1 (page 8) we look at various theories of narrative that have been developed in an attempt to answer these questions. Regardless of the particular theory that is adopted, it is clear that some notion of 'event' will be necessary. After all, it is the events that occur in a virtual world that will form the source material from which we will attempt to construct narratives. In Section 2.2 (page 15) and Section 2.3 (page 29) we will look at how to represent events formally, and how to reason about them. This involves two aspects: in the first section, we will consider the *content* of events (what happened, to whom, where, how, etc.); and, in the second section, we will consider the crucial *temporal* aspects of events (when they happen, and the temporal relations between events).

The question of how to *recognise* events as they occur, and how to *comprehend* how those events fit into a larger context is the focus of Section 2.4 (page 36), which reviews the literature from the areas of plan and activity recognition. This work is closely related to our task, as it is concerned with making sense of sequences of observed events (at a low level) in terms of higher-level concepts, such as plans, intentions, and activities. While this is not an identical task to forming a narrative, the process of understanding the significance of events at a higher level of abstraction is a useful first step. Finally, there is the task of actually narrating the story that has been constructed. In Section 2.5 (page 42) we address some approaches to this final task, concentrating on generating natural language text as a first target.

2.1 Theories of Narrative

The question of what constitutes a narrative, and how to understand narrative as a form of discourse, has been studied at least since the time of ancient Greece. In the *Republic*, Plato distinguishes between two ways of presenting a story: *diegesis*, in which the story is told by the narrator; and *mimesis* in which the events are 'imitated' or directly acted out (shown rather than told). Aristotle's *Poetics* [5] describes how forms of 'poetry' differ in three ways: the medium of imitation (e.g., rhythm and harmony in music, language in poetry); the objects of the imitation (the men whose actions are being described, and whether they are conceived as of high or low moral character); and the mode or manner in which the story is related (whether the story is narrated or directly presented). In this section we will look at theories of narrative, and in particular *structuralist* theories of narrative, that aim to analyse the structure of a story, and how such stories are distinguished from mere reports.

The review provided here concentrates on those elements of formalist and structuralist theories of narrative that seem most relevant to this thesis. We do not cover areas such as 'deep narrative structure' associated with theorists such as Lévi-Strauss and Greimas, or the 'post-structuralist' theories associated with Barthes and others. The focus on structuralist theories is justified as these theories provide the most promising approach to constructing narratives automatically. The persistent virtual worlds from which we will extract our source material are inevitably implemented in terms of some formal structure of events and objects existing in the (virtual) world. We can therefore build on this structure and apply techniques from structuralist narrative theories without the difficulty of extracting such a structure from a text.

2.1.1 Propp and the Russian Formalists

Modern narrative theory (or narratology) is usually seen to begin with the Russian Formalists, such as Vladimir Propp. Propp's *Morphology of the Folktale* [93] analysed a large number of Russian fairy tales ('wonder tales') and describes a structure common to each of them, in terms of 31 basic 'functions' (episodes of action with a particular purpose in the story) that occur in the same linear sequence in all such tales (although some functions may be omitted). These functions move the story from an initial situation in which there is a misfortune or lack of something, to the hero being despatched on a quest, encountering a magical helper of some kind, being subjected to various tests, and finally succeeding on

the quest and being rewarded (by marriage). Propp also describes how these functions are joined together into *moves* which make up the tale as a whole. Moves can be combined in several ways, such as sequentially (one move terminates and then another begins) or interleaved (one move begins and terminates within another move). Propp identifies seven main character roles ('spheres of action') within these stories:

1. the *villain*;
2. the *donor* (who provides the hero with a magical artefact);
3. the *helper*;
4. a *princess* and her *father* (taken together as a single role);
5. the *dispatcher*, who sends the hero on the quest;
6. the *hero* themselves;
7. and, a *false hero*.

The primary focus of the analysis is on the functions and the sequence in which they occur. However, Propp also describes other important aspects such as the way in which characters are introduced into the story, and how these characters are described. Aspects of character development are only briefly touched upon, however, presumably as this is not a major part of fairy-tale stories. Character development is seen as more central in modern novels however, so it is not clear how well a Propp analysis would apply to such works.

Perhaps the key idea from the Russian formalists is that a narrative can be split into two parts: the chronological sequence of events that are being narrated (*fabula*) and the way in which those events are represented/narrated (*sjužet*). This important distinction between the story or plot and the narrative or discourse that relates those plot events is present in most of the narrative theories that have subsequently been developed.

2.1.2 Structuralist Theories of Narrative

The distinction between story and narrative is further elaborated in various *structuralist* theories of narrative that were developed during the twentieth century. Such works attempted to analyse a narrative by discovering the underlying structure of the story being

narrated in terms of the events and characters being portrayed. Gérard Genette [48] analyses the relationship between these two layers of a narrative (which he terms *histoire* and *récit*) in terms of temporal relationships (order, frequency and duration) and aspects of *mood* and *voice*.

Order refers to the relationship between the temporal order of events in the story (i.e., the order in which events actually happened) and the order in which those events are narrated. Differences between these two orderings are termed *anachronies*, of which there are two major types: *prolepses* anticipate events that are yet to occur in the story, and *analepses* describe events that occured earlier in the story (e.g., flashbacks). Each of these anachronies has a *reach* (how far in time it moves from the current narration point) and an *extent* (the length of time that it covers). The types of anachrony can then be further characterised by these considerations: an external analepsis, for instance, has a reach that lies outside of the timeline covered by the original narrative, whereas an internal analepsis covers a time that is within the narrative timeline. Such internal analepses can be further divided into *heterodiegetic* analepses that deal with a different storyline to the main one (e.g., filling in background on a character), and *homodiegetic* analepses that deal with the same storyline (e.g., filling in gaps in earlier narrative, or simply repeating earlier events). Such anachronies are always relative to some original base-line narrative. It is possible that anachronies can be nested, so that for instance there is a flash-back in a flash-back, or perhaps an anticipation within a flash-back (perhaps anticipating the events in the original narrative timeline). This allows for a complex relationship between time on various levels of narrative.

Duration refers to the relationship between the length of time an event took to occur versus the amount of time dedicated to it in the narrative. A rough measure of the 'speed' of a narrative can be described by the amount of text (lines, words) devoted to each time unit (days, minutes) of story. Genette identifies four basic narrative movements:

- *ellipsis*, where whole sections of time are skipped in the narrative;
- *pause*, where some text is taken to describe something that takes up no time at all in the story (e.g., describing an object or scene);
- *scene*, where there is a rough equality between story time and narrative time (this is most often in dialogue);

- and, *summary*, where there is a variable pace of narrative, which can vary from scene to ellipsis.

Frequency refers to how often an event occurs related to how often it is narrated. An event in the story can occur once or it can occur many times (such as a regular train journey), and either sort can be narrated either once or many times. For instance, a single episode can be retold several times from different points of view, or several events can be combined with phrases such as 'every day' or 'for the whole week'. This combining of several regular events into one piece of narrative is termed an *iterative*. Genette again breaks this down into the *determination* (limits on the overal temporal interval being described, e.g., 'between 1939 and 1945'), the *specification* (rhythm of recurrence, e.g., 'every Sunday'), and the *extension* (how long each episode lasts, e.g., one day). The iterative can be seen as an alternative to summary: instead of accelerating the pace with which events are told, it instead abstracts over the common details of a sequence of events to describe them as a whole.

Genette also discusses difference between the identity of the narrator and the character through whose point of view the story is focused (the *focalization* of the narrative). Several aspects of the relationship between these two concepts are discussed, such as the relative knowledge of each, and how the narrator can shift focalization in a text. There are also various relationships between the narrator and the story being narrated, such as the temporal relationship (is the narration supposed to be subsequent to the story, simultaneous with it, or perhaps even anticipating, as in a prophesy), and whether the narrator is present in the story (homodiegetic) or absent from it (heterodiegetic).

Chatman

Seymour Chatman [24, 25] builds on the formalist/structuralist view of narrative and attempts to generalise it to cover diverse media, such as film and dance, as well as verbal discourse. Chatman looks at narrative using ideas from semiotics and linguistics. Thus, narrative is analysed along two axes: on the one hand it is split into *content* and the *expression* of that content (the familiar distinction between story/plot and discourse/narrative), and on the other hand it is further broken down into *form* and *substance*. The form of content is the structure of the events that occur and the characters and settings involved in these events. The substance of this content is the 'real' world people and things that are

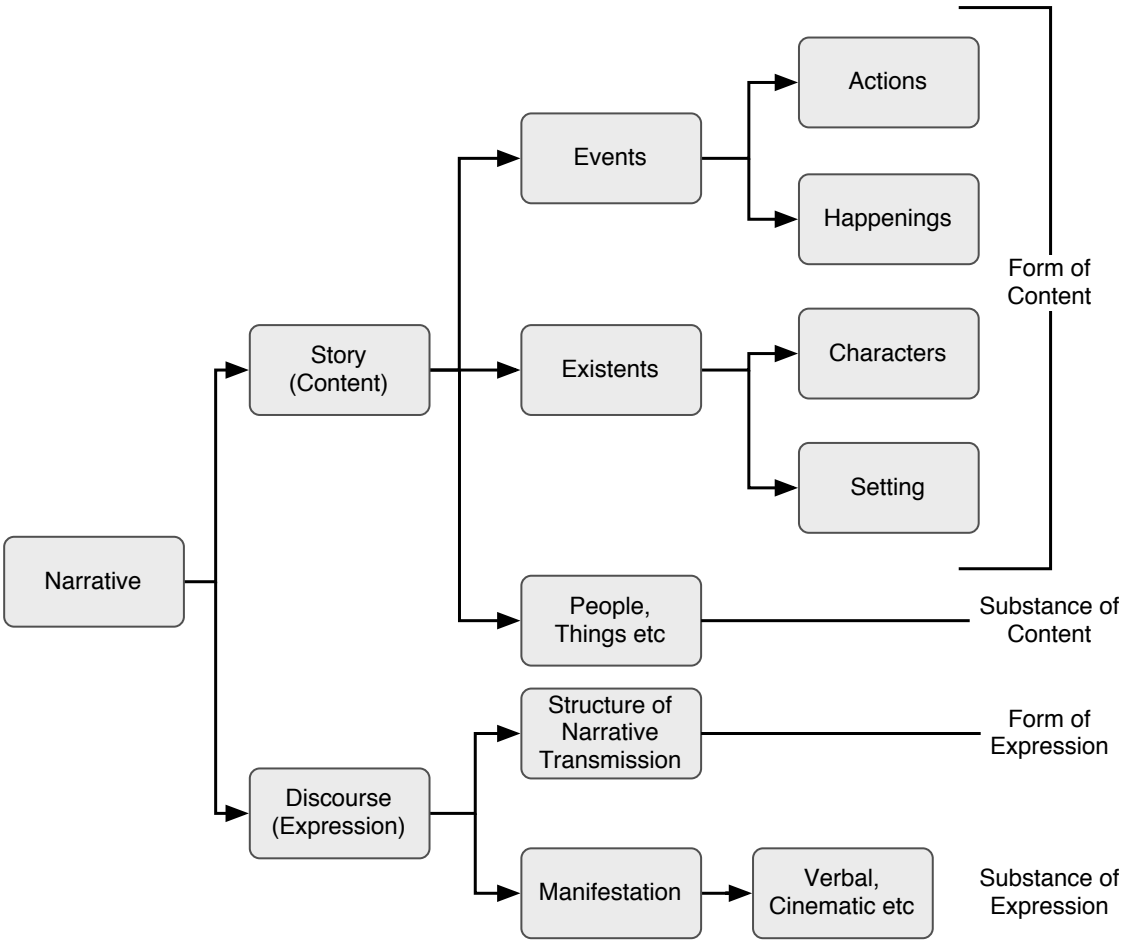

FIGURE 2.1: Chatman's structure of narrative.

referred to. The discourse (expression) is also broken down into form and substance. The form of discourse is those elements of narrative that are common to any medium, whereas the substance is a particular medium used for storytelling (e.g., cinema, books etc.). These ideas are shown in Figure 2.1, adapted from [25] (p. 26).

In Chatman's theory, the process of creating a narrative discourse consists of first *selecting* and *ordering* events from an underlying plot structure, together with the *existents* (things that exist in the world) implied by those events, into a sequence of narrative statements. These statements can then be translated into one or more media of presentation. Narrative statements can themselves be broken into *process* statements that describe actions and happenings, and *stasis* statements that describe characters and settings. Process statements can either be *enacted* or *recounted*, corresponding to the distinction between mimesis and diegesis discussed earlier. Process statements can also imply (or index) existents, and stasis statements can project events (e.g., the statement 'John is a loser' projects a history of events in which John has repeatedly lost). Events themselves can either be logically essential to the plot (*kernels*) or not (*satellites*). Events in a plot are typically not just temporally sequenced, but there is a logical relationship between them; if not causation, then at least some form of 'contingency'. Kernel events are significant in that they advance the plot and take part in this causation or contingency. Kernel events are pivotal moments, and cannot be deleted without changing the logical structure of the story. Satellite events however, can be deleted without changing the structure of the story. Their purpose is to fill out and expand upon the kernel events, resulting in a richer narrative discourse. Chatman employs Genette's descriptions of the temporal dimensions of events and their relationship to narrative (order, frequency, duration) to describe this 'microstructure' of events. However, he avoids describing a 'macrostructure' of plot types, such as that described by Propp for Russian fairy tales. For Chatman, while Propp's work is clearly valuable, the task of describing a typology of plot types in general would require a much better understanding of the cultural assumptions underlying the conventions used in particular story types. Developing such a typology is therefore left as an area of future research.

Chatman also devotes an equal amount of time to discussing character as well as events. Most previous theories have made character subservient to plot. Characters are described (e.g., in Propp) by the functions they play in the plot; it is the actions performed that indicate character. However, as Chatman argues, it is the character of agents in a story that causes them to perform certain actions. Therefore both character and event must be

treated on an equal basis. Chatman bases his theory of character on a notion of *traits* (i.e., personality or psychological traits), such as 'jovial', 'good humoured', 'murderous' etc.[1] Traits are indicated by habits (repeated behaviour); for instance, a habit of repeatedly washing ones hands might indicate a 'compulsive' trait. A character is then a 'paradigm' (here seeming to roughly mean set or collection) of traits. Chatman describes how a reader may come to understand the character of an individual while reading a story ([25] pp. 127):

> *"We sort through the paradigm to find out which trait would account for a certain action, and, if we cannot find it, we add another trait to the list..."*

Individuals may not actually be characters in a story. For instance, there may be general descriptions of groups of individuals in the background of the story (e.g., crowds at a football match). Clearly, each of these individuals is not a significant character in the story, but instead form part of the *setting*. Chatman identifies 3 possible means for distinguishing characters from setting: biology (e.g., whether the individual is human), identity (whether the individual has a name in the narrative), and importance to the plot. None of these options seems entirely satisfactory, and counter-examples can be found for all three. The most promising seems to the last, and so character is based primarily on the importance of an individual to the plot: as with kernel events, it is not possible to delete a character without changing the story.

In addition to describing the story/plot-level elements of events and existents, Chatman also describes in detail the various aspects of the discourse level, such as the narrator and the narratee. In fact, he describes a complicated situation in which there is not simply an author, a narrator and a reader, but instead a sequence of entities involved in the communication of a narrative: the real author, an implied author, the narrator, a narratee (who the narrator is narrating to), an implied reader, and the real reader. He also describes such issues as point of view of the narrator and characters, the voice of the narrator, and the meaning of speech and thoughts, both narrated and attributed to characters, in terms of speech act theory.

Other theorists, such as Shlomith Rimmon-Kenan [100], present refinements to the basic structuralist theory of narrative outlined above. However, the core of the theory remains similar to those of Chatman and Genette. In particular, the consideration of both events and characters as in Chatman, and the temporal relationships between story

[1] Chatman notes a study that identified some 17,953 trait names in Webster's Unabridged Dictionary.

and discourse, along with the idea of focalization, from Genette, feature prominently in Rimmon-Kenan's work.

2.2 Knowledge Representation

The structuralist theories of narrative we have reviewed in the previous section describe the structure of narrative as consisting of events and existents (characters, settings, etc). These basic elements can themselves be complex constructions (e.g., a character can have various traits, an event can have causes and effects). In this section we will review various approaches to knowledge representation and reasoning that provide powerful methods for representing these complex entities and the relationships between them, and for reasoning about this knowledge to infer relevant information (e.g., possible causes for events). In this section we will concentrate on basic knowledge representation formalisms that allow us to describe the basic properties of such entities, such as physical characteristics, location, etc. We will postpone discussing how to reason about the dynamic properties of objects (i.e., aspects that change over time) until Section 2.3 (page 29) when we discuss how to represent the temporal aspects of events.

The content of events may be quite complex, involving locations, buildings, objects of various types, and of course participants. The participants themselves may be involved in complex relationships that might be of interest to the audiences for which we are generating reports. In general then, it is useful to develop a formal model, or ontology, of the kinds of objects and actors that exist within the environments we are narrating, the properties they have, and the types of events that they may participate in. Given that one of the stated goals of this research is to develop a *general* framework for collaborative narrative generation, it would be useful to employ a general knowledge representation mechanism that can capture the character of a diverse range of environments in general, and that can be appropriately tailored to a particular environment if need be. A number of popular knowledge representation technologies have been described in the literature, and here we review a selection of the most useful approaches. Each technology represents a particular trade-off between representational expressiveness and the computational complexity of reasoning over those representations, and we will discuss these aspects for each technology.

2.2.1 Propositional and Predicate Logic

One of the most fundamental knowledge representation formalisms is that of propositional logic, which allows for representing and reasoning about propositions (atomic statements that may be either true or false). First-order logic (FOL) builds on propositional logic to allowing reasoning about sentences at a finer level of granularity. Whereas propositional logic only allows reasoning about specific propositions (such as 'Socrates is mortal'), FOL allows reasoning about general formulas (such as 'all men are mortal'). These languages are introduced in almost any undergraduate textbook on AI or logic, such as [13, 92, 102].

The language of propositional logic consists of a set of logical symbols, whose meaning is fixed within the language, and a set of nonlogical symbols, whose meaning is defined by the application or domain of use. The logical symbols of propositional logic consist of punctuation (parentheses) and the logical connectives: "$\wedge$" (conjunction; 'AND'); "$\vee$" (disjunction; 'OR'); and "$\neg$" (negation; 'NOT'). The only nonlogical symbols in propositional logic denote propositions, e.g., Raining is a propositional symbol that might denote the proposition that it is currently raining (whether it does or not depends on the interpretation given to such symbols). The syntax of propositional logic consists of the set of well-formed formulas (*wffs*), defined inductively by the rules:

1. if P is a propositional symbol then it is a wff;
2. if ϕ is a wff then $\neg\phi$ is a wff;
3. if ϕ and ψ are wffs then $(\phi \wedge \psi)$ is a wff;
4. if ϕ and ψ are wffs then $(\phi \vee \psi)$ is a wff;
5. nothing else is a wff (closure).

We can also define some useful abbreviations:

implication: $\phi \Rightarrow \psi$ is defined as shorthand for $\neg\phi \vee \psi$;

equivalence: $\phi \Leftrightarrow \psi$ is defined as shorthand for $(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$.

With these rules we can write sentences such as Raining $\Rightarrow$ StayIndoors, which would be interpreted as meaning "if it is raining, then I stay indoors" given appropriate meanings for the propositional symbols.

Propositional logic can be expanded to first-order predicate logic by introducing new syntax, and this language does allow us to talk about general statements. First-order predicate logic generalises propositional symbols into *predicate* symbols, and introduces a new set of *function symbols*. Function and predicate symbols have an arity, which is the number of arguments they take. A predicate of zero arity is a proposition, whereas a function of zero arity is a *constant*. Constants denote objects. For instance, the constant socrates might denote the person Socrates[2]. A function symbol denotes a function over objects in some domain. For instance, the function fatherOf might denote a function f : Person $\rightarrow$ Person. In addition, first order logic introduces new logical symbols: an infinite set of *variables*, x, y, z, etc; and the quantifiers $\forall$ ('for all') and $\exists$ ('there exists'). The syntax of FOL expands that of propositional logic with $terms$:

1. if x is a variable, then x is a term;
2. if $t_1, \ldots, t_n$ are terms, and f is a function symbol of arity n, then $f(t_1, \ldots, t_n)$ is a term;
3. nothing else is a term.

The set of wffs is then also expanded:

1. if $t_1, \ldots, t_n$ are terms and P is a predicate symbol of arity n, then $P(t_1, \ldots, t_n)$ is a wff;
2. if x is a variable and ϕ is a wff in which x occurs as a free variable, then $\forall x.\phi$ is a wff;
3. if x is a variable and ϕ is a wff (again, in which x occurs free), then $\exists x.\phi$ is a wff;
4. if t_1 and t_2 are terms, then $t_1 = t_2$ is a wff.

We can now form sentences such as $\forall x.(\mathsf{Man}(x) \Rightarrow \mathsf{Mortal}(x))$, which we can intuitively take to mean "all men are mortal". The semantics of first-order logic makes this meaning more precise. Equality of terms is not always included in the language, and there are other ways of defining it to that presented here. We include it in the basic syntax of the language in order to describe functional relationships directly in the language, e.g., $\mathsf{fatherOf}(\mathsf{neil}) = \mathsf{norman}$.

[2]We will use the convention in this thesis that predicate and propositional symbols begin with an upper-case letter, such as Raining, whereas function and constant symbols will begin with a lower-case letter, e.g., socrates

The meaning of formulas of first-order logic is given by an interpretation $\Im$, which consists of a domain of interpretation (a non-empty set of objects), $\mathcal{D}$, and an interpretation mapping, $\mathcal{I}$, that maps nonlogical symbols to functions or relations over $\mathcal{D}$. For propositional logic $\mathcal{D} = \{True, False\}$ (i.e., the concepts of truth and falsity, respectively). The denotation of terms is found by assigning elements of $\mathcal{D}$ to each variable in a formula, and then for each term of the form $f(t_1, \ldots, t_n)$ recursively substituting the denotation of each $t_1, \ldots, t_n$ and passing those as arguments to the function denoted by f (i.e., $\mathcal{I}[f]$). The denotation of term t given an interpretation $\Im$ and a variable assignment μ is written $[\![t]\!]_{\Im,\mu}$. An interpretation *satisfies* a formula ϕ if ϕ is true in that interpretation, written $\Im \models \phi$. Equivalently, we can say that $\Im$ is a *model* of ϕ. A formula of the form $P(t_1, \ldots, t_n)$ is satisfied iff the tuple of the denotations of each term t_i is in the relation $\mathcal{P} = \mathcal{I}[P]$, i.e., $\langle [\![t_1]\!]_{\Im,\mu}, \ldots, [\![t_n]\!]_{\Im,\mu} \rangle \in \mathcal{P}$. Rules for satisfaction of more complex formulas are also defined in a straight-forward way, for instance $\Im \models (\phi \wedge \psi)$ iff $\Im \models \phi$ and $\Im \models \psi$. A formula is *valid* if it is satisfied by all possible interpretations (a tautology, written $\models \phi$), *satisfiable* if it is satisfied by some interpretation, and *inconsistent* if it is satisfied by no interpretation. A set of wffs Γ *entails* a wff ϕ if ϕ is satisfied in every interpretation in which every wff in Γ is satisfied, written $\Gamma \models \phi$.

Representing Event Information in FOL

First-order logic is very expressive, and allows a great deal of precision in how we can represent the details of events we wish to describe. One of the most useful ways of representing this information in first-order logic is to use abstract, *reified*, individuals to represent events and then use unary and binary predicates or functions to associate information with these event individuals. For instance, we might describe a birthday party by the following collection of facts:

$$
\begin{aligned}
&\mathsf{BirthdayParty}(\mathsf{bp1}) \\
\wedge\ &\mathsf{host}(\mathsf{bp1}) = \mathsf{john} \\
\wedge\ &\mathsf{loc}(\mathsf{bp1}) = \mathsf{johnsHouse} \\
\wedge\ &\mathsf{Attended}(\mathsf{bp1}, \mathsf{mary}) \\
\wedge\ &\mathsf{Attended}(\mathsf{bp1}, \mathsf{nick}) \\
&\vdots
\end{aligned}
$$

This representation is flexible as we can add further details to the event easily, whereas if we used a straight-forward predicate representation such as $\mathsf{BirthdayParty}(\mathsf{john}, \mathsf{johnsHouse}, \ldots)$ we would have to decide beforehand what the relevant details were and make them part of the BirthdayParty predicate. The approach using abstract individuals is commonly used in knowledge representation.

As well as representing facts about some particular event, we can also use first-order logic to state terminological facts about types of events. For instance, we might like to state that birthday parties are a specialisation of parties in general:

$$\forall e.(\mathsf{BirthdayParty}(e) \Rightarrow \mathsf{Party}(e))$$

Or we could state that all parties must have a host:

$$\forall e.(\mathsf{Party}(e) \Rightarrow \exists p.(\mathsf{host}(e) = p))$$

It should be clear that first-order logic provides an extremely expressive framework for describing the entities and relationships present in the world we wish to narrate. Indeed, FOL can express a very large variety of information about a domain and it has been used extensively. There are some limitations to first-order logic, and there exist extended logics for e.g., quantifying over predicate symbols (second-order logic). However, FOL is expressive enough to describe most interactions occurring in an environment for the purposes of narration.

Reasoning in First-Order Logic

Reasoning in first-order logic is performed using inference rules, such as *modus ponens*: $\phi, (\phi \Rightarrow \psi) \vdash \psi$, which states that if we know ϕ and $\phi \Rightarrow \psi$ then we can conclude ψ. A set of inference rules, together with a set of basic axioms, such as $\phi \Rightarrow (\psi \Rightarrow \phi)$, forms a proof system for first-order logic. A proof system $\mathcal{P}$ is *sound* if $\mathcal{P} \vdash \phi$ implies that $\mathcal{P} \models \phi$, and *complete* if for every ϕ such that $\mathcal{P} \models \phi$ we can construct a proof (sequence of applications of inference rules) that $\mathcal{P} \vdash \phi$.

There exist sound and complete proof systems for both propositional logic and first-order logic. Furthermore, inference in propositional logic is *decideable*: given unbounded resources we can decide whether a theorem is entailed or not within a finite amount of time. However, reasoning in first-order logic is in general undecideable. This

is clearly an important drawback to using first-order logic as a general knowledge representation mechanism. However, on the other hand, propositional logic seems too weak for many purposes. Research in knowledge representation has investigated languages which offer much of the expressiveness of FOL while remaining decideable. We will explore some of these logics in following sections. It should be noted, though, that while inference in propositional logic is decideable this doesn't mean that it is necessarily computationally tractable. Indeed, inference in propositional logic is known to be NP-complete in general [31], which means that inference may take exponential time in the worst case. The implications of this are that while we should attempt to use a knowledge representation formalism that is decideable, it will always be necessary to consider performance aspects while designing the knowledge base. In practice, many problems can be encoded in such a way as to admit efficient reasoning, and many reasoning technologies exhibit good performance on a wide range of tasks (e.g., SAT solvers or description logic reasoners).

2.2.2 Horn Logic and Logic Programming

One restricted version of first-order logic is that of Horn clauses. A Horn clause is a clause (disjunction of literals) with at most one positive atom. Such a clause can be written in the form of an implication with a conjunction of atomic formulas as the antecedent and at most one atom as the consequent. For instance, the Horn clause $[\neg\mathsf{Raining}, \neg\mathsf{Weekend}, \mathsf{StayIndoors}]$ can be written as $\mathsf{Raining} \wedge \mathsf{Weekend} \Rightarrow \mathsf{StayIndoors}$. Reasoning with Horn clauses is less complex than for full first-order logic, and the propositional case can be implemented very efficiently. However, in the first-order case Horn logic is still undecideable. Despite this, it is possible to implement efficient reasoning procedures for Horn logic that work for many problems. For this reason, Horn clauses are often used in logic programming.

PROLOG

The first general-purpose logic programming language, and the most well-known, is PROLOG (PROgramming in LOGic), which was developed in the 1970s. PROLOG programs are based on Horn-clauses and rules in a PROLOG program can be viewed either declaratively (as a Horn clause) or procedurally. Evaluation takes the form of SLD-resolution which results in a backtracking depth-first search proof procedure. PROLOG programs can be compiled to efficient code and can achieve comparable performance to traditional

programming languages in some cases [101]. However, developing efficient PROLOG programs often involves the use of non-logical facilities such as *cuts* which can reduce the search space at the cost of sacrificing the declarative interpretation of rules.

PROLOG incorporates the idea of negation-as-failure, such that the system can conclude $\neg$P if it cannot construct a proof of P. This is a form of non-monotonic reasoning, known as a closed-world assumption, as if the system later acquires a fact P (though additional assertions to the knowledge base) then it will no longer be able to conclude $\neg$P. OWL (discussed later) on the other hand, uses an open-world assumption, which means that $\neg$P must be explicitly asserted. This latter assumption allows for theorems whose status is unknown by the system: it can prove neither P nor $\neg$P. The combination of Horn clauses and negation-as-failure increases the expressive power of PROLOG and, in particular, makes it quite well suited to the sort of common-sense reasoning used in inferences about actions and events.

A number of variations on PROLOG have been developed, including parallel implementations, statically-typed versions, and higher-order versions. Other languages have used different evaluation strategies (such as DATALOG, discussed later) or have extended PROLOG with facilities to support other computational models, such as agent-based programming or constraint programming. One particularly interesting extension from the point of view of generating narrative of events, is that of *abductive logic programming* (ALP). An ALP program consists of a normal PROLOG program, describing a domain, together with a set of integrity constraints for that domain, and a set of partially-described *abducible predicates*. In addition to answering queries, ALP can also generate *explanations* for observations in terms of the abducible predicates, that satisfy the integrity constraints. Once an explanation has been chosen, this then forms part of the theory and can be used in further reasoning. Such a system could be used as a means of explaining events occurring in an online environment. We will discuss other techniques in Section 2.4 (page 36).

The availability of a number of good quality, high-performance implementations of PROLOG makes it a compelling knowledge representation technology. For this reason, PROLOG is also often used as a compilation target language for other knowledge representation technologies. For instance, a number of attempts have been made at translating OWL ontologies into PROLOG rule bases.

2.2.3 Relational Databases

Probably the most widely-used knowledge representation technology is that of relational databases. Such databases are based on the relational model of data invented by Codd [26, 27]. The relational model can be seen as a cut-down form of first-order logic, in which the facts stored in relational tables in the database constitute axioms of the system, and queries represent theorems. The original relational model is fairly restricted in comparison with full first-order logic. For instance, recursion is not expressible, so formulas like the following, which expresses that all the ancestors of somebody's parents are ancestors of themselves, are not expressible:

$$\forall x.\forall y.(\mathsf{AncestorOf}(x,y) \Leftarrow \exists z.(\mathsf{ParentOf}(x,z) \wedge \mathsf{AncestorOf}(z,y))$$

In this regard, relational databases are one of the least expressive forms of knowledge representation we will discuss. The reason for these restrictions is to make query answering very efficient (and decideable). Typically the semantics of a relational database involve a closed-world assumption; that is, that the state of the world being modelled is entirely captured by the assertions in the database. Together with some other assumptions (such as a unique names assumption) this means that there is a *single* model of the database, with a one-to-one correspondence between facts in the database and entailed theorems. This semantics allows for a very efficient implementation. Indeed, query answering in the basic relational model has polynomial time complexity related to the size of the database. A number of commercial and open-source relational database management systems (RDBMSs) have been developed, offering decent performance on large data sets, and useful features such as persistence and atomic transactions. Given the relatively limited expressive power of such a database they are usually used as a component in a larger application: the database provides a central store for crucial data and provides services such as query answering, integrity constraint enforcement and other facilities; while more expressive languages are used to develop applications that make use of the data. This is usually done using an intermediate query language, such as SQL (Structured Query Language), to allow the application to communicate with the database server. Other approaches are possible, such as using the database as a component of a logic programming system: ground facts are stored in the database, and the language layers rules on top of this.

Most implementation of the relational database model go beyond the basic model, adding various features that increase the expressivity of the model or add convenience features. For instance, SQL now supports a limited form of recursive queries. A number of other features have been added on to SQL to increase its expressiveness, mostly in a fairly ad-hoc manner, and it provides support for some interesting features such as aggregate functions (e.g., sum) which can be applied to an entire column of a relation; in this case, calculating the sum of all the values in the column.

DATALOG

An alternative to SQL is the DATALOG language, which syntactically is a subset of PROLOG but with a set-based model-theoretic semantics rather than PROLOG's proof-theoretic operational semantics. In other words, DATALOG computes the *set* of answers to a given query at once, rather than calculating each answer one at a time and then backtracking to generate further answers as required. The syntactic restrictions of DATALOG in comparison to PROLOG discard programs that can lead to infinite loops in the latter, ensuring that all DATALOG queries will, in principle, terminate. The restictions also give DATALOG a purely declarative semantics. All of these features make DATALOG an excellent database language. Indeed, many DATALOG programs can be translated into equivalent Relational Algebra (RA) expressions [23], and vice-versa. The two languages are not entirely equivalent, however: DATALOG supports recursive queries. For example, the ancestor query can be expressed in DATALOG as:

```
ancestorOf(X, Y) :- parentOf(X, Y).
ancestorOf(X, Y) :- parentOf(X, Z), ancestorOf(Z, Y).
```

On the other hand, pure DATALOG has no negation and so cannot express the difference operator of RA. Like the relational model, DATALOG can perform query answering in time polynomial in the size of the database (assuming a fixed set of rules). Various extensions have been made to DATALOG to support features such as disjunction in the head of rules, or negation as failure. These features increase the expressive power of the language, but at a cost of increased time complexity (usually beyond polynomial time). Beyond DATALOG there is a class of similar languages used for Answer Set Programming (ASP)[47]. Such languages are more expressive than DATALOG and powerful answer set solvers have been

developed that allow tackling difficult (NP) search problems, allowing many queries to be answered efficiently in finite time.

A number of deductive database systems (i.e., systems mixing logic programming and database technology) exist and implement expressive extensions of DATALOG, such as DES (Datalog Educational System) [103]. The semantics of these systems provide advantages over both relational databases (such as support for complex recursive queries) and some advantages over PROLOG (such as guaranteed termination for some queries that would result in an infinite loop in PROLOG). However, most of these projects are research or teaching projects, and lack the large-scale facilities of typical relational database implementations, such as persistence, transactions, and highly optimised implementations.

For representing events, DATALOG provides much of the required expressiveness, allowing us to express many of the same sorts of terminological facts and abstract individuals that we could represent in first-order logic. It seems likely that a good proportion of the reasoning required for event recognition would fit within the expressive power of DATALOG, and using the language could provide an efficient implementation. However, in dealing with events in persistent virtual environments we need to develop agents that can keep track of a potentially large amount of information over an extended period of time. The lack of robust, persistent implementations of DATALOG limits its usefulness in practice, even if in theory it would make a good fit.

2.2.4 Description Logics

Description Logics [7, 13] are a family of logics designed specifically for describing the terminology of a domain and the individuals in that domain. Most description logics are based on restricted forms of first-order predicate logic with an emphasis on decideable reasoning procedures (although these procedures may still take exponential time in the worst case). The development of description logics has been influenced by earlier approaches to knowledge representation, such as semantic networks or frame systems[82], but within a logical framework. Different description logics have different trade-offs between expressivity and complexity of reasoning.

Description logics typically allow unary predicates (atomic *concepts*), binary predicates (*roles*), and constants (*individuals*). A *class* is a set of individuals that share some property, such as that of *being a chair* (a concept) or of *being red in colour* (a role). The syntax for

building up complex classes from these parts is restricted to prevent the use of computationally intractable constructs such as unrestricted universal or existential quantification. All description logics provide some support for subsumption relationships, such as our previous example that all birthday parties are parties. In this case, we would say that the concept of 'party' subsumes that of 'birthday party'.

A description logic system consists of two parts: a terminology (TBox), consisting of named definitions of atomic and complex concepts and roles; and a set of assertions about individuals (ABox) in terms of the concepts defined in the TBox. The semantics of description logics are described in a model-theoretic manner similar to that of first-order logic.

One of the most basic description logic languages is $\mathcal{ALC}$ (attributive language) [9]. The syntax of $\mathcal{ALC}$ is defined in terms of atomic concepts (unary predicates, such as Person) and roles (binary predicates, such as HasBrother):

1. if C is an atomic concept then it is a wff;
2. if C is an atomic concept then $\neg$C is a wff;
3. $\top$ is a wff;
4. $\bot$ is a wff;
5. if C and D are atomic concepts, then C $\sqcap$ D is a wff;
6. if ρ is a role and C is an atomic concept, then $\forall\rho$.C is a wff;
7. if ρ is a role then $\exists\rho.\top$ is a wff.

The special symbols $\top$ and $\bot$ represent a universal concept (one that subsumes everything) and a bottom concept (one that is subsumed by everything) respectively. As can be seen from the syntax, several restrictions are placed on the form of wffs in this language, such as only allowing negation of atomic concepts. We can still describe some interesting formulas using this language, such as the class of all people who have no brother: Person$\sqcap\forall$HasBrother.$\bot$. Various extensions of this language relax these restrictions, providing features such as union of concepts, full existential quantification over roles, cardinality restrictions (e.g., the class of people with no more than 2 brothers), and complex concept

negation. The basic syntactic constructs of description logics can usually be straightforwardly translated into equivalent formulas of first-order logic. For example, our concept of people with no brother can be translated as: $\forall x.(\mathsf{Person}(x) \wedge \neg\exists y.(\mathsf{HasBrother}(x, y)))$.

The TBox contains terminological axioms about concepts and roles that can take one of two forms:

- $C \sqsubseteq D$, or;
- $C \equiv D$.

where C and D are either both concepts or both roles. The first kind are inclusions, while the second kind are equalities. Concepts denote subsets of the domain of the application, and so the semantics of these axioms are defined in simple set-theoretic terms: $C \sqsubseteq D$ iff $\mathcal{I}[C] \subseteq \mathcal{I}[D]$; and, $C \equiv D$ iff $\mathcal{I}[C] = \mathcal{I}[D]$. *Definitions* are equalities whose left-hand side is an atomic concept, such as $\mathsf{OutsideParty} \equiv \mathsf{Party} \sqcap \exists\mathsf{HasLocation}.\mathsf{OutsideLocation}$[3], which defines the class of outside parties as those parties which have a location that is outside. A *terminology* (TBox) is a set of definitions in which no atomic concept is defined more than once. Most reasoning tasks against a TBox (such as satisfiability of concepts, computing subsumption relations etc.) can be reduced to satisfiability, which allows efficient SAT-solvers to be used in the implementation. Reasoning tasks against an ABox include consistency checking, retrieving all instances of a given concept (i.e., all a such that $\mathsf{ABox} \models \mathsf{C}(a)$ for some concept C), and classification (i.e., finding the most specific concepts that some individual is a member of). Unlike in most logic-programming environments, ABox semantics are open-world, so we cannot conclude $\neg\phi$ from the mere absence of ϕ, but instead need to introduce explicit extra assertions to record this fact.

It is straight-forward to translate description logic formulas into equivalent formulas of first-order logic. We can do this by defining a translation function π that translates concept names into unary predicates and roles into binary predicates, and then maps the

[3]Note that this expression goes beyond the language of $\mathcal{ALC}$ which only allows $\top$ to appear in an existential quantification.

other constructors onto equivalent first-order logic fragments (with free variable x):

$$\pi(\mathsf{C}) = \mathsf{C}(x)$$
$$\pi(\neg\mathsf{C}) = \neg\mathsf{C}(x)$$
$$\pi(\top) = x \vee \neg x$$
$$\pi(\bot) = x \wedge \neg x$$
$$\pi(\mathsf{C} \sqcap \mathsf{D}) = \pi(\mathsf{C}) \wedge \pi(\mathsf{D})$$
$$\pi(\forall\rho.\mathsf{C}) = \forall y.\rho(x,y) \Rightarrow \mathsf{C}(y)$$
$$\pi(\exists\rho.\top) = \exists y.\rho(x,y)$$

We can translate TBox and ABox assertions in a similar fashion. TBoxes are translated by universal quantification over the free variable x:

$$\pi(\mathsf{C} \sqsubseteq \mathsf{D}) = \forall x.(\pi(\mathsf{C}) \Rightarrow \pi(\mathsf{D}))$$
$$\pi(\mathsf{C} \equiv \mathsf{D}) = \forall x.(\pi(\mathsf{C}) \Leftrightarrow \pi(\mathsf{D}))$$

ABoxes are translated by substituting the free variables in role and concept translations with the specific individuals, resulting in propositional formulas. As well as this predicate logic view of DLs there is also a close correspondence with Modal Logics [8] (see next section), where concept names are translated into propositional variables and role names into modal parameters. DL interpretations can also be viewed as Kripke structures, and vice versa. Both translations are not perfect: some expressions in more sophisticated DLs are more difficult to translate. However, the existance of both translations allows for DLs to be used to express a wide range of logical theories while retaining efficient reasoning performance.

Description logics vary greatly in expressive power, up to some forms which contain features not expressible in first-order logic, and in corresponding computational complexity of reasoning. This trade-off is well explored in description logics, and this makes them a good choice for describing the ontology of a domain such as that of events in a virtual world, as we can fine-tune the exact language we need based on the expressivity requirements that arise in modeling the domain. As with relational databases, a number of efficient implementations of description logic reasoners exist, although these are somewhat less mature than the relational databases. There is also good tool support for developing ontologies in description logic languages, thanks in large part to the popularity of

OWL (the Web Ontology Language), which is based on the description logics $\mathcal{SHIF}(D)$ (OWL Lite) and $\mathcal{SHOIN}(D)$ (OWL DL), or $\mathcal{SROIQ}(D)$ in OWL 2 [57].

2.2.5 Comparison of Representation Formalisms

The various knowledge representation formalisms discussed so far in this chapter all have different features which affect the overall expressiveness of the logic. Some mechanisms can express information not expressible in other formalisms. For instance, First-Order Logic is able to express statements about infinite sets, which are not expressible in propositional logic. This section concludes the look at basic knowledge representation formalisms by attempting to characterise the relative expressive power of each, and the sorts of knowledge that each is good at encoding. The least expressive formalisms considered are the relational model that forms the basis of most current relational database systems, and DATALOG, which is used in many deductive databases. The differences between them are easy to classify, as they mostly overlap. The original relational algebra is able to express all of DATALOG except for recursive queries. On the other hand, the original DATALOG language has no negation and so is not able to express the difference operator of relational algebra. The two formalisms can be made equivalent by either introducing negation-as-failure to DATALOG, or by extending the relational model to be able to handle DATALOG's stratified recursion. PROLOG (not shown) is more expressive than DATALOG, as it removes the restrictions on recursion, leading to an undecideable language, which is Turing-complete. 'Description Logic' refers to a whole family of different representation formalisms, with different expressive properties, so are harder to classify as a whole. In general, description logics aim to admit decideable reasoning, so will always be less expressive than full FOL. Description logics often include features such as transitive roles, which seem to go beyond the relational model. However, quite expressive subsets of many description logics can be translated into (disjunctive) DATALOG. It is not clear whether an existing DL can completely subsume the expressiveness of either the relational algebra or DATALOG. This seems not to be the case for the $\mathcal{SHOIN}(D)$ logic of OWL-DL, which is unable to express event definitions such as a crime being 'an action that is performed in a region where that action is illegal' (which requires equating the location of the event with the jurisdiction of a law). Such an event can be described in DATALOG, e.g.,:

```
crime(X) :- action(X), performedAt(X,L), illegalIn(X,L).
```

The cost of increased expressiveness in logic is the increase in complexity of reasoning that accompanies it. Inference in FOL is undecideable, whereas most of the less expressive formalisms are decideable, and some have very efficient inference procedures for large sub-sets of the languages they describe.

Combining Knowledge Representations

It is becoming more common for knowledge representation technologies to be combined within an application. For instance, relational database systems are almost always employed as one component in a system that includes more expressive general purpose languages. As with a relational database, it is possible to embed a description logic reasoner as part of a larger system architecture. A number of approaches to integrating description logics with a more expressive rule language (such as a logic programming, agent programming, or ASP language) have been explored[37, 81, 38, 58, 118, 104, 86]. One approach is to translate description logic classes into equivalent rules in PROLOG or a relational database [118]. Other approaches, such as $\mathcal{AL}$-log [37] provide an integration between a description logic and a host rule language (in this case DATALOG), in which the language is divided into two parts: the classes defined in the *structural* component (the description logic) can then be used as constraints on variables in the *relational* component (the rules). For instance, the relational component might have a rule like:

```
ancestorOf(X, Y) :- parentOf(X, Z), ancestorOf(Z, Y) &
                    X : Person, Y : Person, Z : Person
```

Here, the variables X, Y, and Z are constrained to be instances of the Person concept defined in the structural component. Some work has also been started on integrating ontological reasoning, of the sort supported by description logic reasoners, with emerging *agent-oriented* programming languages, such as *Jason*[86] or *Nuin*[36].

2.3 Temporal Aspects of Events

When considering the generation of narrative, the most important aspect, beyond the characters and settings that are involved, is that of the *events* that occur and how they relate to the story. The most important distinguishing feature of events in comparison to the other elements we have so far considered is that they involve a temporal dimension: events are

characterised not just by *what* occurs, but also by *when* they occur. In the usual logical representation formalisms, such as the propositional and predicate calculi, the focus is on the truth of sentences: the truth of these sentences is independent of the time at which they are considered. When we consider events, however, we cannot easily apply the same reasoning, as we must consider aspects of the events that vary over time (for instance the position of an object can vary over time). The nature of time and how to reason about it has been the subject of a great deal of research in artificial intelligence, and more generally in logic [44]. The representations and methods developed have found applications in areas such as planning, prediction of future events, or in the explanation of sequences of events. In this section we review some of the most influential approaches.

Approaches taken to the representation of time in logic vary in a number of different dimensions. Firstly, we can distinguish between *modal* and *first-order* approaches. The former use modal temporal operators to make statements about the truth of a formula with respect to the past, present or future. The latter approaches instead reify time or events as individuals and allow making statements related to 'earlier' or 'later'. These different positions are largely similar in expressive power, but take different philosophical positions regarding the status and nature of time itself (see [44] for a discussion). In the interpretation of temporal logics, we can also distinguish between point-based and interval-based semantics, and in whether time is considered to be a linear sequence of instants (deterministic) or a branching structure (nondeterministic).

When considering temporal entities, we can distinguish those which have homogenous temporal incidence, such as states (e.g., 'Mary is asleep')—which are also true of any proper sub-interval—and those which don't, i.e., events (e.g., 'John walks to the station')—which are only true of the entire interval and not of any proper sub-interval [44]. Properties of objects that can change over time are known as *fluents*. For instance, the colour of an object can be described as a fluent, and the act of painting the object would be an event that changes the value of this fluent. The following sections review the various approaches that have been taken to representing fluents, states and events and for reasoning about changes over time. A number of important problems have been described in the area of temporal reasoning, such as the famous *frame problem* [80]. The frame problem, briefly, is the problem of determining which properties of an object *do not* change when an event occurs. For instance, painting a house typically does not (noticeably) change the mass of the house, or its position.

2.3.1 Situation Calculus

One of the earliest approaches to representing time is the Situation Calculus of McCarthy and Hayes [80, 79]. This calculus is a dialect of first-order logic extended with the notion of *situation*, which represents the complete state of the universe at a particular point in time, and *fluents* which are functions whose domain is the set of all situations. Fluents can be the values of functions, such that we might write $\mathsf{raining}(x)(s)$ to state that it is raining at location x in situation s. This can be abbreviated as $\mathsf{Raining}(x, s)$. This simple scheme allows us to start making statements about situations without needing to have complete knowledge about the state in that situation. For instance, we can make statements such as that a particular person, p is in a particular location and that it is raining there:

$$\mathsf{At}(p, x, s) \wedge \mathsf{Raining}(x, s)$$

Causality can be asserted using a special propositional fluent F, where $F(\pi, s)$ asserts that situation s will be followed (eventually) by a situation in which the propositional fluent π holds. This allows us to write statements such as the following, which states that if a person is standing outside in an area in which it is raining, then that person will become wet[4]:

$$\forall x.\forall p.\forall s.\mathsf{Raining}(x, s) \wedge \mathsf{At}(p, x, s) \wedge \mathsf{Outside}(p, s) \Rightarrow F(\lambda s'.\mathsf{Wet}(p, s'), s).$$

As well as making statements about situations, the situation calculus also allows for reasoning about actions. The notation $\mathsf{result}(p, \alpha, s)$ denotes the situation that results from person p performing action α in situation s. With this we can write sentences describing the effects of actions, such as:

$$\mathsf{Closed}(d, s) \wedge \mathsf{Unlocked}(d, s) \wedge \mathsf{NextTo}(p, d, s) \Rightarrow \mathsf{Open}(d, \mathsf{result}(p, \mathsf{openDoor}(d), s))$$

This sentence states that if a door, d, is closed but unlocked in situation s and a person, p, who is next to the door, attempts to open it then it will indeed be open in the situation that results.

In this formulation the world consists of situations, which are point-based states of the world (i.e., they describe the world at a particular instant in time) and actions are transitions between states.

[4] These examples are taken directly from [80]

A number of variations on the situation calculus have been developed over time. One such reformulation [79] treats *events* as primary and makes actions a special case of events (rather than the other way around). In this formulation, $\mathsf{result}(e, s)$ denotes the situation that results from event e occurring in situation s, and the notation $\mathsf{Occurs}(e, s)$ states that event e does occur in situation s. This allows *occurrence axioms* to be stated describing when events (*internal events*) occur. Actions (*external events*) have no occurrence axioms. The notation $\mathsf{Holds}(p, s)$ states that the propositional fluent p holds in the situation s, which allows for quantification over fluents. Likewise, $\mathsf{value}(f, s)$ gives the value of a term fluent f in the given situation. The term $\mathsf{next}(s)$ gives the next situation after s and can be defined in terms of Occurs and result for events which have an occurrence axiom: $\mathsf{Occurs}(e, s) \Rightarrow \mathsf{next}(s) = \mathsf{result}(e, s)$.

The situation calculus is a relatively simple approach that nevertheless is quite flexible in describing events and the effects of actions. It is perhaps the most well-known formalism for dealing with dynamic change in artificial intelligence. One drawback of the use of global state situations is that it is difficult to describe the effects of multiple simultaneous actions within the calculus.

2.3.2 Interval Temporal Logic

The Interval Temporal Logic [2, 3, 4] is another influential approach to modeling time in a logical framework. In contrast to the situation calculus, interval temporal logic represents time as *intervals* rather than as states of the world. This allows modelling of events that have duration rather than just instantaneous events. Events in this language are primarily descriptive in nature: the same time interval may be described by several different events, or the same event can be described at different levels of detail. It also allows us to describe events that do not change the state of the world (such as somebody preventing an object from moving).

Interval temporal logic defines seven basic relations between time intervals, along with their inverses, shown in Table 2.1. Intervals are associated with predicates of first-order logic by adding an extra interval argument, such that a predicate $P(x_1, \ldots, x_n)$ becomes $P(x_1, \ldots, x_n, t)$ where t is an interval. Negation is given a weak interpretation by default, so that $\neg P(t)$ is true iff it is not the case that P is true for the whole time of t. A stronger form of negation would say that $\neg P(t)$ is true only if P is not true at any time dur-

Relation	**Description**	**Inverse**
$\mathsf{Before}(i, j)$	i finishes before j begins	$\mathsf{After}(j, i)$
$\mathsf{Meets}(i, j)$	i finishes exactly when j begins	$\mathsf{MetBy}(j, i)$
$\mathsf{Overlaps}(i, j)$	j starts after i starts but before i finishes	$\mathsf{OverlappedBy}(j, i)$
$\mathsf{Starts}(i, j)$	i and j start at the same time	$\mathsf{StartedBy}(j, i)$
$\mathsf{During}(i, j)$	i starts after j starts and ends before j ends	$\mathsf{Contains}(j, i)$
$\mathsf{Finishes}(i, j)$	i finishes exactly when j finishes	$\mathsf{FinishedBy}(j, i)$
$\mathsf{Disjoint}(i, j)$	i and j do not overlap in any way	n/a

TABLE 2.1: Relations of Interval Temporal Logic.

ing t. This latter interpretation would lead to problems though if the truth of P changes at some point during t—in this case we would be able to conclude neither $P(t)$ nor $\neg P(t)$. The stronger form of negation can be expressed in terms of the weaker form by universal quantification over subintervals of t.

In order to allow for more flexible event descriptions, interval temporal logic reifies events as abstract individuals. This allows expressing events along the lines presented in Section 2.2.1 (page 18):

$$\exists e.(\mathsf{Party}(e) \wedge \mathsf{host}(e) = \mathsf{john} \wedge \mathsf{time}(e) = t_1 \wedge \dots)$$

As well as events, interval temporal logic also contains *action* terms, such as $\mathsf{pickUp}(\mathsf{cup1})$. These action terms are distiguished from events to allow for representing attempts to perform an action that might fail. Thus for instance, if an agent fails to pick up the cup then we would have no $\mathsf{PickUp}(e)$ event, but we would have a $\mathsf{Try}(\mathsf{pickUp}(\mathsf{cup1}), t)$ formula, where $\mathsf{Try}(\alpha, t)$ represents an attempt to perform action α during interval t. This separation allows for talking about the effects of an event succesfully occurring independently of the conditions under which an action can be expected to achieve (and thus cause the event). In reporting on events it is certainly useful to be able to characterise the effects of events, and knowledge of what actions can cause what events may itself be useful in recognition of those events.

2.3.3 Event Calculus

Another first-order logic-based calculus for dealing with events is the *Event Calculus* of Kowalski and Sergot [67, 66] (see also [106] for an alternative formulation). The event calculus avoids the global situation states of the situation calculus in favour of qualifying statements with (named) time periods. This allows for describing simultaneous and partially ordered events. The calculus is, like the situation calculus, based on an explicit representation of events in first-order logic rather than introducing modal temporal operators. The original work was aimed at developing reasoning techniques in the logic programming language PROLOG, and so is expressible in Horn-clause form making use of negation-as-failure.

Fluents in the event calculus are *reified*, that is they are first-class objects which can be quantified over and can appear as the arguments to predicates. A fluent is, as before, something which varies over time (e.g., the current temperature) or a proposition which is true at certain time (e.g., "it is hot"). Relationships (such as "possesses") hold over intervals of time and are started and terminated by events (such as "gives"), which are assumed to be instantaneous (but could actually have duration). Thus the event calculus is an interval-based approach to events rather than the point-based approach of the situation calculus (a situation is taken to be a single instant in time), and events start and end time intervals rather than indicating global state transitions.

The event calculus uses the notation $\mathsf{after}(e, f)$ to state that fluent f holds after event e occurs. For instance, $\mathsf{after}(\mathsf{give}(\mathsf{john}, \mathsf{book1}, \mathsf{mary}), \mathsf{Possesses}(\mathsf{mary}, \mathsf{book1}))$ states that, after John gives Mary a book, Mary possesses that book. The full event calculus also includes a symmetric $\mathsf{Before}(e, f)$ but this is ignored in [66] to reduce the complexity of the language. Similar to the situation calculus, the event calculus contains a predicate $\mathsf{Holds}(\mathsf{after}(e, r))$ or $\mathsf{Holds}(\mathsf{before}(e, r))$ which states that the relationship r holds for the time period indicated by the argument. Events are described usually by a series of unary and binary predicates, intended to be a logical representation of semantic network formalisms. A number of such predicates are used as standard in the event calculus, for instance $\mathsf{Happens}(e)$ which states that event e occurs, $\mathsf{Initiates}(e, r)$ which states that relationship r holds after e occurs, and $\mathsf{Terminates}(e, r)$ which states that r no longer holds after e. An example event description would be:

$$\mathsf{Happens}(\mathsf{e1}) \wedge \mathsf{act}(\mathsf{e1}) = \mathsf{buy} \wedge \mathsf{buyer}(\mathsf{e1}) = \mathsf{jeff} \wedge \mathsf{seller}(\mathsf{e1}) = \mathsf{bob} \wedge \mathsf{object}(\mathsf{e1}) = \mathsf{cart1}$$

This represents the event e1 in which Bob sells Jeff a cart. General rules can be written such as:

$$\mathsf{Holds}(\mathsf{after}(e, r)) \leftarrow \mathsf{Happens}(e), \mathsf{Initiates}(e, r).$$

along with specific rules such as:

$$\mathsf{Initiates}(e, \mathsf{Possess}(x, y)) \leftarrow \mathsf{act}(e) = \mathsf{buy}, \mathsf{buyer}(e) = x \wedge \mathsf{object}(e) = y.$$

These rules would allow us to conclude $\mathsf{Holds}(\mathsf{after}(\mathsf{e1}, \mathsf{Possess}(\mathsf{jeff}, \mathsf{cart1})))$ from e1. This sort of reasoning provides a lot of flexibility in describing events. As events are first-class, predicates can be defined for ordering, or for associating explicit times with events (e.g., $\mathsf{time}(e) =$ '7Jan2007'), or for adding modalities (e.g., $\mathsf{Planned}(e)$). Assertions in the event calculus are more localised than in the situation calculus, as they describe only what is changed by an event and not the entire situation that arises.

A predicate $\mathsf{HoldsAt}(r, t)$ is used to reason that a given relationship r holds at a certain point in time t and is defined using the persistence axiom:

$$\mathsf{HoldsAt}(r, t) \leftarrow \mathsf{Holds}(\mathsf{after}(e, r)), e < t, \mathsf{not}\ \exists e'[\mathsf{Happens}(e'), \mathsf{Terminates}(e', r), e < e', e' \leq t].$$

In other words, r holds at time point t if it holds after some event e which occurs before t and no other event e' occurs before t (and not before e) that causes r to terminate.

2.3.4 Tense Logics

Tense Logic adopts a modal approach to time, using modal operators to indicate at which times given formulae are stated to be true. The modal operators provide a view of time relative to some reference time frame (i.e., 'now'). In the original framework four such operators are defined [44]:

Pϕ: 'It has at some time been the case that ϕ'

Fϕ: 'It will at some time be the case that ϕ'

Hϕ: 'It has always been the case that ϕ'

Gϕ: 'It will always be the case that ϕ'

The semantics of tense logic are given in terms of *temporal frames*, which consist of a set of times, $\mathcal{T}$ and an ordering, $<$, on $\mathcal{T}$. An interpretation assigns a truth value to each atomic

Tense Logic Formula	First-Order Equivalent
$\mathbf{P}\phi$	$\exists t.(t < \textbf{NOW} \wedge \phi(t))$
$\mathbf{F}\phi$	$\exists t.(\textbf{NOW} < t \wedge \phi(t))$
$\mathbf{H}\phi$	$\forall t.(t < \textbf{NOW} \Rightarrow \phi(t))$
$\mathbf{G}\phi$	$\forall t.(\textbf{NOW} < t \Rightarrow \phi(t))$

TABLE 2.2: Tense Logic formulae and equivalent first-order formulae.

formula, ϕ at each time in $\mathcal{T}$. For example, $\mathbf{P}\phi$ is true at time t if and only if ϕ is true at some time t' such that $t' < t$. An equivalence can also be given between modal tense logic and a first-order logic extended with temporal arguments together with a reference point (**NOW**) and a binary temporal ordering relation ('before', $<$) shown in Table 2.2.

A number of extensions to Tense Logic have been made, introducing new operators such as the binary 'since' and 'until' operators (**S** and **U** respectively), and the unary 'next time' operator, **O**. **S** p q means 'q has been true since a time when p was true', and **U** p q means 'q will be true until a time when p is true'. The next-time operator **O** states that the given formula will be true in the next time step (assuming a discrete semantics of time).

A number of attempts have been made to incorporate temporal and tense logics into popular description logics, to allow a combination of ontological and temporal reasoning [6].

2.4 Event and Activity Recognition

The previous sections have described a number of formal approaches to describing the objects, events and activities of individuals that will form the basis for generating narrative from virtual worlds. In this section we concentrate on the task of *recognising* these events and activities, and comprehending their significance at a higher-level, suitable for incorporation into a narrative. Event comprehension involves recognising at a higher level the significance of a sequence of lower-level events in terms of the possible reasons, effects and objectives behind these actions. This involves generating a hypothesis h (or more generally, a set of hypotheses) that is an *explanation* for a series of observations $\alpha_1, \ldots, \alpha_n$ given some background theory T. There are a number of obvious constraints placed on

candidate hypotheses:

1. The hypothesis must be sufficient to explain the observations (i.e., $T \cup \{h\} \models \alpha_1, \ldots, \alpha_n$);
2. The hypothesis is consistent with the background theory (i.e., $T \not\models \neg h$);
3. The hypothesis is formulated in the appropriate terms (i.e., in terms of plans, goals, etc. rather than just simply the observations themselves);
4. The hypothesis is in some sense *minimal*: it doesn't involve any terms which are not necessary in explaining the observations.

In finding such an explanation we are essentially carrying out a form of non-deductive reasoning. A number of approaches to such reasoning are possible, such as abduction (from q and $p \Rightarrow q$ posit p), circumscription, default reasoning, and techniques based on probability theory. In this section, we review the literature from the areas of *plan* and *activity* recognition and examine a selection of techniques using some of these methods.

2.4.1 Plan Recognition

The plan recognition problem was first described in [105] and a number of approaches have since been described in the literature (see [21] for a general survey). The basic plan recognition problem is this: given a series of observations of actions performed in some environment, $\alpha_1, \ldots, \alpha_n$, come up with a set of hypotheses, $\{H_1, \ldots, H_m\}$ that best explain these observations in terms of higher-level *goals* that agents may be trying to accomplish, and the *plans* they may be using to achieve these goals. It may be the case that there are multiple conflicting explanations for a given set of observations. Some method is therefore needed to choose between competing hypotheses, and a number of approaches have been developed, based on techniques such as circumscription (for minimising the number of hypotheses), using heuristics, and employing probabilistic techniques. In this section we review a number of such approaches.

There are a number of factors that influence the choice of plan recognition techniques: actions may only be partially observable (i.e., we may fail to observe some actions, or fail to observe their effects); there may be multiple agents in the environment with conflicting or overlapping plans and goals; a particular action may form a constituent of several plausible plans; etc. The simplest plan recognition tasks involve a single user in

a restricted domain, where the user is aware of the plan recognition process and actively supports it (*intended* plan recognition). At the other end of the spectrum there are *keyhole* plan recognition tasks, in which the users do not cooperate in the recognition process, or their actions are only partially observable.

A Formal Theory of Plan Recognition

The overall problem of plan recognition is described well in Kautz's PhD thesis [63] and subsequent work [64]. This work clearly separated out the description of the problem in a formal way from particular approaches to solving it. The problem is formulated in terms of a language of events and an event hierarchy that decomposes events into layers of abstraction. The language is based in part on Allen's interval temporal logic, described in Section 2.3.2 (page 32). The event hierarchy is described in first-order predicate logic in a manner similar to the approach of Section 2.2.1 (page 18). The task of recognition is performed by combining the event hierarchy with a set of observations and then adding "covering" assumptions to link each observed event to some part of the event hierarchy. Within the event hierachy there is a distinguished event type End whose sub-types are the events that we are interested in. A covering model then tries to categorise each observed event so that it is part of some End event.

The event hierarchy, H consists of:

- H_E: a set of unary event type predicates;
- H_A: a set of abstraction axioms, of the form $\forall x.\mathsf{E1}(x) \Rightarrow \mathsf{E2}(x)$;
- H_{EB}: the set of basic event types (i.e., those having no subtypes which are directly observable);
- H_D: a set of decomposition axioms describing the components parts of an event (i.e., something like a plan description);
- H_G: general axioms about the world not related to events.

An example of a decomposition axiom (plan) in this scheme is the following [63]:

$$\begin{aligned}\forall x.\mathsf{MakePastaDish}(x) \Rightarrow & \\ & \mathsf{MakeNoodles}(\mathsf{step1}(x)) \\ & \wedge\ \mathsf{MakeSauce}(\mathsf{step2}(x)) \\ & \wedge\ \mathsf{Boil}(\mathsf{step3}(x)) \\ & \wedge\ \mathsf{agent}(\mathsf{step1}(x)) = \mathsf{agent}(x) \\ & \wedge\ \mathsf{result}(\mathsf{step1}(x)) = \mathsf{input}(\mathsf{step3}(x)) \\ & \wedge\ \mathsf{During}(\mathsf{time}(\mathsf{step1}(x)), \mathsf{time}(x)) \\ & \vdots\end{aligned}$$

These axioms specify not just the steps of the plan, but also constraints on the individuals involved in each step, temporal constraints, and preconditions and effects. The structure of such decomposition axioms make it clear that the actions that make up a plan comprise only necessary conditions (i.e., if someone is following this plan then they necessarily will perform the given actions), and not sufficient conditions. For some types of event it may be possible to strengthen these axioms to also provide sufficient conditions (i.e., if someone performs these actions then they must be following the given plan). Some of the higher-level events that we might wish to narrate could actually be formulated in this manner (e.g., we could formulate sufficient conditions for determining when somebody has committed a crime), but in general there will be some events that we cannot easily describe in this way. The general area of plan recognition concentrates mainly on the types of problems for which sufficient conditions cannot be specified. In such cases we must use non-deductive inference techniques to determine likely explanations for observed events. The approach taken by Kautz is based on the idea of minimum covering models.

A *covering model* is a model of the event hierarchy in which each non-End event occurs only as a component of some other event. A covering model of some observation therefore is an explanation of that event in terms of End events. Propositions which hold in all covering models of an observation are *c-entailed* by the observation. Covering models are constructed by an application of predicate circumscription, minimising the predicates in the event hierarchy. Three assumptions are used in this process:

1. that there are no event types outside of H_E;
2. that all abstraction relations between event types are captured by H_A;
3. that all non-End events occur only as components of other events.

Covering models are combined to produce *minimum covering models* which minimise the *number* of End events used to explain several observations, using a slightly different form of circumscription. A model M is a minimum covering model of a set of observations Γ with respect to a hierarchy H iff: M is a model of Γ; M is a covering model of H; and, M has mimimum cardinaility in End events among covering models of H. A proposition which is true in all minimal covering models of Γ is *mc-entailed* by Γ ($\Gamma_H \models_{mc} p$). Intuitively, this notion of mc-entailment tries to explain a set of observations Γ by assigning each element of Γ to a component of the set of End events such that we minimise the number of such events required to explain the observations. This captures the criteria we specified at the beginning of Section 2.4 (page 36). However, the theory has some problems when applied to incremental plan recognition. As new observations are added they may require the number of End events used to be increased. In this case, the theory gives no guidance on how we should restructure the explanations. (Should existing observations be reassigned to the new End events, or should we leave the original explanations as-is?) A number of heuristics are outlined which provide different approaches to this problem. For instance, we can only assign new observations to new End events, or we could try and group new observations with the most recent observation with which they are consistent (the *sticky covers* heuristic). Another approach would be to use some quantitative method to rank groupings of observations by likelihood (i.e., the probability that some observation contributes to a particular event). Such a quantitative measure can be applied after incremental minimum covering models have selected a set of candidate End events. A form of incremental mc-entailment is developed which is monotonic: conclusions derived from earlier evidence are never retracted. This is known as *imc-entailment*, and provides a reasonable heuristic for plan recognition.

After this semantic characterisation of the process of plan recognition, the work goes on to describe algorithms for assigning observations to End events using *explanation graphs*, and algorithms for incrementally revising these graphs to account for new observations. The worst-case complexity of these algorithms is analysed in terms of the size of the

graph data structures created (which also gives a rough estimate of the time complexity in this case). These worst-case complexities are quite high (exponential in the size of the event hierarchy in some cases), although the implementation performs adequately on a number of test scenarios. Despite these concerns, the work is an excellent description of the plan recognition problem and presents a very clear approach to implementing a solution.

T-REX

The T-REX system developed by Robert Weida [119] applies concepts from description logics and knowledge representation to the problem of plan recognition. The work builds on the approach described by Kautz, but offers a number of advantages. A T-REX plan consists of:

- A set of action steps. Each step is an action described in a description logic taxonomy;
- A set of qualitative constraints on these steps, based on Allen's interval relations;
- Metric constraints on the steps, specifying properties such as their maximum duration, or the maximum time between steps;
- Coreference constraints, specifying e.g., that the same agent must perform each step in a plan.

These constraints can be checked for validity by the system before use, to ensure that vacuous or equivalent plans are eliminated and that plans are satisfiable. In addition, T-REX extends the notion of subsumption from description logics to include plans and actions. This allows a plan library to be automatically classified, and T-REX can use this information during plan recognition. For instance if plan A subsumes plan B then if a set of observations supports plan B we know that they also support plan A, even if the observations do not directly support this. In addition, the system supports subsumption-based querying of the plan library to select sets of plans that match some conditions, and domain-specific constraints can be added to plans using a rule-based system that matches against the plan library. These capabilities greatly enhance the flexibility of the plan library and the recognition process.

T-REX plan recognition identifies all plans that are consistent with a set of observations. In the case that multiple plans are consistent with the observations, T-REX makes

no attempt to pick a most likely plan. An application-specific process can then make this decision, if needed, employing either heuristics or probabilistic reasoning or some other technique. As in Kautz's work, plan recognition assumes completeness of the plan library. This assumption allows some addition inferences to be made: if a set of plans is consistent with a set of observations, then we can assume any property which is true of all of those plans. Such assumptions are not actually made by T-REX itself, and as a form of non-monotonic reasoning it would require tracking those assumptions (for instance, using a Assumption-based Truth Maintenance System) in case they turned out to be false (which could happen if the observed agent was following a plan not recorded in the plan library).

T-REX is implemented on top of two main sub-components: a K-REP or CLASSIC description logic reasoner is used to reason about actions and related concepts, while the MATS system is used for temporal reasoning (qualitative and metric constraints). T-REX itself does any other reasoning (such as checking coreference constraints).

The T-REX system provides a good example of how description logics can be combined with temporal reasoning to create a powerful approach to plan recognition. The main contribution of the work is in developing an expressive representation of a plan library, and how this can contribute to the flexibility of the plan recognition process. The development of a notion of subsumption for plans in addition to individual events, is also a major contribution that goes beyond the previous work.

2.5 Generating Narrative Prose

A number of approaches to generating narratives have been developed, and in this section we review some of the most relevant to this thesis. While narrative is not limited to just prose text (e.g., poems, films and theatre can all be viewed as different forms, or media, of narrative), most computer generation of narrative has concentrated on this medium. Text also makes a good first choice for generating narrative from games as the techniques for generation of natural language text are well researched, and text is a flexible format that can be used for a variety of output devices (e.g., web pages, SMS text messages, instant messaging, etc). In addition, some other interesting possibilities, such as a 3D 'talking head' narrating reports from a virtual world, would likely take a natural language text, suitably marked up, as input. In this section we will therefore first provide a brief overview of natural language generation (NLG) techniques and then describe a number of

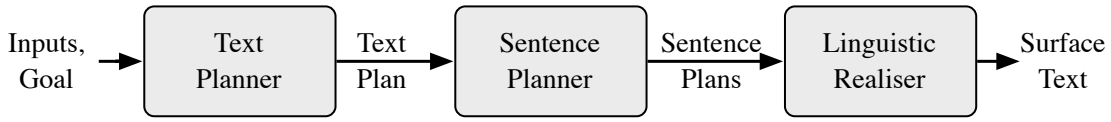

FIGURE 2.2: Architecture of a typical NLG system.

approaches to generating narrative.

2.5.1 Overview of NLG Techniques

Natural language generation (NLG) is a well established area of research, and a number of systems have been built that demonstrate increasingly sophisticated text generation abilities. Approaches to NLG range in complexity from simple template-based systems in which the text is largely pre-canned and the input is used only to parameterise the final output (e.g., much like 'mail merge' functionality of word processors), up to complex systems in which highly tailored output can be generated from a sophisticated representation of the structure of the messages to be conveyed and knowledge of grammar and terminology appropriate for the domain of use and the target natural language. Text templating systems are fairly straight forward to implement, and a number of technologies in wide-spread use in industry follow the same general idea. For instance, various webpage templating systems exist, such as PHP or JSP, that allow a document to be written by an author and then small snippets of procedural code inserted to fill in details relevant to each particular reader (e.g., their name or their current bank account balance).

At the other end of the spectrum from simple text templates are full blown NLG systems. In this section we will provide a brief sketch of the typical design and operation of such systems, without delving too much into the details. A good overview of NLG techniques from an applied perspective is given in [99]. One key point is that there is not a hard division between text templating techniques and more sophisticated NLG packages. Rather, these should be considered points on a scale of sophistication. A text templating engine can be made incrementally more sophisticated, allowing for more interesting forms of variation in the output produced. In this section we will concentrate on the basic architecture and workflow of a typical 'full blown' NLG system, with the idea that even if a full blown solution is not employed, some of the same techniques can be adapted to a less complex text templating approach.

The architecture of a typical NLG system is usually conceived as a three-stage pipeline, as shown in Figure 2.2. The three stages each consist of a number of tasks taken from a 6 stage overall workflow, as follows:

1. **Text Planning** is concerned with deciding which set of *messages* are to be conveyed, in a process known as *content discrimination*, and then deciding on the order and structure in which those messages will be described, in *discourse planning*. Content discrimination is a process of filtering and summarising input data into a set of message structures described in a suitable formal language (such as a description logic or a frame system). Such messages consist of entities in the domain of discourse, along with domain-specific concepts and relations. Discourse planning then adds structure to this set of messages, typically by constructing a tree structure, in which leaf nodes are individual messages and internal nodes represent various *discourse relations*, such as sequencing or elaboration, which describe how messages relate to each other, and the order in which they should appear.

2. **Sentence Planning** is then concerned with forming actual sentences. This consists of three processes: (i) *sentence aggregation* combines messages together into single sentences (not essential, but can improve the quality of the resulting text); (ii) *lexicalisation* chooses words and phrases to represent domain concepts and relations; and, (iii) *referring expression generation* picks words and phrases to refer to individual entities in the domain. This last process can make use of a 'discourse history' describing previous communications with a user, to allow for instance use of pronouns where appropriate.

3. **Linguistic Realisation** is the final stage, in which grammar and other rules are applied to produce syntactically correct texts, as well as taking care of capitalisation and pluralisation.

Intermediate representation formats are used to communicate between each stage in the pipeline. Typically text plans are represented as trees, as described above. In the context of a system for producing narrative from virtual environments, the output of the event recognition process would be at this stage of the pipeline: i.e., event recognition would produce a tree-like structure where leaf nodes represent basic events and actions that have occurred and the rest of the structure of the tree would group these into higher-level events

and plan structures. Sentence plans are used between the sentence planning and linguistic realisation phases. These plans can be simple text templates in which parameters can be filled from messages in the text plan, or they can be more abstract structures that describe the linguistic structure of the text to be produced.

Different techniques are used in each of the stages. Content discrimination is typically based on domain specific heuristic rules, gathered from existing human experts in the particular domain. These rules are used to determine which messages should be conveyed from the input data. Discourse planning can make use of more domain-independent theories of various discourse relations, such as rhetorical structure theory (RST), which group and order messages to provide good style and structure to the resulting text. Beyond generating basic prose, some recent work has been done on introducing variation into the generation process [83, 85, 84], particularly in the area of interactive fiction.

2.5.2 Reporting Agents

The closest work to that presented in this thesis is the reporting agents framework developed by Daniel Fielding and applied to the task of reporting on events occurring in the *Unreal Tournament* fast-paced action game [74, 41]. The framework consists of multiple *reporter* agents that are embodied within an environment and can recognise simple events and form reports on those events. These reports are passed to an *editor* agent which tries to verify the accuracy of reports by cross-checking multiple reports of the same event from different reporters, and then collates reports and passes them on to one or more *presenter* agents that are specialised in arranging and formatting reports in a manner suitable for some output medium (e.g., a live IRC chat channel, or a post-match website report). The editor agent can also act as a centralised co-ordinator, directing reporting agents in order to ensure good coverage of events (e.g., to cover for a reporter that has been killed during the game).

The approach taken in this framework was inspired by the structure of traditional news organisations. The framework is able to detect, categorise, and comprehend (at a shallow level) the events occurring within the *Unreal Tournament* game in order to form reports of these events. The sorts of events that are reported on include deaths of players, flag captures and drops (in 'capture-the-flag' games), and point scores. Reporter agents directly observe these events, or infer them using some simple rules (e.g., if the

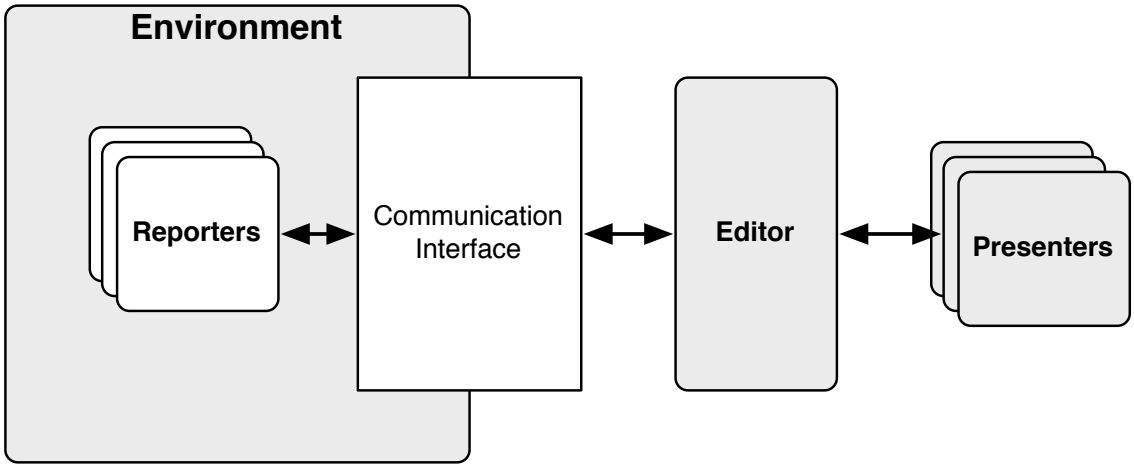

FIGURE 2.3: Architecture of original reporting agents framework.

score has just changed then one team must have captured a flag). Editor agents then combine reports from multiple reporters, checking that the details are the same in each report, and combining them to create a single report, before passing this to a presenter agent that is specialised for a particular output medium. Two main output presenters were implemented: one that produced a live IRC chat output, and another that produced a post-game one-paragraph summary in HTML format, highlighting notable events in the game. The multi-agent architecture of the system is shown in Figure 2.5.2.

Reporters

Reporter agents are responsible for the main work of detecting, recognising and characterising events that are occurring in the game world. These agents are embodied within the environment they are observing in the same manner as other participants. A different approach would be to develop a single omniscient process that collects perfect information about all activities occurring within the environment. The embodied approach was adopted because it offered the following advantages:

- Embodied reporter agents do not require privileged access to the environment they are reporting on: they are embodied as ordinary participants are;
- A team of reporter agents scales better than a central omniscient process, especially in large environments where there may be a very large number of events occurring

at a high rate;

- Participants in the environment can be aware of an embodied reporter, and adjust their behaviour when reporters are present.

The last point is important where there are issues of privacy and secrecy, which is the case in many games. Players may wish to keep details of strategy secret from reporters in order to not give away vital information to their opponents. It is therefore important that players be able to recognise when reporters are present, so that they can adapt their behaviour appropriately. More generally, participants may have many reasons for shielding themselves from reporter agents. With embodied agents, the players have some measure of control over what information is reported about them. This control is not present with an omniscient process. Such control is an important part of responsible reporting of events to an external audience.

Each reporter agent consists of an *avatar* which is the in-world representation of the agent, and a set of procedures and rules for detecting and recognising events occurring in the game world. Typically the avatar is provided by the game environment itself, using existing capabilities provided for human participants. This avatar is then controlled by reporter agents by sending commands to the game engine to perform actions on behalf of the reporter, and to retrieve information about what the reporter can currently sense of the environment. The main components of a reporter agent are:

- A collection of state describing the current situation of the avatar (location, health, etc), needed for basic action selection;
- An interface used to communicate with the in-game avatar and control it;
- A set of basic tasks that the agent can perform (e.g., following players, watching a location, etc), which can be prioritised;
- A set of event types that the agent can recognise, which can also be prioritised;
- A working memory, for storing information sensed from the environment, current beliefs, and deductions based on these beliefs;
- A set of rules for recognising events, collecting details on events, reasoning about causes for events, deciding on tasks to perform, and communicating with an editor agent.

The working memory and rules are implemented using a forward-chaining production rule system [108]. The rules are organised into distinct sets covering different functional roles, such as movement, event recognition, and evidence evaluation.

As described above, agents are parameterised by the set of tasks which they can perform and the set of event types they report on, and these can be prioritised. This allows creating reporters that are specialised or show a preference for certain types of events or tasks, allowing agents to be given different roles in the overall task of covering events in the world. Reporter agents can also receive instructions from editor agents, as described in the next section. This instructions suggest tasks which the reporter should perform, and allows editors to control the distribution and actions of reporters in order to ensure good coverage.

Editors

Editor agents' primary responsibility is to collate and edit reports of events coming from multiple reporters in the game environment. However, they can also serve a second purpose, which is to supervise the behaviour of the reporter agents, directing them in order to ensure good coverage of different areas, players and event types. Typically, a single editor agent supervises a team of reporter agents, and passes on edited reports to one or more presenter agents. Other configurations are also possible, such as having multiple editors control separate teams of reporters; perhaps covering distinct areas of the environment, or with responsibility for different types of events.

Unlike reporter agents, editors are not embodied in the game environment, and have no avatar representation. Reporters communicate with the editor through some communication interface. The details of this interface are not defined in the original framework, but left up to integration with individual environments. The interface to the *Unreal Tournament* game is implemented using the Gamebots interface [45], and uses the in-game chat channels for communication between agents. Editor agents keep track of descriptions of the reporters under their command, and the status of each of these reporters. They have rules and procedures for dealing with communication with reporters, and for collating reports of the same event from different reporters, and assessing the accuracy of these reports.

Reports of events are structured as simple facts, represented as lists. The head of

the list contains a symbol representing the type of event, while the rest of the list contains attribute-value pairs describing the details of the event. Each event also has a slot for storing any direct evidence of the event (if the reporter directly witnessed the event, as opposed to inferring it from other facts), and a qualitative certainty value, which is used for assessing the credibility of the report. Editors only perform fairly basic operations on individual event reports before passing them on to presenter agents, and events cannot be combined to create more complex events.

Editors can assign roles to reporters, which are suggestions for the focus of the reporter's attention. Roles are tasks, allowing the editor to override the task priority in reporter agents. The roles and tasks in the original framework are tailored to the particular domain of UT CTF games.

In addition to assigning roles and receiving information from reporters, editors can also receive requests for information from presenters (e.g., what the current score of a particular player is). These requests are dealt with either by consulting known information, or by broadcasting a request to the reporters to see if any of them know the information.

Presenters

Presenter agents are the primary interface between the reporting agents framework and the audience that the reports are created for. Presenters are specialised for a particular output medium and target audience. Presenters exist for relating events in real-time to a chat medium (IRC), and for creating short post-game summaries as prose text, suitable for posting to a website.

Presenter agents are the simplest component of the architecture, being concerned solely with extracting suitable details from reports and formatting these appropriately for a certain output medium. Reports are passed to presenters as declarative structures describing the type of event, and the specific details. Presenters then decide if the audience is interested in the event (based either on the type, or specific details) and if so, add it to the report, generated either in real-time or post-game. Presenters may also send requests to editor agents asking for specific information, as described in the last section. The implementation of the framework is outlined in Figure 2.5.2. The agents are implemented using the SIM_AGENT toolkit, and the game interface uses the existing Gamebots network pro-

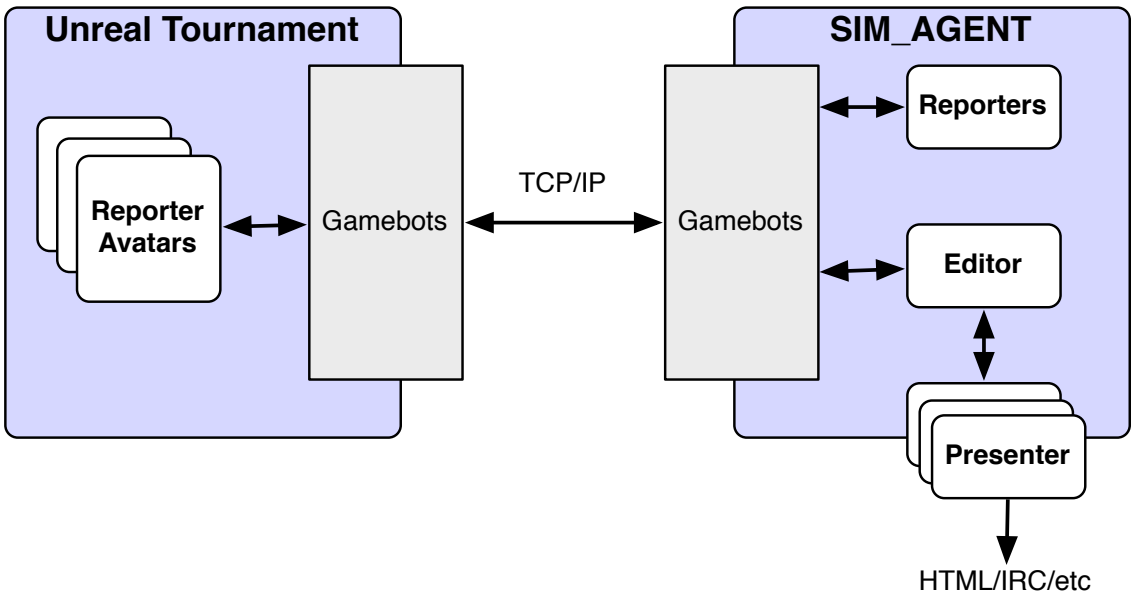

FIGURE 2.4: Implementation of the original framework.

tocol for *Unreal Tournament*.

This reporting agents framework forms the basis from which the work presented in this thesis has been developed. In particular, the present work builds on the notion of an embodied multi-agent team consisting of reporters, editors and presenters. However, the current work goes much farther than the original framework, in particular developing the event recognition and report generation capabilities to be able to handle much more sophisticated events and reports. The current work also aims to handle large-scale, complex, and persistent virtual environments, where the original framework was only applied to very short-lived (typically 10–20 minutes) games involving only a handful of participants. The current work aims to handle persistent games lasting weeks or months, and involving tens to hundreds of participants.

CHAPTER 3

BACKGROUND: MULTI-AGENT SYSTEMS

3.1 Introduction

In developing the narrative generation framework that is described in this thesis, a key early design decision was to adopt an *agent-oriented* approach. This chapter therefore provides some background on that decision, and reviews the literature relating to *intelligent agents* and *multi-agent systems* (MAS), firstly in terms of general concepts and theories, and then looking at specific software architectures that have been proposed.

A variety of different definitions of the term 'agent' have been proposed, emphasising different aspects of what constitutes agency, and what is fundamental to 'agent-oriented' design and development. Most definitions agree at least that an agent is an *autonomous* entity that is *situated* within an environment, and acts in pursuit of its *goals*. Such an agent is autonomous in that it conceptually has its own thread of control, and so (in contrast to a passive object in object-oriented software development) can initiate actions rather than simply responding to events. Autonomy also implies that an agent is free to decide whether and how to fulfill requests that are made of it, again in contrast to an object that slavishly obeys any commands (messages) that it is sent. An agent is situated in that it can perceive events occuring in an environment, which may be shared with other agents and processes. Finally, an agent generally acts in pursuit of its own goals and objectives (although these may have been delegated), rather than simply obeying a fixed set of commands.

As a view of complex systems, an agent-oriented approach takes its roots in the *intentional stance* of the philosopher Daniel Dennett[35]. In this view, it can be useful to describe a complex system not only in terms of its physical implementation (the physical

stance) or in more general terms of its design (the design stance), but also by considering the system as an intentional agent, with goals, beliefs, intentions, and other mental states. Such a view is not intended to be a statement of the reality of such ascriptions to a particular system, but rather a recognition that it can be useful to view a complex system in such terms. As an approach to software design and development, an agent-based methodology can be viewed as a particular approach to developing concurrent or distributed systems based on the idea of autonomous agents (processes) that act individually and collectively, communicating via message passing, rather than through shared state or other mechanisms. More sophisticated approaches to agent design attempt to formalise the concepts of the intentional stance, and implement concrete data structures corresponding to beliefs, desires, plans, and so on. A number of the most important such agent architectures will be reviewed in this chapter.

A key question that must be addressed when employing a particular design methodology is what advantages does it present over other approaches, and in particular what advantages an agent-oriented approach would have to the problem of narrating events occurring in virtual environments. One advantage of agent-oriented development in contrast with other common software development methodologies, such as object-oriented analysis and design, is that it explicitly addresses the concerns of building large-scale and distributed software entities. In particular, the emphasis on decomposing problems using the basic building blocks of autonomous agents, and the use of message-passing for communication between agents, naturally leads to development of loosely coupled and robust systems, where the failure of a particular component (agent) generally doesn't prevent other components from performing useful work. Of course, no design methodology can entirely eliminate critical dependencies and the potential for single points of failure having a large impact on performance of the system as a whole, but the emphasis on autonomous agents helps to reduce this likelihood. Additionally, the use of message-passing for communication between agents also reduces the complexity of developing concurrent systems and it is conceptually straightforward to move from a concurrent to a distributed system, where messages are passed over a network (or other medium) rather than simply to other processes within the same operating environment. With particular regard to generating narrative from virtual environments, the key benefits of an agent-oriented approach relate not to the software development advantages (although these still apply), but to the fact that agents are embodied within the environment being narrated, and so participants

have a degree of control over what gets reported, and can even interact with the agents to shape the narrative being produced. This interaction between narrator and participants provides a crucial aspect of the research, and lies at the heart of why this is *collaborative* narrative generation, and not merely reporting.

In the next section we review the theoretical belief-desire-intention (BDI) model of agents that has been developed in an attempt to formalise the various 'folk psychological' terms that are used to describe agents: beliefs, intentions, capabilities, and so on. While other formal agent models have also been developed, the BDI model is the most commonly used as a basis for agent architectures and languages. After reviewing the theory we describe the practice by reviewing the concrete agent *architectures* that have been proposed, ranging from simple reactive agents to more complex deliberative and proactive agents. In section 3.3 we then turn our attention to societies of agents and multi-agent systems (MAS).

3.1.1 BDI Logics

One of the most successful agent models has been the *belief-desire-intention* (BDI) model, which has been the basis of the widely-known BDI agent architecture, discussed in the next section. This model, like other models of intentional action (e.g., [28]), builds upon the work of Michael Bratman in formulating a theory of practical reasoning based on the notion of individual intention [15, 16], beliefs and desires. Beliefs correspond to information that an agent has about its environment. Desires are states of the world that the agent would ideally like to achieve. Intentions are then desires that an agent has actually committed to achieving. The separation between desires and intentions captures the idea that a real agent has limited capacity and resources, and so cannot commit to achieving every desire it might have. Instead, the agent must decide which desires to commit to and how to achieve them. Thus, it is a theory of *practical reasoning* [14]. Intentions serve to focus the reasoning of the agent as it can quickly discard any desires that conflict with its existing intentions.

The BDI model has been formalised in a family of BDI logics [96, 97, 98] that capture the notions of belief, desire and intention in a multi-modal logic. The semantics of the logic is based on a possible worlds semantics, as usual for modal logics, where each possible world is itself a complex branching temporal structure (contrasted with Cohen

and Levesque's model in which possible worlds are linear sequences of states), where each instantaneous time point in a particular world is a *situation*. Each branch represents a choice of action available to the agent at that time point, and transitions are labelled with whether the attempted action failed or succeeded. The logic is based on a variation of Computational Tree Logic (CTL*, [39]) together with modal operators for belief (**BEL** i ϕ), desire (**DES** i ϕ), and intention (**INTEND** i ϕ), indexed by an agent i and a proposition ϕ. The temporal language also includes modal operators to state that a particular path formula in the time tree is *optional* (**E** ϕ) or *inevitable* (**A** ϕ, i.e., ϕ is true of all paths from that point). The standard temporal operators $\bigcirc$ ("next"), $\Diamond$ ("eventually"), $\Box$ ("always"), and **U** ("until") are also defined. A belief accessibility relation $\mathbf{B}_t^w(i)$ associates an agent i with the set of worlds that the agent believes are possible (are *belief-accessible*) in the situation given by $\langle t, w \rangle$ (a time-point t in world w). Notions of goal-accessible worlds and intention-accessible worlds are similarly defined. Goals should be consistent and achievable, leading to goal-accessible worlds being sub-worlds of belief-accessible worlds. Likewise, intention-accessible worlds must be sub-worlds of goal-accessible worlds, to ensure intentions are compatible with goals (i.e., that an agent doesn't commit to something that is not one of its desires). Axioms for the logic are introduced to ensure compatibility between beliefs and desires and intentions, that goals and intentions are believed to be goals and intentions, and to ensure that the agent does eventually act to achieve its goals. Further, various intention revision strategies can be defined to describe how an agent's current intentions should influence its adoption of future intentions. The authors show how the logic can be used to formalise many of the same properties as Cohen and Levesque's formalism, but treating intention as fundamental rather than derived, which they claim leads to fewer cases in which an agent adopts goals as unwanted side-effects.

3.2 Agent Architectures

The use of modal logics and possible worlds semantics in most agent models leads to a clear and mathematically elegant description, but poses a number of problems in the practical application of theory to real-world agent specification and implementation. Firstly, most theories assume logical omniscience: an agent believes every formula that is entailed, and likewise for desires and intentions. Clearly, this is a problem for a practical, resource-constrained agent! Secondly, there is no obvious connection between a possible worlds

semantics and any concrete computational implementation. Just how should these theories be implemented, and what the relationship will be between the implementation and the theory is not clear. In this section we review a number of concrete agent architectures, both theoretically based and more ad-hoc designs. Architectures have been developed that span the range from reactive to deliberative, proactive, and even reflective agents.

3.2.1 Reactive Agent Architectures

$$\mathsf{react} : \mathsf{Percept} \rightarrow \mathsf{Action}$$

FIGURE 3.1: A simple reactive agent.

The most fundamental aspect of any agent is its ability to perceive and respond to events occuring within the environment in which it is situated. For instance, a robot exploring Mars must be capable of sensing when it has bumped into an obstacle, and to take action to avoid the obstacle. Even agents that are situated in an environment abstracted from the physical world, such as an email processing agent, must be able to detect and respond to events (e.g., new mail arriving). Agents which simply respond to events as they occur, are known as *reactive* agents (or stimulus-response agents). Figure 3.1 shows an example function signature react that illustrates a simple reactive agent: whenever the agent perceives something new (a *percept*), it calls the react function with the percept and receives back some *action* to execute in response.[1] Of course, we have deliberately left out much of the details of how percepts and actions are represented, or how actions are executed, and so on. The key idea is that the react function maps fairly directly from a percept to a particular action response. The implementation of the react function is the *agent program* and defines how this particular agent responds to the environment and pursues its goals. The rest of the code, which has been left out, such as how the agent perceives the environment, how it performs actions, and so on, is known as the *agent architecture* ([102], §2.3).

How the react function is implemented is also left open. At the simplest level, this function would be a real function: given the same percept, it would always propose

[1] For simplicity we only consider agents reacting to discrete events, rather than systems reacting to continuous signals.

$$\mathsf{sense} : \mathsf{Percept} \times \mathsf{WorldModel} \rightarrow \mathsf{WorldModel}$$
$$\mathsf{act} : \mathsf{WorldModel} \rightarrow \mathsf{Action}$$

FIGURE 3.2: A reactive agent with memory.

the same action. More sophisticated agents can be created by adding some state (memory) to the system, so that the agent is able to recognise and react to particular *sequences* of percepts, rather than individuals. Figure 3.2 illustrates how this might be achieved: we split the function into two parts, sense and act. The first is responsible for updating the stored world model (memory) to incorporate new information from the percept, and the second then uses this updated model to decide upon a suitable action. Note that depending on the specific agent architecture, these two functions need not operate in lock-step. For instance, the sense function might be called multiple times to update the world model before the agent decides it is time to select a new action to perform.

Assuming events arrive sequentially, then the possible sequences of percepts can be characterised similarly to language strings conforming to some grammar. It follows that an agent program equivalent to a finite state machine (FSM) would be able to recognise and respond to sequences of events corresponding to a *regular* language[55]. By adding a stack datastructure for remembering previous percepts (i.e., transforming the agent into a pushdown automaton), then the agent would be capable of responding to *context-free* sequences of events (for instance, closing the gates to a safari park only when everyone that entered has left again), and so on through the Chomsky hierarchy until we get to agents that are capable of responding to any recursively-enumerable sequence (i.e., pretty much anything). Such an agent is clearly a very sophisticated and powerful computational machine. However, the agent is still reactive in that it must wait for some external event to occur or for a percept to arrive before it begins reasoning about what to do. It is worth noting though, that even quite simple reactive agents can exhibit surprisingly sophisticated behaviour when interacting with a complex environment.

Reactive agents can be implemented in a variety of ways. Typical approaches include finite state machines, production rule systems, or trained connectionist representations such as various artificial neural network designs (which essentially learn a finite

state machine). The degree of sophistication of the world model retained in memory, and whether it is an explicit symbolic representation, or some more implicit representation, varies with different agent architectures. Some researchers have argued[19, 18] against the use of explicit world models at all, in favour of simple state-machine behaviour and tight feedback loops with the environment. While this approach has achieved some success, particularly in robotics, most agent architectures maintain some state (explicit or otherwise), and it seems hard to avoid when developing agents with more sophisticated capabilities. The trade-off between maintaining a world model and yet keeping track of the latest changes in a dynamic environment is a difficult balance that an agent architecture must handle. Examples can be found across a range of different positions on this issue, and we will examine some later in this chapter.

A number of reactive agent architectures have been proposed, for both software agents and for physical robots. Perhaps the most famous such architecture is the Subsumption architecture of Rodney Brooks[17]. This robot control architecture avoids any explicit world model representation in favour of tight control feedback between the agent and the environment [19, 18]. The subsumption architecture is organised as a collection of layers of functionality. Higher-level behaviours (such as path following) are built on top of lower-level layers, such as obstacle detection and avoidance. Each layer is an independent process consisting of a finite state machine augmented with some timers. Higher layers can modulate the behaviour of lower layers by overriding either the inputs or outputs of that layer. Brooks' robots demonstrated some success at navigating complex environments, especially when compared to other robots of the time that tended to not adapt to changing environmental circumstances very well. However, the lack of internal representations together with the increasing complexity of connections between layers as more behaviours are added, leads to problems as the architecture is applied to more and more complex tasks. Moreover, the central thesis of Brooks' approach, that explicit world models should be avoided, has been criticised as being particular to the domain of physical robots navigating a complex environment, and is less applicable to real-world software domains[40].

$$\begin{aligned} \mathsf{sense} &: \mathsf{Percept} \times \mathsf{WorldModel} \rightarrow \mathsf{WorldModel} \\ \mathsf{propose} &: \mathsf{WorldModel} \rightarrow \mathsf{Action}* \\ \mathsf{select} &: \mathsf{Action}* \times \mathsf{Goal} \rightarrow \mathsf{Action} \end{aligned}$$

FIGURE 3.3: A deliberative agent.

3.2.2 Deliberative Agents

Beyond simply reacting to events in the environment, we can consider agents that also consider the future state of the environment. At the most basic level, we can consider *deliberative* agents that reason about which of a selection of candidate actions will produce the most beneficial future state of the environment with regards to the agent's goals. Figure 3.3 shows how such an agent might be designed. This new agent is similar to the reactive agent with memory in Figure 3.2, but now the act function has been split into two: an initial propose function examines the environment model in memory and proposes a *set* of possible actions to take. The select function then evaluates these alternatives to pick the most promising single action to execute, based on the current goal (or goals) of the agent. How the agent evaluates potential courses of action again depends on the particular agent program and architecture. For instance, a chess-playing agent might evaluate actions based on a limited search of the state-space of possible future board configurations. More generally, an agent might employ some numerical measure of expected *utility* to evaluate different potential actions, such as in Decision Theory[102] (or Game Theory when considering multiple agents).

One of the most influential deliberative agent architectures is the belief-desire-intention (BDI) architecture that builds on the theory of BDI agents described in Section 3.1.1 (page 53). This architecture was implemented in a number of concrete agent systems, such as the Procedural Reasoning System (PRS), IRMA and dMARS [60, 87]. Figure 3.4 shows the classic system diagram for the PRS system, in which the goals (desires), beliefs, plans, and active intentions of the agent are implemented as individual data structures and a central reasoning service (interpreter) acts on these data structures to advance the state of the agent. The basic interpreter cycle of such an agent is straight-forward: new percepts are added to the belief base structure, triggering events. The reasoner then selects relevant

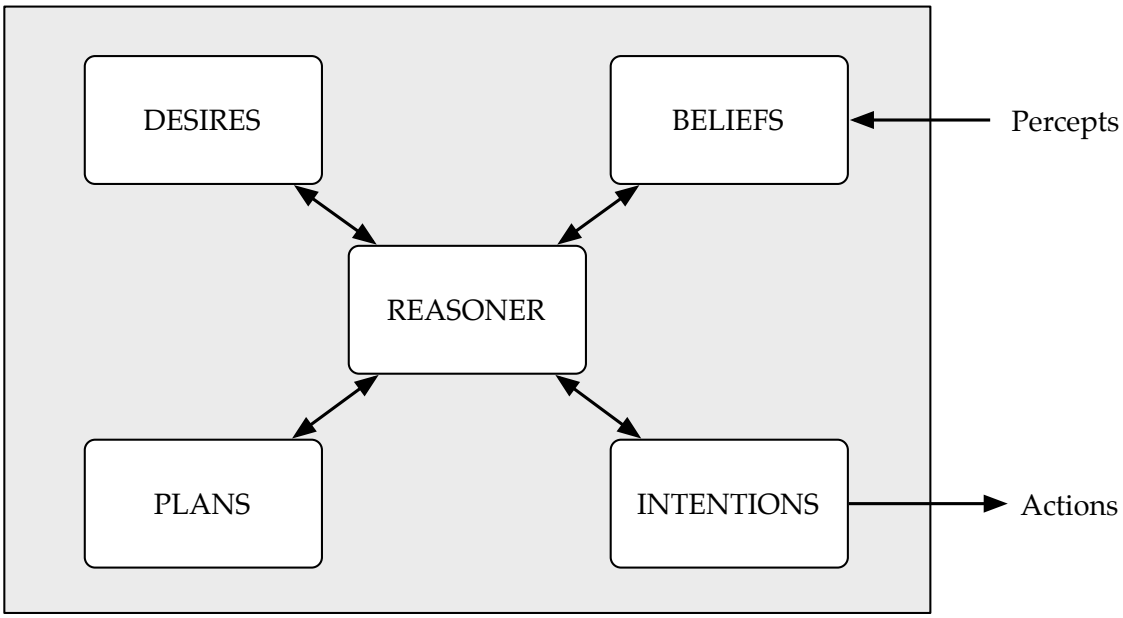

FIGURE 3.4: The Belief-Desire-Intention (BDI) Architecture.

plans based on the current beliefs and desires (goals) of the agent from a plan library. From the set of selected plans, the agent then picks which plan (or plans) to commit to and these then become active intentions. On each cycle each active intention can then propose actions or make revisions to the beliefs or goals of the agent.

The BDI architecture forms the basis of a number of agent programming languages and toolkits, such as JADEX, JACK, Jason, and 2APL. Both Jason and 2APL are based in part on the AgentSpeak(L) programming language described by Rao [95]. AgentSpeak is a PROLOG-like logic programming language, extended with plans and events. The operational semantics of the language are based on the reasoning cycle of PRS, while providing a formal connection to the theory of BDI logics.

3.2.3 Proactive Agents

A *proactive* agent is a type of deliberative agent that actively plans to achieve some future state, rather than simply picking the best action that currently seems applicable. Figure 3.5 presents a sketch of such an agent. As before, the sense function still updates the world model in response to new percepts, but the agent no longer chooses an action directly based on this world model. Instead, the agent first chooses a *plan* (i.e., a sequence of steps

$$\begin{aligned} \mathsf{sense} &: \mathsf{Percept} \times \mathsf{WorldModel} \rightarrow \mathsf{WorldModel} \\ \mathsf{plan} &: \mathsf{WorldModel} \times \mathsf{Goal} \rightarrow \mathsf{Plan} \\ \mathsf{execute} &: \mathsf{Plan} \rightarrow \mathsf{Action} \end{aligned}$$

FIGURE 3.5: A simple proactive agent that plans ahead.

or sub-goals) to achieve its goal, and then executes this plan to decide on particular concrete actions to perform. This sequence is known as the *sense-plan-act* cycle, although in reality these functions are usually decoupled. For example, the execute function will typically be called multiple times for the same plan to achieve each step in turn, and the plan function will be called less often (as it is typically expensive). Planning itself can take many forms, and can be done either dynamically or ahead-of-time. Many agent architectures do not perform much explicit planning once an agent is running, but instead make use of a preprepared *plan library* that contains prototype plans for use in various situations (perhaps parameterised over specific details). More sophisticated agents might have a dedicated planning component, such as a limited theorem-prover for a suitable representation language (such as the situation calculus of section 2.3.1), although this is often confined to a particular domain, such as route planning.

In contrast to the simple picture presented in Figure 3.5, a real agent may have multiple simultaneous goals, and multiple active plans it is working on. The architecture of such an agent will typically be more sophisticated. In particular, executing a plan must propose actions, and then a further action selection function must decide which action(s) from which plan(s) can be allowed to execute at any particular moment in time (i.e., we need to reintroduce the select function of Figure 3.3, or some other form of *conflict resolution*). As well as deciding between particular actions to perform, an agent may also want to deliberate over which goals to actually pursue (*intention selection*) or which set of plans to currently execute.

3.2.4 Reactive Planning

One problem with planning ahead is that environments tend to change over time, sometimes invalidating assumptions that were used during initial formation of a plan. If an

$$\mathsf{execute} : \mathsf{Plan} \times \mathsf{WorldModel} \rightarrow \mathsf{Action}$$

or

$$\mathsf{repair} : \mathsf{Plan} \times \mathsf{WorldModel} \rightarrow \mathsf{Plan}$$

FIGURE 3.6: Two approaches to reactive planning.

agent blindly follows a plan without taking into account these changes to the environment, then it may end up wasting resources or even acting against its own interests. On the other hand, planning is typically expensive and so the agent should avoid replanning as much as possible. One solution to this problem, of course, would be to avoid planning at all, or to limit the size of plans so that changes are unlikely in the time it takes to execute the plan. However, this is not always possible or desireable. Alternative plan representations and plan execution strategies have been developed which attempt to incorporate changes in the environment into the ongoing execution of a plan; taking advantage of new opportunities, or limiting the damage of unfortunate changes. Such techniques are known as *reactive planning* (or *reactive execution*). Figure 3.6 shows two possible approaches to reactive planning. The first adapts the execute function to also take into account the current world model during execution of a plan. This is the approach taken by plan representations such as *Teleo-Reactive Programs*[91] (TRPs) or Reactive Action Packages[43] (RAPs). In these approaches, the plan representation incorporates conditions to check on the state of the environment and to adapt the behaviour appropriately. For instance, in a TRP, each step of the plan is guarded by a condition which must be true for this step to be performed. By checking each guard condition in turn, the plan is able to both skip steps that happen to already have been performed, and to retry steps that have failed. Another alternative is to aim to detect when the assumptions of a plan have been invalidated and then to attempt to *repair* the plan. The idea here is that instead of formulating an entirely new plan (which is expensive), the agent instead attempts to minimally alter its current plan to take into account the new conditions of the environment, as in 2APL [33]. This is the approach sketched in the repair function of Figure 3.6. Of course, an agent can incorporate a variety of different mechanisms, including pre-canned plan libraries in some reactive plan representation, as well as dynamic planning and replanning, as required.

3.2.5 Reflective Agents

Beyond deliberative and proactive agents, there are further agent capabilities that we can term *reflective* or *introspective*. Such agents are capable of reasoning and deliberating not just about the current, past and future states of the environment, but also about their own internal representations and reasoning processes. Such self-reasoning capabilities can be useful to avoid getting stuck in loops, and for learning and other tasks. These are what Aaron Sloman refers to as *meta-management* capabilities[107]. Simple meta-management capabilities are already present in the goal arbitration and action selection mechanisms of some deliberative agents (i.e., the agent is deliberating in a limited way about its own internal representations). Various programming languages and environments also support reflective or introspective capabilities (e.g., [20, 54]). Reflective capabilities can provide different levels of abstraction to the agent when it is introspecting on its own reasoning processes: for instance, at the implementation level, allowing an agent access to the data structures of its interpreter; or at a more abstract level, allowing the agent to adopt the intentional stance towards its own activities, reasoning about itself in terms of its own beliefs, goals, plans, etc. Agent architectures with some degree of reflective capabilities include the Procedural Reasoning System (PRS) [60] and Soar [89].

3.3 Multi-Agent Systems

In order to apply an agent-based approach to generating narrative from events in large-scale persistent worlds it is necessary to employ a multi-agent system (MAS), as it is in general impossible for a single embodied agent to provide adequate coverage of events occurring in such a world. A multi-agent system is simply an environment in which multiple agents are active simultaneously. Agents within a MAS may either cooperate with each other or compete for resources. In this section we will consider only mechanisms for coordinated agent action, assuming that we are designing the MAS as a whole and that the agents can therefore be assumed to be cooperative (we ignore the issue of non-cooperative human participants at present). A multi-agent approach will only be effective, however, if those agents coordinate their actions in some manner. Otherwise, in the worst case, we could have a situation in which all agents attempt to cover the same small area of the environment. At best, this would be no better than the single agent approach, and at worst the agents could hinder each others ability to accurately report on the environ-

ment (e.g., by getting in each other's way). Coordination can be achieved either on an individual basis (agents have individual behaviours to avoid inhibiting each other, as in flocking behaviours), or on a collective level, whereby agents explicit *cooperate* with each other to achieve common goals. Castelfranchi [22] differentiates between *social* action, in which agents individually reason about the goals and intentions of other agents (and either adapt their own behaviour or try to influence another agent's behaviour), and *collective* action in which agents explicitly adopt collective goals and intentions. Collective action is built on social action.

Cooperation may itself take several forms. In *task sharing* a problem is solved in a distributed fashion by assigning particular tasks to individual agents. Each agent carries out the tasks it is responsible for, relying on other agents to also achieve their tasks, so that a common solution can be achieved. In *result sharing*, agents work individually, but may share relevant information with other agents for mutual benefit [120]. In this section we review the simple Contract-Net protocol that forms the basis of many implemented multi-agent cooperation strategies, before looking at implemented multi-agent system architectures in Section 3.4 (page 65). Other, more detailed, theories of agent cooperation have been described in the literature, such as the *joint-intentions* theory of Cohen and Levesque [29, 73, 30], the related joint responsibility model of Jennings and Wooldridge [61, 62, 121], and the SharedPlans theory of Grosz and Kraus [50, 51]. Such theories are not described further in this thesis as their relative sophistication was not required for the task.

3.3.1 Contract Net

One of the earliest mechanisms for multi-agent coordination is the Contract Net protocol [109, 34], which has since been standardised by FIPA [42]. The protocol allows tasks to be shared between nodes of a network of problem solvers based on the metaphor of human legal contracting. There are four stages involved in the Contract Net (C-Net) protocol:

1. Task announcement
2. Bidding
3. Awarding
4. Expediting

In the first stage, a node[2] (the *manager* agent) recognises a problem which it is unable to solve locally, and broadcasts a request for assistance. This broadcast can either be a *general broadcast* in the case where the manager has no knowledge of which other agents will be able to help, or a *limited broadcast* where the manager sends the announcement to just those other agents which are known to be able to solve the problem, or, in the case where the manager has a single preferred agent to solve the problem a direct *point-to-point* announcement can be made to just that node.

Once an announcement has been made, those agents that have received the announcement can examine the eligibility requirements contained in the announcement to determine if they are capable of fulfilling the contract. If so, they can send a *bid* message back to the manager offering assistance. This forms the second stage of contract negotiation. After a specified deadline for bidding has expired, the manager determines which of the bids is the most appropriate and sends an *award* message to that node establishing a manager-contractor relationship between the two agents. Finally, during the *expediting* phase, the contractor carries out the work necessary to fulfill the contract (which may involve further sub-contracting) and sends the manager a *report* message upon completion.

Alternatively, there is a lighter-weight protocol for cases where the manager prefers a particular node for performing the task. In this case the manager can directly award the contract to that node rather than going through the process of announcing the contract and accepting bids. In this direct awarding process the recipient of the award must acknowledge the award, but can also refuse the contract. In addition, there are simple *request* and *information* messages that can be used for querying other agents and exchanging information without the overhead of a contract.

The Contract Net is a relatively simple protocol in which there is always a single agent in control of the interaction (the manager). For instance, there is no mechanism for resolving inconsistent beliefs between agents and no mechanism for coordinating activities beyond the simple manager-contractor hierarchical relationship. In addition, CNET is a one-shot, short-term mechanism and does not have features to explicitly support longer term coordination between agents.

[2]I will use the terms 'node' and 'agent' interchangeably in the following discussion.

3.4 Multi-Agent System Architectures

In this section we review a number of concrete agent architectures that have been developed specifically for multi-agent cooperative systems.

3.4.1 STAPLE and the Adaptive Agent Architecture

More recent work on the joint intentions theory has led to the development of the STAPLE language [69, 70] for agent development that incorporates communication and teamworking capabilities. STAPLE has a logical semantics based on the joint intentions theory and incorporating further theories of communication based on joint intentions. Based on Prolog, the interpreter consists of six main components:

> *"(1) a modal reasoner, (2) a knowledge base maintenance system, (3) a rule interpreter, (4) a plan interpreter, (5) an observer and actuator interpreter, and (6) the main interpreter. Observers and actuators are high-level abstractions for sensors and effectors."*
>
> ([70], pp. 567).

The Adaptive Agent Architecture (AAA) has also been developed by the team [68, 71] as a fault-tolerant brokered architecture based on teamworking. This architecture extends the joint intentions theory with the ability for team members to dynamically join and leave teams over time. It also introduces a theory of 'restorative maintenance goals' that can be used to start new brokers and add them to teams in order to maintain a certain number of brokers at any time, despite failures. Conventions ensure that new team members are kept in contact with even after temporary disconnections, and that the team ensures that a certain number of agents (specified at team formation) is present in the team at all times. The concept of Joint Persistent Goals (JPGs) is extended to that of Team Persistent Goal (TPG) which is formulated in terms of a team (as an independent entity) rather than in terms of the individual members of the team. This allows for a persistent team whose identity is independent of the particular members, which in turn means that the team commitments won't be dropped if any of the team members leave (as would be the case with JPGs). In order for this to work, each member of the team must now have individual beliefs about membership of teams to ensure that mutual belief can be correctly established.

3.4.2 ARCHON and GRATE*

The Joint Responsibility model formed the basis of the GRATE* architecture, an extension of the GRATE architecture (Generic Rules and Agent model Testbed Environment), both of which were developed as prototypes of the ARCHON industrial control system project [62]. The architecture is a Belief-Desire-Intention architecture extended with joint intentions [61]. Commitments and conventions are represented as rules in a rule based system. There are two main components to the architecture: a *domain-level system* that contains domain-specific functionality; and a *cooperation and control layer* that operates on the domain layer and ensures that local actions are coordinated with other agents. The domain layer consists of a set of atomic (from the view of the control layer) 'tasks' representing particular capabilities of the agent. The cooperation and control layer is itself made up of 3 primary modules, each of which are forward-chaining production systems with their own local working memory:

- A *Control Module* that interfaces with the domain layer;
- A *Situation Assessment Module* that is in overall control of the agent and decides which activities should be performed locally and which require social cooperation;
- A *Cooperation Module* that manages social activities as requested by the Situation Assessment Module. These activities include establishing new social interactions (e.g. finding an agent capable of performing some task), tracking ongoing cooperative activity, and responding to requests for cooperation from other agents.

In addition, there are some utility components: A general purpose *Information Store* that stores useful information from the domain layer and also information about other agents; *Acquaintaince* and *Self* models that store information about state, capabilities, intentions, etc of agents; and a *Communication Manager* that handles the details of sending and receiving messages to/from other agents.

Any agent can initiate group activity. When an agent notices an opportunity for coordinated activity it can initiate such activity by contacting other agents it would like to participate in the activity. The initiating agent then becomes the organiser/leader of the group and is responsible for contacting other potential participants, deciding the common solution strategy, and assigning actions and times to team members. Agents are assumed to be capable of executing only a single action at a time, and so consistency of intentions in

GRATE* is based solely on the times at which actions have been agreed to be performed—an agent cannot commit to executing two actions at overlapping times.

In addition to local-only (i.e. a single agent doing all tasks) and full collaborative action, an agent may also opt for a cheaper middle way alternative. In this approach the agent performs most activity locally and only makes requests for short-term assistance as and when it is required. This alternative is available to avoid the overhead of full cooperative planning when only a small amount of external cooperation is required: for instance some information sharing, or performing one or two simple actions.

3.4.3 TEAMCORE and STEAM

The TEAMCORE project [114, 112] is another effort to add flexible teamworking capabilities to agents, based on the joint intentions theory, encouraging a *'Team-Oriented Programming'* style [94]. The architecture builds on top of and integrates the previous STEAM (Shell for TEAMwork) architecture [113], which combines elements of the joint intentions theory and the SharedPlans model to create a domain-independent knowledge base used by agents to reason about teamworking. This knowledge base was captured in 300 Soar [89] rules, divided into 3 categories:

- Coherence preserving rules ensure that agents communicate to establish mutual belief of relevant conditions (such as a plan becoming unachievable);
- Monitor and repair rules specify how team members can be replaced if they fail to achieve their tasks;
- Decision-theoretic selectivity-in-communication rules are used to evaluate the utility of communication to avoid too much expensive communication.

Teams in STEAM can be either flat or hierarchical in organisation, and divided up into *roles* which can be assigned to individuals or to sub-teams. Roles may be either persistent or assigned on a short-term basis (task-specific roles) and can be either pre-assigned or dynamically assigned (e.g. as a result of negotiation). Furthermore, mechanisms for reallocating roles exist within the architecture ([88] describes a method for evaluating role allocation and reallocation strategies).

Team activities are represented as a hierarchy of *team operators* (team reactive plans). These are similar to reactive plan hierarchies in architectures for individual agents,

such as RAP, PRS, or Soar. Each operator in the hierarchy specifies a particular task that is to be accomplished, and these tasks are broken down into sub-tasks. When a particular operator is activated then (in Soar) a new *problem space* is created in order to reason about which sub-tasks should be performed, and how. In STEAM, these hierarchies can contain team operators as well as individual operators. Team operators are similar to individual operators and have preconditions, application rules, and termination rules, but team operators operate on a separate *team state* memory that stores the mutual beliefs of the team, rather than the agent's own private belief state. Team state is not shared between agents directly—each agent has its own copy of the team state for each team it is involved in—but it is kept synchronised between different members of the team. Team organisation is separate from task decomposition, and the mapping between the two is accomplished by roles. Roles constrain which sub-tasks of the current team operator that a particular agent can take on—an agent can only take on those tasks which agree with its current role within the team. Team organisation is expected to be *persistent*. A variation, STEAM-L (the L is for 'lookahead'), has been developed which uses decision-theoretic techniques to reason about the long-term expected utility of team actions and thus avoid teams making locally optimal, but long-term suboptimal, decisions about resources [115].

TEAMCORE builds on STEAM and adds a number of new components and capabilities. Firstly, the domain-independent teamworking knowledge embodied in STEAM has been separated out into 'wrapper agents'. In the original STEAM architecture, this team-working knowledge was integrated directly into the agents' other domain-specific capabilities. In TEAMCORE, existing agents are wrapped with a TEAMCORE agent that handles all communication and coordination on behalf of the existing agent. The existing agent must be adapted to communicate with the TEAMCORE agent, but this is presumably a much simpler task than the previous integration. Communication in TEAMCORE is via the standard KQML agent communication language, and there is a built-in argumentation and negotiation module, CONSA (COllaborative Negotiation System based on Argumentation) which is used to resolve conflicts in beliefs and plans within the team.

An important component of the TEAMCORE architecture is the role-based team learning component, ROBOL (ROle Based Organisational Learning). Each agent within the team maintains a Partial Organisational Structure (POS), which is an acyclic graph structure representing the relationships (edges) between roles (vertices). Each vertex is labelled with the agent(s) assigned to that role, as known to the agent maintaining the

POS. Each agent uses its own POS to reason about the team while performing its own tasks. Based on feedback from interactions with the rest of the team, each agent modifies its POS to better reflect the actual current composition of the team and assignments of roles. Thus rather than having a static team structure with fixed role assignments, each team member maintains a dynamic view of the current setup of the team. This flexible approach allows for changes to be made to the team structure dynamically without having to go through a complicated process of renegotiation of roles. Instead, each team member evolves their POS to better reflect the current team structure. A global measure of team performance can also be used to learn an assignment of roles that performs best on the current task, although the details of how this is accomplished in practice are not fully described.

3.4.4 CAST

The CAST (Collaborative Agents for Simulating Teamwork) framework [122] is designed to support mixed human and artificial agent teams. This focus places extra constraints on the design of the system. In particular the amount of communication that takes place between agents should be minimised so as to prevent flooding human participants with lots of potentially redundant information. To this end CAST supports reasoning about the goals and responsibilities of team members. This reasoning allows CAST agents to only provide information that is actually useful to the recipient. A further advantage of this reasoning is that CAST agents can engage in *proactive information exchange* in which they predict what information different team members will require at different times and provide that information without having to be prompted. This further cuts down the amount of messages that must be exchanged. In order to achieve this, CAST agents maintain models of the shared intentions of the team and mental states of other team members.

Petri Nets are used to store these shared mental models as an approximate method that avoids the computational complexity of other formalisms, such as higher-order modal logics used in belief reasoning. The Petri Nets are generated from team structure and process descriptions written in the MALLET (Multi-Agent Logic-based Language for Encoding Teamwork) language. A Petri Net for an individual team member encodes background knowledge of individual responsibilities (goals, operators) and how their role is integrated with the rest of the team. The knowledge written in MALLET includes information on *roles* (references to specific steps in team plans which individual agents can be

assigned to), *operators* (named atomic actions), *team operators*, and *plans*. Operators are conjunctions of literals plus variables, and also have pre- and post-conditions associated with them. Team operators have a variety of different 'share' types: AND operators require action by all team members; OR operators require action by at least one (any) agent; and XOR operators require action by at most one agent. Plans are described using constructs for sequencing, parallel execution, conditionals, and loops. Team plans extend individual plans with role variables which can be instantiated with particular team members. Role variables can have associated constraints on which sorts of team members can fulfill them (e.g., the team member closest to some entity). In addition, team members can be given broad responsibility for certain operators so that they will be considered whenever that operator is proposed. A dynamic role selection (DRS) algorithm determines which agents can fulfill which roles in a team plan under consideration, and resolves conflicts when there are multiple candidates for an XOR team operator.

These descriptions of agents and team plans are compiled into the individual Petri Nets for each team member. During compilation an information flow analysis stage determines what communication will be required during each stage of each plan so that each role has the information it needs to perform its operation at the correct time. The proactive information exchange algorithm is based on three conditions. Agent A should inform agent B of information I iff:

1. A knows the truth value of I;
2. A believes that B does not know I;
3. B has a goal that depends on knowing I.

The DIARG (Dynamic InterAgent Rule Generator) algorithm identifies opportunities for proactive information exchange. The algorithm is run on each cycle by each agent, and identifies which agents should be asked for information when it is needed, as well as which agents require information that the current agent knows.

CHAPTER 4

COLLABORATIVE NARRATIVE GENERATION

4.1 Introduction

This chapter presents the basic approach taken to generating narrative from persistent virtual environments. The overall design of the system is presented, along with the motivations for the various design decisions. The following chapters then go into further depth on the architecture and implementation of the system.

The overall aim of this research is to provide a framework for automatically generating narrative accounts of the activities of participants in medium to large-scale persistent virtual environments, with a focus on the area of (massively) multi-player online role-playing games (MMORPGs). As discussed in the introduction to this thesis, these games currently offer individual players a significant amount of freedom in how they approach their gameplay, but this freedom usually comes at a cost of reduced involvement in any overarching narrative experience; a hallmark of single-player RPGs. The reason for this lack of personalised narrative content stems from the sheer complexity of the task of producing custom narrative based on each individual's actions, and of incorporating these individual storylines into a coherent narrative structure. This thesis directly addresses the first of these two interrelated problems, by automatically generating narrative based on individual and group actions.

The approach adopted not only attempts to provide automatic narrative generation for large numbers of simultaneous users, but also to allow those users to interact with the system and influence the storylines that are so generated. For this reason, we refer to the approach as *collaborative* narrative generation: authoring of the narrative becomes a collaborative effort between the system itself and the participants in the game environment.

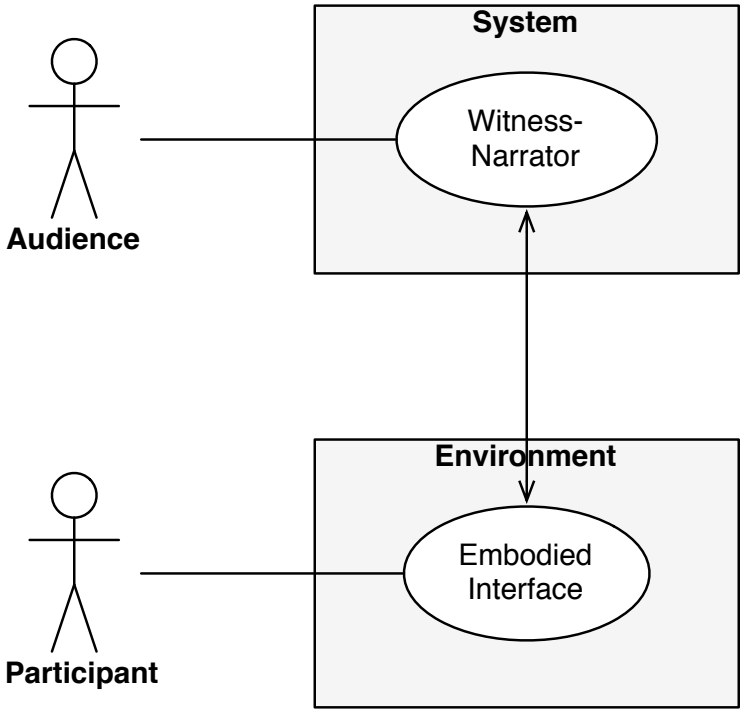

FIGURE 4.1: Primary system use-cases.

The mechanisms by which players can influence the narrative, and the extent to which this collaboration is supported, are discussed in the following sections, before a detailed discussion of the high-level specification of the system as a whole.

Within the scope of this research, we can identify two primary classes of user of the system, shown in figure 4.1:

1. The *participants* in the game world, who clearly have a vested interest in how their exploits are reported;

2. The *audience* of the produced stories, who may wish to indicate a preference for certain styles of story, or certain classes of events.

Clearly, these two groups of users are strongly related, and much overlap can be expected. For example, especially with low-user environments, it can be expected that the audience reading reports about an environment will consist almost entirely of participants within that environment (e.g., wishing to catch up on any events they may have missed). However, the distinction is still valid as, relative to any particular narrative, only a proportion

of those reading the report will have been directly involved in the events it records. Addressing the needs of each user group therefore requires mechanisms for each to feed into the narrative authoring process.

Another key design requirement of the system is that the output be presentable via a variety of different output and communication media, such as HTML web pages, mobile telephone SMS messages, syndication formats such as RSS or Atom [1], and directly in-game via a suitable interface. The design is therefore factored so as to separate the generation and organisation of narrative from the specific formatting for a particular output medium. It is also expected that this could facilitate localisation and internationalisation of output stories, but this is not directly addressed in this thesis. The system as implemented concentrates on just a handful of output mediums. In particular it is capable of publishing stories to an online weblog ("blog") via the Atom Publishing Protocol (AtomPub) [1], supported by many popular publishing platforms (which, in turn, typically support a variety of output media), and also in-game via environment-specific means.

4.2 Witness-Narrator Agents

The design of the system is based on a multi-agent approach, whereby the system is organised at a conceptual level as a collection of individual agents that collaborate to achieve the tasks outlined above. In particular, we build on the notion of a 'witness-narrator' agent, initially developed within the scope of the INSCAPE European project [111]. The concept of a witness-narrator agent draws on ideas from literary theory, and in particular the notion of 'narrative voices', which describes the various relationships between a narrator and the world being narrated (e.g., [100]). Witness-narrator agents are embodied within the environment (rather than being omniscient) and both observe what is occurring in the environment as well as narrating these events to audiences inside and outside of the environment. The agents are 'witnesses' rather than protagonists, as they do not actively play a part in the activity of the world beyond their presence and the narration they provide [111, 75].

Embodiment provides an interface to the narrative system which is seamlessly integrated with the virtual environment. Participants can interact directly with the witness-narrator agents in same way as other non-player characters (NPCs). For example a player may approach a witness-narrator agent to request information about current events else-

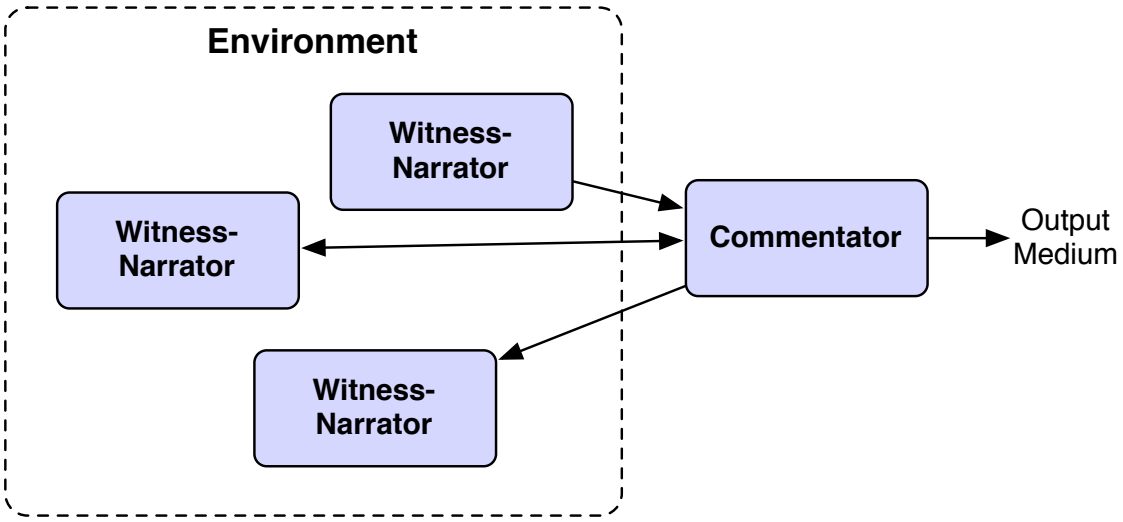

FIGURE 4.2: Overall agent framework, showing embodied witness-narrator agents and non-embodied commentator agents.

where in the environment, or that the agent accompany them as they progress through the game, to share reports of their activities with others. Participants can also interact indirectly with the agents. Being embodied in the environment grants the agents (approximately) the same access to events as a human participant. Participants can therefore determine when they are being observed, and what information an agent is likely to be able to obtain given its position relative to the participant. As a result, players can try to avoid the agents, or can modify their behaviour when around them. For example, participants may wish to keep details of their strategy secret from their opponents in order to preserve an element of surprise. Conversely, players can deliberately try to influence the agents by approaching them, either 'acting up' (e.g., celebrating a victory) or perhaps targeting specific messages at particular individuals in the outside world. We believe that such control over what gets reported is an important part of responsible reporting of events to other participants or an external audience.

The witness-narrator agent framework is organised as a society of agents, with different types of user interacting with different types of agent, see Figure 4.2. We distinguish the two main types of user identified previously: *participants* in the virtual environment, who are the subject of the narrative; and an external *audience* who are not (currently) embodied in the world but read accounts of the action via some other medium, such as a web page, IRC channel or text messages. Participants interact with *witness-narrator agents*

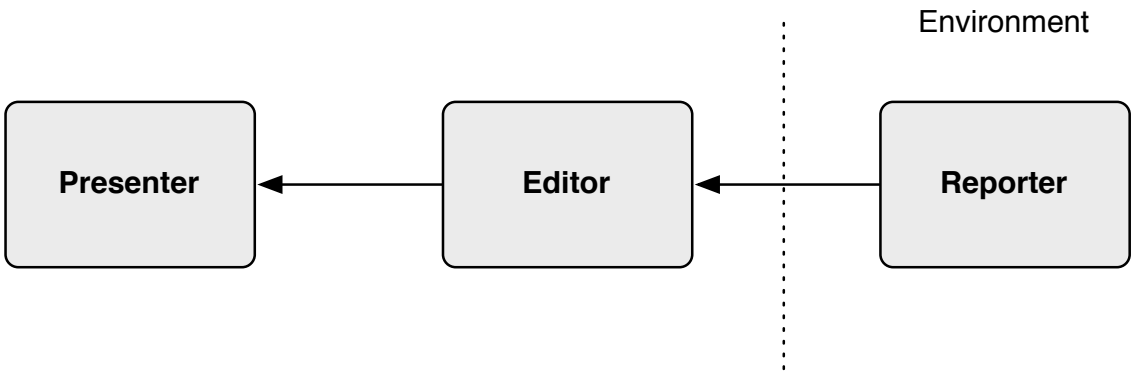

FIGURE 4.3: System workflow overview.

using the standard mechanisms for interacting with NPCs within the game (i.e., menus and text output). Witness-narrator agents are embodied in the environment and observe events in the same way as a human player. Members of the audience interact with *commentator agents* tailored to a specific output channel. Commentator agents are not embodied in the environment and have no first hand knowledge of events in the environment, relying on the witness-narrator agents to provide this information.

The agents function both as an implementation technology, and, more importantly, as an interaction framework which both structures the user's experience of the narrative and allows them to (partially) shape the development of individual narrative strands. The witness-narrator agents' embodiment in the game, and their resulting "first person" view of events explains both the ultimate source of the narrative and makes explicit the limitations on what is knowable about the game. Like human players, their (individual) view of events is limited to the actions of human players, and while they can speculate about the thoughts, feelings or motives, of other players, these remain ultimately opaque.

4.3 Workflow

The basic workflow of the system is shown in figure 4.3 and is conceptually based on Daniel Fielding's previous work on reporting from *Unreal Tournament* [41]. The three basic roles identified in this workflow are as follows:

- **Reporters** are embedded within the environment and are responsible for gathering

information about events as they occur;

- **Editors** are responsible for aggregating reports from multiple reporters, checking them for consistency and completeness, and then combining them into a single coherent narrative;

- **Presenters** are responsible for relating generated narratives to an audience via some output medium, formatting the narrative appropriately, and collecting any feedback to the system.

In Fielding's work, these roles were implemented as individual concrete agents. In the current research we treat these roles more abstractly, and in particular allow a flexible relationship between any particular agent and the roles it is capable of performing.

In operation, the framework is designed to work in the following fashion:

1. Firstly, reporter agents observe the entities in the environment and their actions, classifying them according to some formal ontology;

2. Next, the reporters identify particular observed events that are *interesting*, and collect together all information on such events into formal report structures;

3. These reports are then sent to an editor;

4. The editor combines reports from multiple reporters, checking them for consistency, and producing a combined report;

5. The editor continues to expand the detail of reports as further information arrives from reporters, possibly integrating details into a higher-level event report (e.g., recognising a battle from reports of individual combat actions);

6. A presenter periodically makes a request to an editor asking for any interesting reports. These are then sent to the presenter, which then generates a narrative account of the reported events in a suitable output format.

4.3.1 User Interaction: Focus Goals

Users interact with the system by making requests for information about past, present, or future events and by rating the information produced in response to their request. Re-

quests for information are represented within the system as *focus goals*. A focus goal describes the sorts of events that are currently of interest to agents within the system, and the audiences that they are narrating to. Focus goals thus originate from presenter agents initially, but may also be generated spontaneously by reporting agents in response to events. A focus goal γ consists of five components:

$$\gamma = (\epsilon, \tau, \lambda, \iota, \Sigma)$$

These components are:

- ϵ: A description of the events expected.
- τ: The times at which the goal should be considered active.
- λ: The area or location on which to report.
- ι: An indication of the *importance* of the goal, as an arbitrary integer between 0 and 100. At first these will be assigned by presenter agents when they generate the goal, but they could be based on audience share or some other metric.
- Σ: A set of agents that are interested in matching events. Any reports generated that match this goal should be sent to all agents in the set.

The purpose of a focus goal is to direct the activities of the multi-agent system as a whole in order to ensure that output is tailored to the needs and desires of a particular target audience. Adjusting and generating focus goals is the primary means by which participants and external users can interact with the narrative generation process, and therefore the primary form of collaboration within the system. A focus goal differs from a normal achievement goal in that the process that the agents perform in response to a new focus goal is ongoing, and there is no clear criteria in most cases for determining when a focus goal has been 'achieved'. For instance, if the system has a focus to report on stories involving quests in the region of Etum Castle, then there is no clear set of criteria that could be used determine when this goal has been completed: as long as players keep completing quests in that region then there will always be further stories to narrate. Focus goals are also distinct from what might be termed 'maintenance goals', which usually involve maximising some numeric utility score. While there may be some metric which could be used to measure a degree of success (such as coverage), this is not something that the agents

have much direct control over. The agents rely on participants to achieve the goals and are merely attempting to ensure that they are present to observe these important events. In terms of activity recognition, focus goals resemble the End events described in Kautz's theory of plan recognition, c.f. Section 2.4.1 (page 38), but the set of such events is dynamic rather than fixed in the event ontology.

Initially, the system is pre-configured with a number of basic focus goals tailored to the individual environment. For instance, in the *Neverwinter Nights* environment, the system was initially configured to report on all battles, deaths, and achievements (including quests) that occur anywhere within the environment, and to report on these events at regular (20 minute) intervals.

Focus goals generated in response to user requests may refer to past or current events, or to future events. For example a participant may ask a witness-narrator agent what their friends or competitors are currently doing in the environment, or about notable events which took place at the current location in the past. Alternatively, if a participant is about to engage in actions which they consider may be of general interest (or of which they want a personal record), they can ask a witness-narrator to follow them and observe their actions. Similarly, audience members may request information about past or current events in the environment, or coverage of anticipated future events, such as a battle or the actions of another participant. In addition, witness-narrator agents are able to autonomously generate focus goals in response to specific events in the environment (commentator agents do not autonomously generate focus goals). Autonomously generated focus goals always refer to current or future events and are always specialisations of existing focus goals. All witness-narrator agents have an *a priori* set of high-level focus goals which can be used as a basis for autonomous goal generation in addition to any user-specified focus goals. For example, a witness-narrator agent which is following a participant, may notice a battle taking place nearby. The agent already has an (*a priori*) focus goal indicating that such a battle is of interest to the system and so will generate a more specific focus goal relating to that particular event (e.g., specifying the exact location and expected time duration of the event). Other witness-narrator agents (who are not already engaged) can then be recruited to handle this specific event, which otherwise might be overlooked.

A focus goal generated in response to an audience request, or which cannot be achieved by the witness-narrator agent that generated it (either because it lacks the first

hand knowledge necessary to answer the query or because the task it implies is too large for a single agent) are broadcast to all witness-narrator agents, allowing relevant reports to be forwarded to the originating agent. Broadcast focus goals referring to events occurring in the future over a large area or an extended period, or which are likely to be of interest to a wider audience, may give rise to the formation of a team of agents if this is necessary to provide adequate coverage of the events or dissemination of the resulting narrative (as described in more detail below).

Reports are the system's internal descriptions of events and are not presented directly to users. Rather sets of reports which match a focus goal are rendered into *narrative presentations* in one of a number of formats. If the focus goal specifies events in the past or present, this is done immediately. If future events are specified, the narrative may be produced once, e.g., at the end of a specified time period, or periodically, e.g., a daily update to a weblog. Narrative production takes into account the specific constraints of the output channel, e.g., detail present in a weblog may be omitted from SMS messages.[1] Users may 'rate' the narrative produced in terms of its interestingness. Reports which are rated as uninteresting are forgotten by the agents to avoid exhausting system memory. Conversely the most interesting reports are retained, forming a kind of collective memory or 'user generated backstory' of the environment which can be used to satisfy future user requests for information regarding past events.

4.3.2 Reporting

When reporting on an event, there are a number of basic questions that need to be answered by the report. At the most fundamental level are the usual *wh*-questions:

- ***What*** happened?
- ***Where*** did it happen?
- ***When*** did the event occur?
- ***Who*** was involved?
- ***How*** did it happen? What events *caused* this event to occur?

[1] Narrative presentation by witness narrator agents in the environment is only possible if the agent can interact with the participant, which is why focus goals generated by participants referring to future events are limited to "follow me" type requests.

- ***Why*** did the event happen? What were the motives (if any) behind the event?

Some of these questions are easier to answer than others. We assume that the *where*, *who*, and *when* questions are directly observable from the environment. *What* happened is largely a domain-specific question, as the types of events that can occur will vary to some degree from environment to environment. For this reason, we separate this aspect into a pluggable event *ontology* that can be developed independently of the rest of the framework, and tailored to individual environments, described in chapter 5. Tracking *how* an event occurred is largely a matter of keeping track of the order and causal relationships between events, and is strongly connected with the ontology. A further question to answer is *why* the event took place at all. This involves inferring the motives of the players involved in order to determine the reasons for the action. For instance, a murder of another player might be an act of revenge or jealousy. This *motive recognition* is not addressed in this thesis, but left as future work.

4.3.3 Editing

The reporting capability produces reports of low-level events (for instance, individual actions of participants). The primary responsibility of the editing capability is to collate and edit low-level reports from multiple agents in the environment. In particular, editing involves combining low-level reports into higher level reports of events taking place in a wider area. For instance, multiple reports of combat between individuals in a particular region may indicate a large-scale battle occurring between two or more teams of participants. These high-level event descriptions are encoded into the upper-level event ontology (described in the Chapter 5). Specific heuristic rules are implemented to recognise these higher level events from multiple lower level events. These rules are implemented using generic descriptions from the upper-level ontology, allowing them to be re-used in similar environments. For example, a high-level event rule may be concerned with instances of "combat", whereas a particular environment may have an ontology describing particular weapons and types of combat unique to that environment (for instance, spells or futuristic weaponry). The editors abstract from these details using the subsumption relationship in the ontology, while preserving those details in the reports that are sent to presenters.

To describe *how* an event occurred, the order and causal relationships between events are explicitly recorded. For instance, if one participant attacks another and the vic-

tim subsequently dies, then this causal relationship is recorded by specific rules. These rules are again described using only concepts from the upper-level ontology that are common to many environments.

4.3.4 Presenting

The presenting capability is the primary interface between the witness narrator agent framework and the users. It is responsible both for formatting reports for presentation via some output medium and allowing user to rate the resulting narrative, as well as allowing users to specify which events they are interested in.

Narrative presentation of reports consists of three main stages:

1. *Content determination* decides which events to include in a presentation and which details of those events.

2. *Narrative generation* converts these declarative event descriptions into a prose narrative at an appropriate level of detail.

3. *Output formatting* formats the prose narrative for a particular output medium (such as HTML, an Atom newsfeed, or an IRC message).

At present, each of these stages is performed using simple mechanisms, as the main focus of our work is on the collaborative aspects of narrative generation, rather than on producing a polished final narrative.

The implementation of these individual capabilities is described in depth in Chapter 6.

CHAPTER 5

ONTOLOGY OF ROLE PLAYING GAMES

5.1 Introduction

This chapter describes in detail the formal ontology of events and existents (to use the terminology of Chatman, see section 2.1.2) that has been developed to describe the activities of characters in the persistent virtual environments that are the focus of this research work. The scope of possible virtual worlds is obviously very large, and it would be extremely difficult to capture the structure of every possible such environment. For that reason, we concentrate on the specific domain of computer role-playing games (RPGs), which includes single-player games such as *Neverwinter Nights* and also so-called 'massively multiplayer' online role-playing games (MMORPGs) that support many thousands of simultaneous players and environments consisting of hundreds of locations, each of which may be quite substantial. Such environments can still vary widely in their specific design, but most share significant similarities, ultimately sharing a lineage with table-top role-playing games such as *Dungeons and Dragons*. The aim of the ontology is therefore to try and capture the underlying structure of these environments in a way that builds on their similarities (to enable construction of generic techniques) while allowing incorporation of specifics from any particular game world. This is achieved by developing a domain model (the ontology) in a formal language, in terms of the general classes of things that exist within these environments, and providing mechanisms for extending and refining the model for particular games. In this chapter we first discuss how the general ontology was developed, and describe in detail the various elements of the model, before showing how it was applied to the particular environment of *Neverwinter Nights*.

The language used for representing the concepts of the domain and the relations

between them is the $\mathcal{SROIQ}(D)$ description logic language that forms the basis of the OWL-DL subset of the Web Ontology Language (OWL-DL 2) [57, 56][1]. The use of OWL provides a flexible knowledge representation language, which is sufficient to represent much of the structure of our domain of interest, while also providing a number of mature tools that are useful in the construction of such a moderately large ontology. The availability of tools such as the Protégé OWL editor[2] and various description logic reasoners, and growing acceptance of OWL as a standard knowledge representation format, make it easier for the ontology to be extended and adapted for new environments and domains as required. The $\mathcal{SROIQ}$ language underlying the latest versions of OWL DL supports a number of quite expressive constructs, such as role hierarchies, qualified number restrictions, inverse, transitive, reflexive, and symmetric roles (among others), complex property chains, and 'self properties' which allow a role to relate an individual to itself (i.e., $\exists\rho.\mathsf{Self}$ is equivalent to $\rho(x, x)$ in FOL).

A number of formal theories of various facets of knowledge, described in the previous two chapters, are incorporated into the ontology. These include temporal descriptions of events and actions in the world, as well as notions of beliefs, goals, objectives and intentions. We also incorporate some notions of ability of agents to perform actions, and develop simple ways of encoding social norms and laws that allow us to reason about actions and events involving groups of agents. The resulting language incorporates a large number of different modalities. We encode these different notions as DL concepts and roles, which can be interpreted as either reified abstract individuals in first-order logic, or as modal parameters as described in section 2.2.4. Either interpretation is sufficient for our purposes, and avoids an agent concluding erroneous statements, such as the fact that John believes ψ based on the knowledge that John believes ϕ and that $\phi \Rightarrow \psi$ (which we do not know if John knows). The approach taken has been to encode these theories, as far as possible, directly in the description logic language. In some cases (most notably in the treatment of time) this has not been completely possible, due to the expressive limitations of the language. In these cases the semantics of the language have had to be expanded slightly, and new inference rules introduced to correctly handle reasoning about the domain. We describe these extensions to the language in the text as they are encountered. In

[1] In fact, the vast majority of the ontology falls within the $\mathcal{SHOIN}(D)$ language that forms the basis of OWL-DL 1.

[2] `http://protege.stanford.edu`

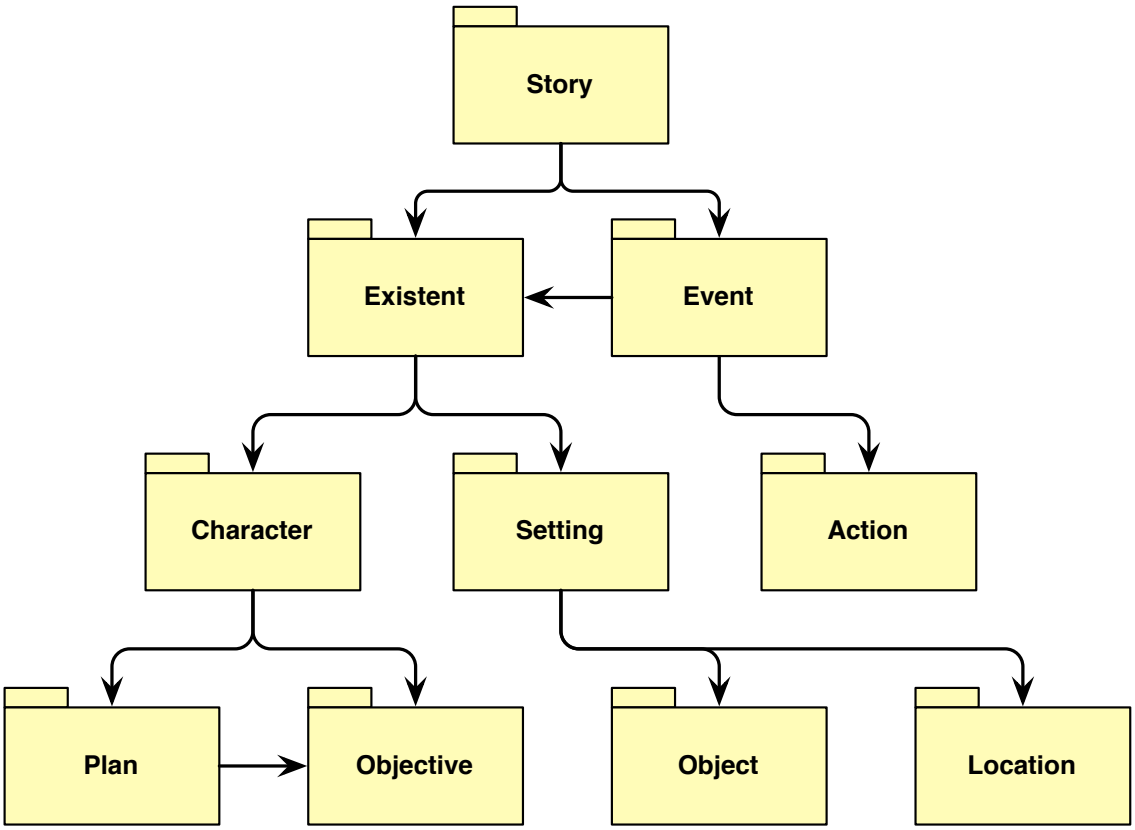

FIGURE 5.1: Ontology modules and dependencies between them.

the implemented system new inference rules are achieved by embedding the description logic ontology as a component within a more expressive first-order rule language (based on PROLOG), creating a hybrid reasoning system as described in section 2.2.5. A key aim has been to limit the effect of such extensions. For instance, our treatment of time does not require introducing extra arguments to any concept or role predicates in the language. In this respect, the extra rules constitute a *conservative* extension of $\mathcal{SROIQ}(D)$: any OWL-DL ontology is a valid ontology in the extended language with identical semantics. Only ontologies that explicitly make use of the extended features have additional semantics.

The ontology is organised in a modular fashion, with axioms and class and property definitions related to a particular area separated out in to individual ontologies, which are then combined and integrated into a coherent whole. The modules that make up the ontology are shown in figure 5.1 along with the dependencies that exist between them. Each module is designed to be self-contained and so can be used independently of the

other modules. Where there are cross-dependencies these have been factored out and moved into higher-level modules. For instance, while a Region is a part of the Setting of a story world, the Location module makes no reference to this fact, but merely describes what sub-classes of regions and other locations there are, and what properties they have. It is the Setting ontology that integrates both the Object and Location ontologies, and records that regions are part of the setting. Likewise, property and class definitions that reference classes and properties from other modules are arranged so that these references are only made in higher-level ontologies. There is no foundational ontology that is used by all modules, although it is expected that the ontology could be adapted to make use of an existing foundational ontology, such as SUMO [90], NASA's SWEET[3], or DOLCE [78], without too much trouble. The methodology used in designing the ontology has been influenced in part by the OntoClean methodology [52, 53], and particular attention has been paid to the correct use of subsumption and instantiation relationships. The main ontology consists of over 200 classes, together with around 70 roles for describing relations between individuals. In total there are slightly less than 500 axioms in the system (including class axioms, object roles, and data-type roles). This chapter describes the main classes and roles only, and leaves out less important distinctions. The concepts that have been omitted are merely specific refinements of more general concepts and are not essential to understanding the structure of the ontology. For example, while we describe the general categories of *props* that can exist, we do not enumerate all the particular classes (such as chairs, different types of tools, etc.). The full axiomatisation of the ontology is given in appendix A.

5.2 Upper Level Ontology

The highest level of the ontology is divided broadly along the lines suggested by Chatman (see section 2.1.2), with an initial separation into *existents* and *events*, with existents subdivided into *actors* (characters) and *setting*, and events divided into *actions* and *happenings*. We deviate slightly from Chatman in separating the abstract description of an action (what the action does) from the actual performing of an action, which we label an *act*. An act is an event, but the action performed is not. This allows us some flexibility in referring to actions independently of a particular performance of that action. For instance, we can describe some actions as being *forbidden* in certain areas (making use of notions from deontic logics),

[3] `http://sweet.jpl.nasa.gov/ontology/`

leading to a conception of a crime as a type of event (a performance of an illegal action), or we can talk of an agent attempting an action, or delegating an action, and so on. We also introduce a number of other top-level categories that seem to fit into neither existents nor events cleanly, such as objectives, plans, and so on. Furthermore, our use of terms differs somewhat from Chatman. For example, in Chatman's ontology, a person in a story may fall into either *setting* or *character* depending on their relation to the story (whether they are a central character or not). We make a simpler distinction that passive elements of the world form the setting, whereas characters (actors) are active (action performing) elements. This cruder distinction is simpler to work with. We later introduce a notion of a *character* as an actor (really, individual) who participates in some story. We make no distinction as to whether the character plays a pivotal role in the story or not. An *act* is an event in which some actor in the world performs an action, whereas a *happening* is an event in which there is no particular responsible actor (for instance, an explosion whose cause is unknown). We group most higher-level events as happenings, and reserve acts for just simple direct actions performed by particular actors. An overview of the highest level of the ontology is shown in Figure 5.2[4].

Note that in this chapter we are concerned only with the elements of the virtual world we are narrating, i.e., the 'story' in Chatman's terms, and in particular with the form of that content. The substance of that content is given by the semantics of the ontology, which is defined by the model-theoretic semantics of OWL 2[5] [49] where the domain is the set of all characters, objects, and so on that inhabit the virtual world[6]. The form and substance of the 'discourse' (Chatman's term) will be discussed in the next chapter in the discussion of how the narrative output of the system is generated.

In the remainder of this chapter, we look in detail at all of the elements of the ontology and how they fit together. Aspects of reasoning related to the ontology are described as necessary to motivate the various choices that have been made, but the main description of the event recognition and other reasoning is more completely described in later chapters. While the main ontology is designed to be as widely applicable as possi-

[4] Thing is the universal concept in OWL, i.e., $\top$.

[5] Note that at the time of writing OWL 2 is still a Working Draft and so subject to change. The description of OWL 2 features in this thesis is based on the 11th April 2008 draft.

[6] We leave unspecified what it means to 'inhabit' a virtual world. Concretely, we might equate objects in our semantics with the data structures used to describe entities in the particular server implementation running the VW. More abstractly, we might consider a possible worlds semantics in which each world is a DL interpretation.

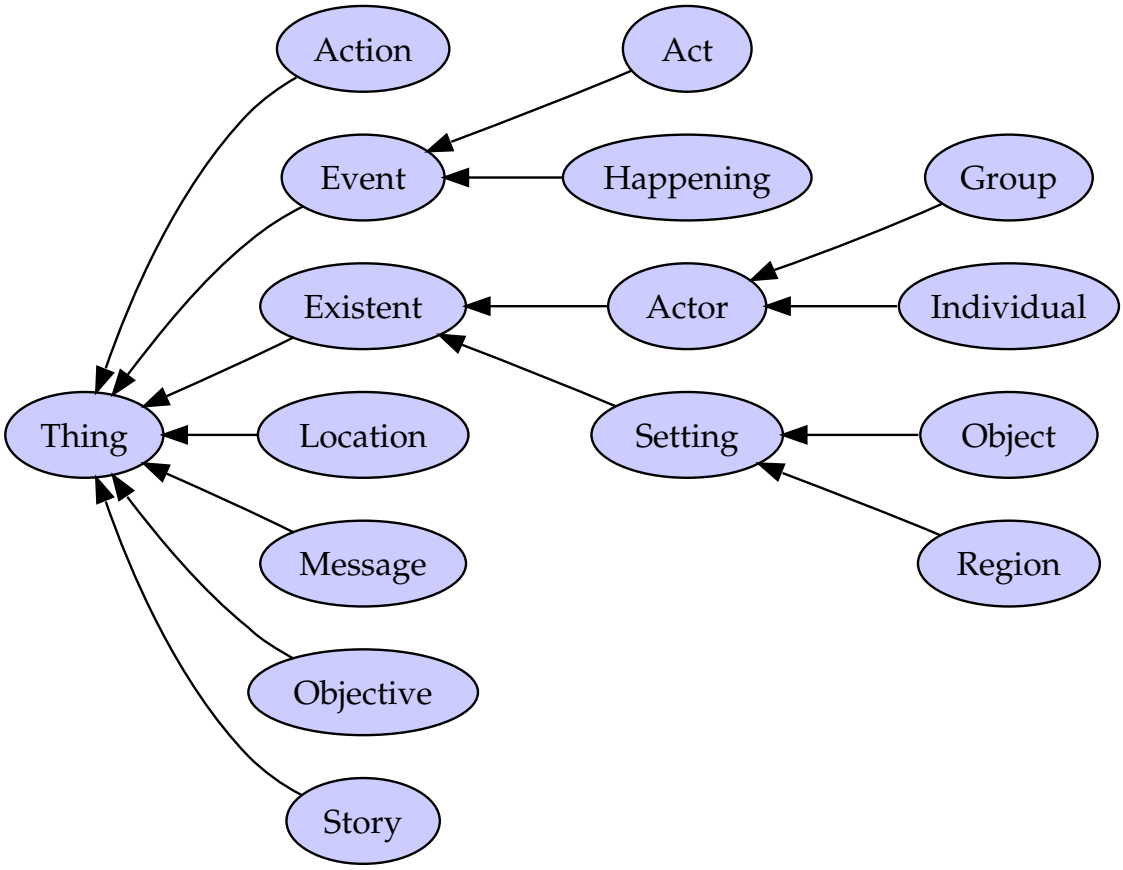

FIGURE 5.2: Highest level of the ontology.

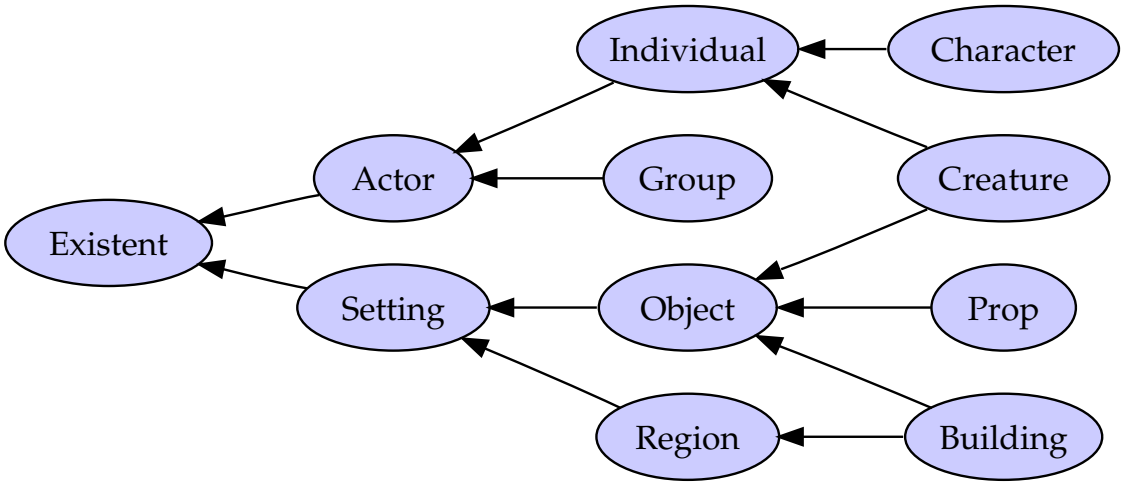

FIGURE 5.3: Basic ontology of existents.

ble, it is necessary to make refinements and extensions to tailor it to a particular concrete domain. Therefore, after a thorough discussion of the ontology we then describe how the ontology was extended to a particular concrete virtual environment that we have used for testing and evaluation of this thesis: the commercial *Neverwinter Nights* computer role-playing game.

The figures in this chapter show only *subsumption* relationships between concepts, indicated by arrows (an arrow from A to B indicates that A is subsumed by B, $A \sqsubseteq B$). Other relations and properties are described in the text.

5.3 Existents

The most basic elements of the ontology are those relating to the objects, characters and settings that exist in the environment we are narrating. We follow Chatman in labelling these elements as *existents*. The basic structure of the ontology of existents is shown in Figure 5.3. The class of existents is made up of settings and actors. Setting refers to the physical objects and places that make up the story world and that can be perceived or manipulated by other entities in the environment. An actor is any active entity capable of performing actions in the world. These notions of actor and setting overlap in that there are physically embodied agents in the world which are capable of performing actions. We group this intersection under the heading 'creature', although it might also include

entities such as machines or robots. An actor may be a physical entity, or it may not. For instance, creatures within the environment are clearly both physical entities and able to act. However, a group of actors can be considered as an actor in its own right (the group can act collectively) while having no distinct physical presence. While a group at any particular time is represented by its members, which are likely physically embodied within the environment, it is not correct to identify the group with its members, as the membership may change over time. In addition to groups, a particular environment may support actors in a broad sense that are not physically embodied (for instance, there may be god-like characters that act on the environment but are not a part of it).

The basic properties associated with existents are as follows:

- hasName: A simple textual (string) name. Every existent is assumed to have a name (and only one; this is a functional role), but the name is not assumed to be unique. We do not structure names beyond a simple string of text, e.g., into first and last name.
- hasPart: Existents can consist of parts, which are captured by this role.
- isPartOf: The inverse of hasPart.

5.3.1 Setting: Locations and Regions

Objects that exist in the environment must exist somewhere in that environment. We use a hierarchical notion of *regions* to represent where an object is located in the environment. A region is a physical entity that is capable of containing other physical entities, including sub-regions. A single top-level region represents the entire environment. Each region defines its own local coordinate system in three dimensions, and a *position* is a single point within a region. A general transitive role isWithin (and its inverse role contains) relates a Location (sub-region or position) to the Region it is contained in. We make no further restrictions on location containment: a location could be contained directly within 2 or more regions, and the region hierarchy is not assumed to have a single root (it is thus more of a forest than a strict hierarchy). In practice, it is expected that the region hierarchy will be rooted and a strict tree, with each region having exactly one parent region (apart from the top-most universal region). This scheme allows considerable flexibility in describing the location of entities: we can describe their location to a fine degree as a position, or at a coarser level, stating that they are merely somewhere within a particular region. This

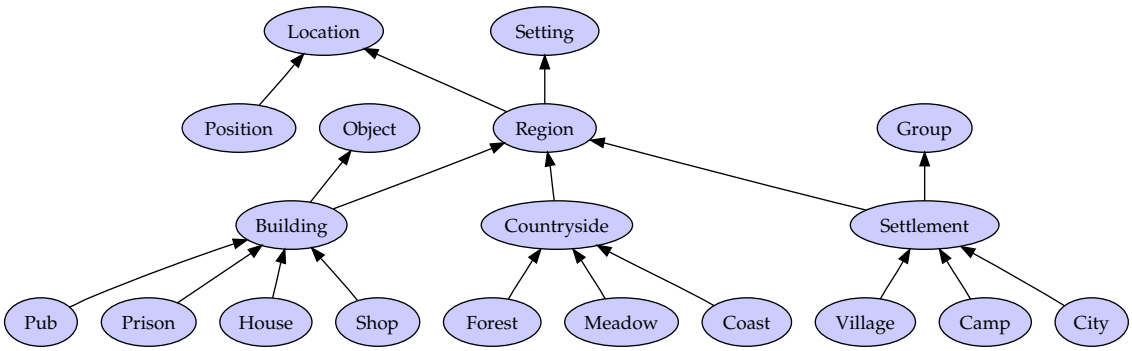

FIGURE 5.4: Sample view of the region ontology.

inexact representation of location is useful for describing objects whose position we are not absolutely certain of, and is also useful for describing sub-regions, where we may not be able to locate them exactly (e.g., the virtual environment may not provide such information). The use of the hierarchy also has benefits when describing events that have occurred, as we can choose whether to describe them exactly ('the battle occurred in Etum Castle District, just outside the Capitol Building') or more generally ('a battle occurred in the city of Etum').

The actual ontology of different regions is quite large, dividing regions up to quite a fine grain into buildings, rooms, settlements, and rural countryside areas. These basic divisions are then further broken down into specific terms. A fragment of this ontology is shown in figure 5.4, although only a few representative classes are shown. The reason for the level of detail in this part of the ontology (and other parts relating to concrete entities in the environment) is to permit greater flexibility when generating natural language expressions describing the setting and entities in a narrative: the more distinctions that are made in the ontology, the greater the diversity of phrases that can be generated. The use of subsumption relations also means that we can provide general descriptions of entities and then incrementally provide more specific descriptions for individual concepts (although this does require the use of inheritance or default reasoning when generating descriptions). The downside of this increased detail in the ontology is that it becomes increasingly difficult to know what is the 'best' way to represent any particular concept. The final ontology was developed as a result of much iterative development, finding which distinctions made most sense for generating narrative fragments and which distinctions are needed for other concepts in the ontology. The resulting ontology bares some resemblance

to the NASA SWEET (Semantic Web for Earth and Environmental Terminology) ontology, which also describes regions in some detail with a similar breakdown into categories, and in future this and some other parts of the ontology might be adapted to make direct use of the SWEET ontology.

A region is itself quite simple in terms of its representation. Each region has a name (a text string) and width, breadth and height dimensions (in metres). Regions are currently assumed to be cuboidal in shape. As regions are physical entities, they can also be located relative to a containing region, although for regions an exact position may not be given. Most of the sub-division of region categories is purely assertional; there are no particular distinctions in the represented properties of regions that would allow them to be classified as one type of region or another. The two exceptions to this are buildings (and their rooms) and settlements. A building is distinguished from other regions as it is also considered an object in the ontology. The Object concept is intended to distinguish physical entities that can be the object of actions. For instance, a building may be created (built), and can be damaged, or demolished. (The same might also be true of some other regions, but it is unlikely in most of the virtual environments we will be considering). A settlement, on the other hand, is both a region and a group. The members of the settlement group are the inhabitants, and the leader is the mayor, lord or governing body for that settlement. Representing settlements as groups is a useful step, as it enables reasoning about a number of interesting types of events (such as uprisings or invasions) that will be discussed later.

We include one deontic notion within the ontology, related to regions, which is the notion of some action (or class of actions) being *forbidden* within a region, loosely capturing a simple notion of social norms or laws. This is captured by the role hasLawAgainst and its inverse isIllegalIn that connect an action and a region. The meaning of this relation is that the given action is illegal within that region, and is used in the definition of crimes, described later (in the section on events). A more comprehensive notion of crime could be formalised by associating deontic concepts with groups, such as a group forbidding or permitting certain actions. This could then be used to describe certain crimes, such as theft as being taking some property without permission. Currently, however, the ontology is not that sophisticated, and instead relies on the simple notion of illegal acts given. We also classify some actions as crimes automatically—for instance, all Steal acts are crimes (thefts). This relies on these actions being primitively observable in the environment, as we have no sufficient conditions for their recognition. This is the case in *Neverwinter Nights* for

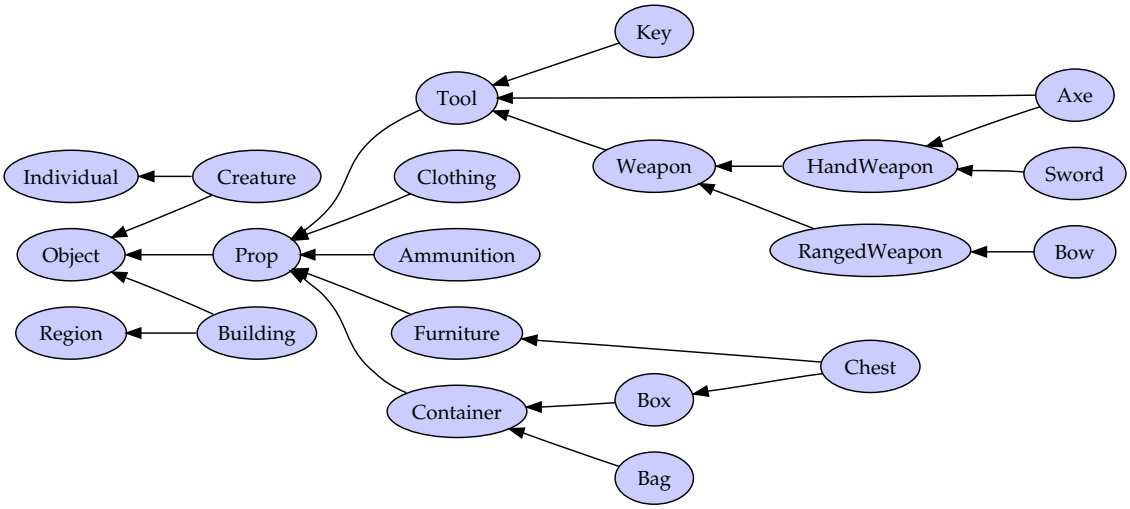

FIGURE 5.5: Objects and Props

instance, where various forms of theft (such as pick-pocketing) are observable as distinct actions in the environment.

5.3.2 Setting: Objects and Props

We now come to the physical objects that make up the environment. As the range of objects that exist in a virtual environment is likely to vary greatly from world to world, we only introduce some general categories of objects in the main ontology, and leave filling out specific objects mostly for environment-specific sub-ontologies (such as that developed later for *NWN*). The basic categories of objects are shown in figure 5.5. At the top-level, we divide objects into Creatures, Props, and Buildings. In addition to a name (as for all existents), an object is also assumed to have a single location, given by the hasLocation functional role that relates an object to a Location (typically a position, but could also be a more vague region). Properties of objects, such as location, can change over time. These properties are therefore *fluents* and need to be carefully dealt with. The details of how this is accomplished in the ontology are discussed later, in section 5.4. We do not attempt to model more complex physical characteristics of objects, such as their volume, shape or mass. Instead, we assume that these details are taken care of by the environment. For instance, if an agent observes a creature carrying some prop, then we can assume that the creature is indeed capable of carrying it (i.e., it isn't too heavy or too large).

A Prop is any passive object that can be moved or manipulated in the environment. As such, props come in a wide variety of different forms and with different purposes. The goal of the ontology is to classify props according to their functional role, and the part they can play in a story. We do not record any descriptive information about props, such as their colour or texture: just their functional categorisation. While these attributes might help when generating textual descriptions, this is not an area that is currently made use of in the framework. Tools are objects that can be used while performing an action (see section 5.5). For example, a Key object can be used when performing the Unlock or Lock actions. We do not record for each tool which actions it can be used for, but again leave it up to the environment to constrain which tools are appropriate for which actions. This also means that we cannot infer the use of some tool from an observation of an action, but in most cases the use of a tool would be directly observable. An instance of an action can be associated with a tool using the using role that links an Action to a Tool. Due to the nature of the game environments we are targetting, the most important type of tool is that of a Weapon. A Container is a prop that can contain other props. A container is assumed to be relatively small in size, and so cannot contain general objects, like a region, but only other props.

5.3.3 Actors and Groups

The most important elements of the environment are those that are capable of action within the environment, and therefore capable of causing events. Such entities are categorised as Actors in the ontology. As mentioned previously, the notion of actor is abstract and does not imply embodiment in the environment. An overview of the concepts involved in this part of the ontology is shown in figure 5.6, along with the subsumption relations between concepts. The main division is between Individuals and Groups. A group is simply a collection of actors, related to the group via hasMember role and its inverse isMemberOf. As groups are themselves actors, it is straightforward to describe sub-groups as simply groups which are members of other groups. Groups can have an organisational structure, based on the idea of *roles* from the literature on multi-agent systems and organisations. We describe organisational roles using description logic roles. The most general such role is hasMember, and other roles are always specialisations of this membership role (using role hierarchies available in $\mathcal{SROIQ}$), for instance hasLeader, which is a functional role. We do

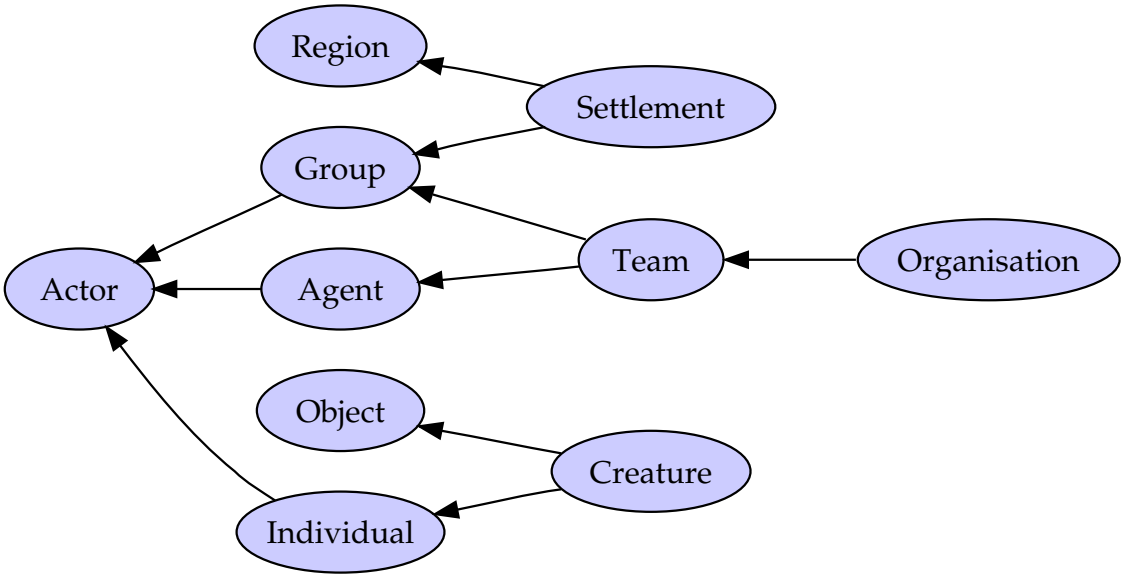

FIGURE 5.6: Actors, groups, agents and teams.

not currently reify the notion of role as a distinct concept in the ontology, and so do not express deontic notions such as permissions and obligations related to roles (for instance, stating that only the leader of a group can perform certain actions).

An actor is not presumed to be an intelligent, thinking being. For instance, a simple automatic door could be considered an actor in our framework (which acts to open and close itself as people approach). To distinguish actors who act in a purposeful manner, we introduce a notion of Agent as an actor to which we can ascribe various mental attitudes (beliefs, goals, etc.) — i.e., we can adopt the *intentional stance* towards agents, but not actors in general [35]. Currently the only mental attributes we attribute to agents are those of *objectives*, discussed later. A notion of Team is then developed as being a group that is also an agent. We do not currently represent more sophisticated aspects of teams, such as shared plans or joint intentions. An Organisation is a long-lived team. Finally, we link this part of the ontology to the physical environment by defining the concept of Creature as an individual that is embodied in the environment (i.e., is an object).

5.4 Time

The temporal aspects of our ontology are based on a view of time as a linear sequence of time points. This should be sufficient for our purposes, as we are narrating events that have already happened, rather than predicting or planning for future events. We define a time interval as a pair of time points (start and end times), and then import the relations from Allen's Interval Temporal Logic shown in Table 2.1 (page 33). An *event* is then a time interval with some other properties. The full description of events and types of events is given in Section 5.7 (page 103). In addition to representing time itself, we also need to be able to describe that some objects and properties change over time. For instance, the location of objects can vary over time, as can physical characteristics such as their weight or colour. It is not just physical objects that can vary over time, but characters and groups (which may not have a direct physical representation) can also change. For example, the membership of a group may change over time, and the beliefs of a character may also change. In general though, change is limited to *existents*[7]. Properties of existents therefore need to be represented by *fluents*. However, there are some unique challenges in describing fluents in description logic, due to the restriction of predicates to unary or binary predicates only. We cannot therefore simply add an extra situation or time argument to such roles. One solution would be to use a separate Observation concept which is a type of event (and thus situated in time) and which we can then associate with an existent and then use as the subject of various roles. In FOL we would write:

$$\mathsf{Observation}(o) \wedge \mathsf{start}(o, t_1) \wedge \mathsf{end}(o, t_2) \wedge \mathsf{ofExistent}(o, x) \wedge \mathsf{Building}(x) \wedge \mathsf{colourOf}(o, \mathsf{Red}) \wedge \ldots$$

The problem with this approach is that the domain of all fluent roles then becomes an observation, rather than the object it actually describes. It is not the observation which is red, but the object that was observed. The solution we have adopted is to treat existents somewhat like situations, in that they encapsulate a state at a particular moment in time. However, unlike in the situation calculus, these situations are limited in scope to a particular existent rather than capturing the state of the entire world. Therefore a symbol denoting an existent denotes that existent during a particular interval of time. We can then

[7] This also includes settings, such as regions. Consider, for example a vehicle, which is a region (can contain objects), but one whose location can change.

write:

$$\mathsf{Observation}(o) \wedge \mathsf{start}(o, t_1) \wedge \mathsf{end}(o, t_2) \wedge \mathsf{ofExistent}(o, x) \wedge \mathsf{Building}(x) \wedge \mathsf{colourOf}(x, \mathsf{Red}) \wedge \ldots$$

This solves the problem of how to represent change, and also ensures fluents are connected to the correct object. However, we now have the problem of identifying when two variables refer to the *same* existent (i.e., of representing identity). We cannot use the OWL sameIndividualAs statement, as this would equate the individuals with no regard for any differing fluents connected to them. Instead, we must introduce our own sameAs role to connect each observation of the same existent to each other. Initially, this would seem to suggest that we need an assertion for each pair of existent-situations as an existent changes over time, leading to a huge number of such assertions. However, we can simply declare a single canonical individual for each existent whose corresponding observation event extends infinitely forwards and backwards in time, and for which the only assertions are for properties that are guaranteed to be static (i.e., non-fluents). We then need only assert that each observation is sameAs this canonical individual.

$$\mathsf{Observation}(o_1) \wedge \mathsf{start}(o_1, -\infty) \wedge \mathsf{end}(o_1, \infty) \wedge \mathsf{ofExistent}(o_1, x_1) \wedge \mathsf{Building}(x_1)$$

$$\mathsf{Observation}(o_2) \ldots \wedge \mathsf{ofExistent}(o_2, x_2) \wedge \mathsf{colourOf}(x_2, \mathsf{Red}) \wedge \mathsf{sameAs}(x_2, x_1)$$

$$\mathsf{Observation}(o_3) \ldots \wedge \mathsf{ofExistent}(o_3, x_3) \wedge \mathsf{colourOf}(x_3, \mathsf{Blue}) \wedge \mathsf{sameAs}(x_3, x_1)$$

With a suitable axiomatisation of sameAs it is then possible to reason about the changing state of this building over time, while minimising the number of assertions needed. In practice, the virtual environment will usually provide some identity mechanism (such as a unique identifier) for each existent, and so we can simply use this mechanism rather than filling the knowledge base with such assertions. The scheme we have adopted therefore combines some aspects of both the situation calculus and Allen's interval temporal logic, but with some unique aspects not present in either. The treatment of existents as localised situations appears to be novel. The notion of an observation event has some resemblance to the HoldsAt predicate in the event calculus, as it connects localised assertions to intervals of time.

5.5 Actions

Actions form the bulk of the primitive events, directly observable in each environment. Figure 5.7 shows an overview of the main classes of actions in the ontology. Actions are

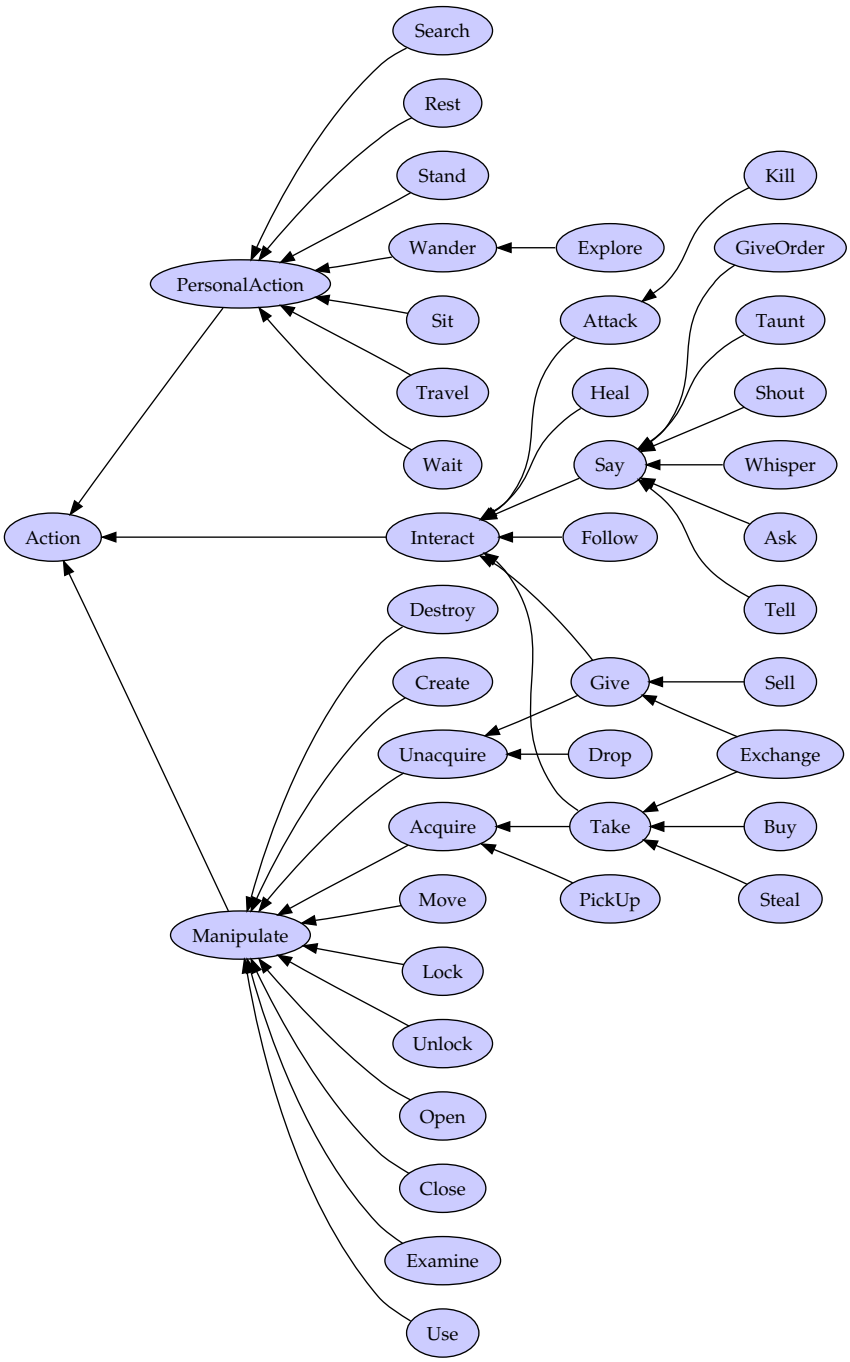

FIGURE 5.7: Actions

not directly linked to the character (or group) that performs them (i.e., the 'subject' in grammatical terms). Instead, this connection is made when the action is performed: an Act event connects an action to an actor and a time and location. This is described more fully in the next section on events. The reason for this distinction is to allow actions to be described independently of a particular instance of them being performed. For instance, we can describe a character as being *capable* of performing some action, or talk about agents delegating actions to other characters. Actions are divided into three main categories:

1. *Personal Actions* are actions performed by an individual on his or her own, with little or no involvement from other actors or objects. These actions include travelling from one location to another, sitting down or standing up, and resting. Typically, these actions play little part in the interesting events of an environment, and merely provide background information.

2. *Manipulations* are those actions in which a character manipulates some (passive) object or prop. The target of this action is described by the hasObject role which associates an action with one or more objects.

3. *Interactions* describe actions that involve two or more (active) characters. The target actor is described by the hasTarget role, which associates an action with an actor. Some actions, such as Give and Take are both manipulations (of the object being given or taken) and interactions (with the receiver or giver), in which case there will be both hasObject and hasTarget roles defined for the action. In these cases, hasObject refers to the direct object, and hasTarget the indirect object, in linguistic terms.

For instance, the action of 'giving Sally a drink' would be described as:

$$\mathsf{Give} \sqcap \mathsf{hasTarget} = \mathit{sally} \sqcap \exists \mathsf{hasObject}.\mathsf{Drink}$$

The only role that is defined over all actions is that of using some tool to aid in performing the action. This role has a domain of any action and a range of any tool. The purpose of this role is to allow us to describe that a certain action was performed using a certain tool. For instance, we might state that a door was unlocked using a particular key, or that a person was attacked using a sword or some other weapon. This relation is separate to the hasObject role used in manipulations, as the object being used is auxiliary to the action. In grammatical terms, it is an adjunct, and as such is an optional extra detail that could be

omitted without changing the meaning of the action. For instance, we might expand our previous action and specify that Sally should be passed the drink on a tray:

$$\mathsf{Give} \sqcap \mathsf{hasTarget} = \mathit{sally} \sqcap \exists \mathsf{hasObject}.\mathsf{Drink} \sqcap \exists \mathsf{using}.\mathsf{Tray}$$

Here, the drink is the direct object, Sally is the indirect object, and the use of the tray is an adjunct.

5.5.1 Personal Actions

Personal actions are those actions that a character performs that affect only themselves. These actions include moving themselves around, either within a region (wander, explore) or between regions (travel), resting, sitting or standing up, waiting, or searching. Most of these actions are not likely to form a core part of any story in which that character features, but mostly serve to fill in the details of the story. In Chatman's terms, these are satellite events rather than kernel events. As such, these actions are described only on a very shallow level, and have few action-specific relations. Indeed, the only personal action that is described beyond simply asserting it, is that of Travel, where we define the toDestination location as a functional role.

5.5.2 Manipulations

Manipulations are any actions that primarily involve some object other than the character performing the action. Typically, these involve changing the state of some prop. For instance, the ontology defines actions for physically moving an object (to some destination location[8]), acquiring and unacquiring objects (giving, taking, dropping), and using tools. It also includes more specific actions, such as locking and unlocking doors.

5.5.3 Interactions

The most interesting basic actions, from the point of view of constructing stories, are interactions between characters, as these form the basic elements of most narratives. Interactions described in the ontology include exchanges and trade between characters (give, take, buy, sell), attacking and healing other characters, and other basic interactions such as following somebody. Perhaps the most interesting interactions between characters are

[8]Note that this action typically implies the actor travelling to the destination.

communications, which are very crudely captured in the Say class. This class could be expanded to include all kinds of illocutionary actions (i.e., speech acts). At present this area of the ontology is not well developed, as unfortunately, it is unlikely that many such speech acts could be accurately discerned from the environment without employing complex natural language understanding (NLU) techniques. We therefore concentrate on a few simple distinctions that are relatively common in current computer games. For instance, a simple distinction between whispered and shouted speech is often available. Many games also include taunts as a basic action, and likewise orders can often be given by players to other characters. Each Say action is linked to an instance of the Message class via a hasMessage functional role. Currently, messages are simply text strings (associated with a separate hasContent role on messages). The only current exception is the GiveOrder speech act, whose associated message, Order, is also a Task (see the following section on objectives). This, in principle, allows the representation of orders in terms of the actions they are intended to perform, useful in various tactical game environments.

5.6 Objectives and Plans

Central to most narratives are the objectives that characters are trying to achieve in the world, and the plans that they have for trying to achieve them. In our ontology, these notions are represented by the top-level classes Objective and Plan respectively. An objective is simply something that some character (or group of characters) is trying to achieve. We use the term 'objective' rather than 'goal' or 'desire' as we are intending to capture only objectives that are grounded in some concrete action or state of the world, and are not trying to capture or describe the mental attitudes of the agent (which are in general unknowable). This might be a state of the world they are trying to bring about, or some specific action that they are trying to perform. Figure 5.8 shows an overview of the various objectives described in the ontology. Objectives fall into three main categories:

1. *Tasks* are simple objectives to perform a certain action.
2. *Personal Objectives* are the objectives of individuals.
3. *Political Objectives* are objectives relevant to groups and societies.

The simplest form of objective that a character can have is that of a Task. Tasks are the basic building blocks of more complex objectives and plans. A task is simply an action that a

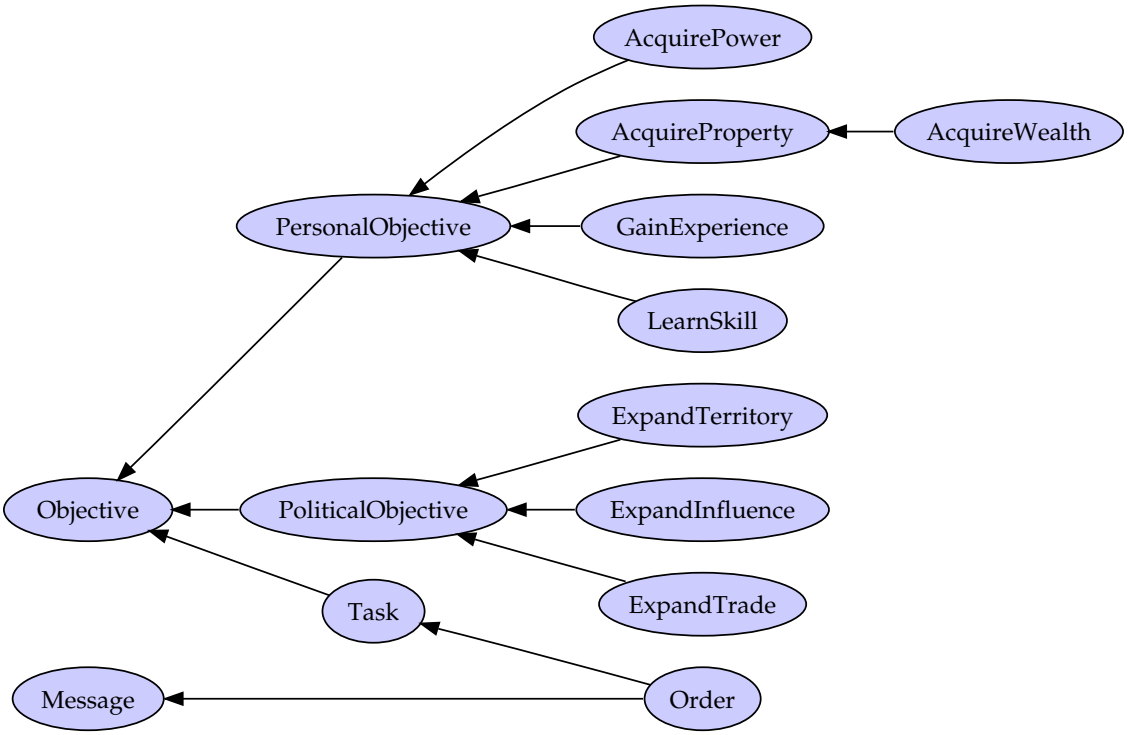

FIGURE 5.8: Part of the objectives taxonomy.

character wants or needs to perform. A task is associated with the action using a toPerform role, whose domain is the class of tasks, and whose range is the class of actions.

Objectives other than tasks are assumed to be states of the world that an individual or group is trying to bring about. This captures the distinction between *intend-to* (tasks) and *intend-that* (other objectives). Currently, objectives other than tasks are only partially modelled in the ontology: we tentatively identify a number of potential personal and political objectives that individuals or groups may be pursuing, but we do not capture the specifics of what it means to have that objective, or under what conditions the objective is considered to be met or unachievable. These specifics are instead left to particular environment ontologies, where certain objectives may be built-in to the game rules and provide mechanisms for detection of their achievement. There are obviously many more potential civil and political objectives that we might consider. For instance, a good civil servant or politician, we would hope, would have objectives to improve services for the community as a whole, and to provide education and justice. These more noble objectives feature little in most current games however, and generally lead to quite dull stories! For this reason, we have not included them in the current ontology. The Game Ontology Project [124] is an attempt to develop a taxonomy of game-related objectives and rules, which was used as a reference during this thesis, but not directly integrated into the ontology.

5.6.1 Plans and Missions

A plan is a partially ordered sequence of steps required to achieve some objective. We represent plans using the class Plan, which is related to the objective it is intended to achieve via a toAchieve role. The plan itself consists of a set of steps (Step instances), indicated by the hasStep role. Each step itself has an objective, indicated by a functional hasSubObjective role. Typically, we would expect most sub-objectives to be simple tasks, but they could be more complex objectives, leading to sub-plans. Following [119] (see section 2.4.1), we can identify the following constraints that may need to be placed onto the steps of a plan in order to make it coherent:

1. A set of qualitative constraints on these steps, specifying the temporal ordering of steps;

2. Metric constraints on the steps, specifying properties such as their maximum duration, or the maximum time between steps;

3. Coreference constraints, specifying e.g., that the same agent must perform each step in a plan.

Of these, only the first two are expressible within the $\mathcal{SROIQ}(D)$ language. We limit this case to a single dependsOn role between steps, that states that a certain step can only be performed after certain other steps have been performed. This corresponds to Allen's *after* relation, which we consider to be sufficient for our purposes. Metric constraints are not currently encoded into the ontology, but could be specified as simple roles (e.g., $rolehasMaximumDuration$). Instead, hard-coded maximum values are used in the implementation. Coreference constraints are not expressible in OWL, and so we must extend the language in order to deal with these. Coreference constraints are similarly hard-coded, with the logic that the same actor must perform each step of a plan. Note that this still allows for groups of agents to perform a plan, in which case the group becomes the actor. The details of how plan recognition is performed are discussed in the following chapter.

A special case of a plan is that of a *mission*, which we take to be a set of objectives given to an actor by another actor. This captures the common notion of missions or quests given in many role-playing games. The only addition to the general plan concept is that a mission is assumed to have some *giver*, denoted by the isGivenBy role, and also potentially some sort of *reward* for successful completion, denoted by the hasReward role (whose codomain is Object).

The ontology of objectives is also extended to define notions of PlanObjectives and MissionObjectives as follows:

$$\mathsf{PlanObjective} \equiv \mathsf{Objective} \sqcap \exists \mathsf{isObjectiveOf}.\mathsf{Step}$$

$$\mathsf{MissionObjective} \equiv \mathsf{PlanObjective} \sqcap \exists \mathsf{isObjectiveOf}.(\mathsf{Step} \sqcap \exists \mathsf{isStepOf}.\mathsf{Mission})$$

5.7 Events

The most important notion present in our ontology is that of an *event*. An event is the principle notion of time within the ontology. Each event represents a time interval, which is made concrete by two functional roles: startsAt and endsAt that indicate the start and end times of the event, respectively (time points are described using the XML schema dateTime data type). In addition to having a start and end time, an event also has a location, given

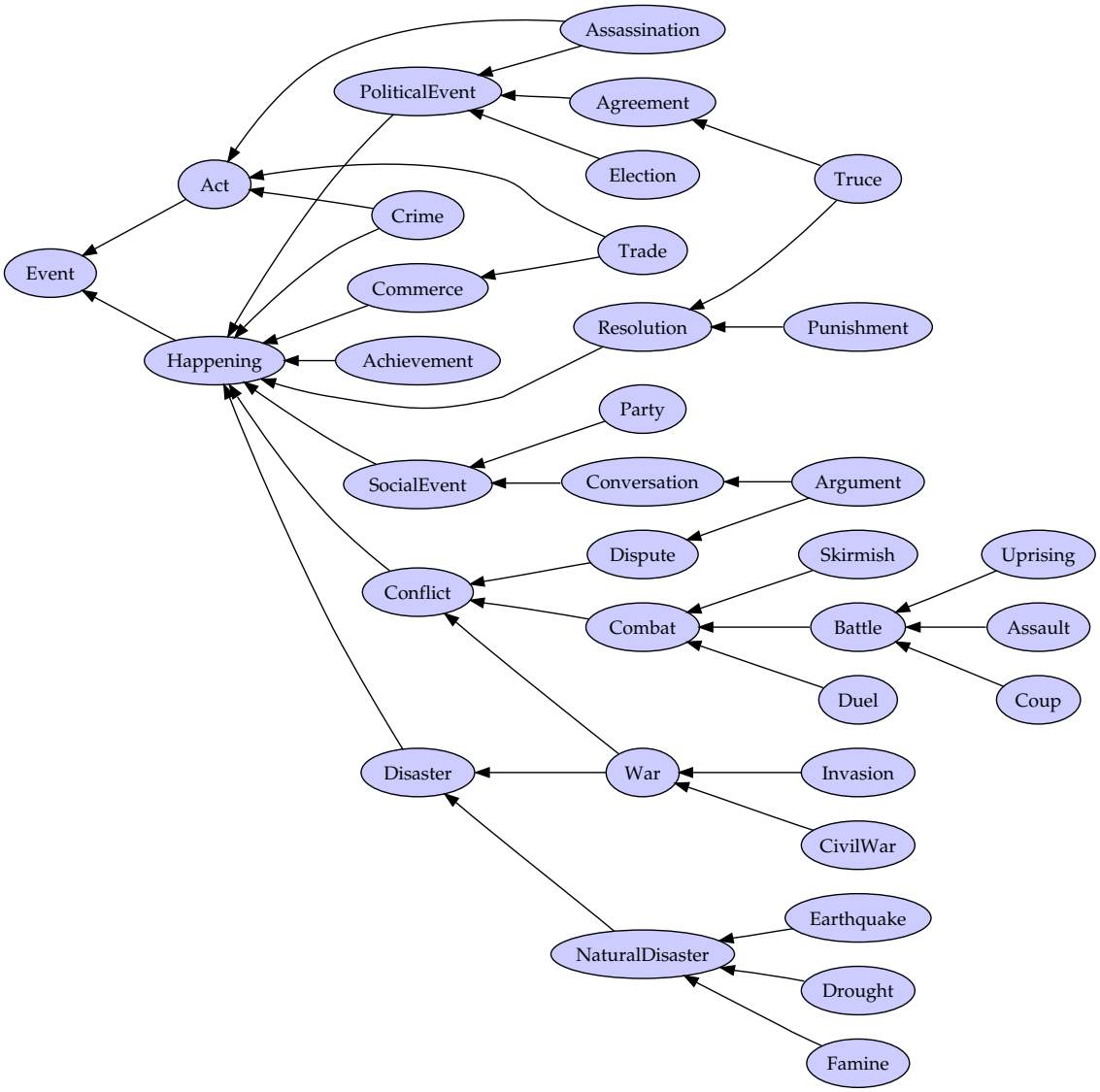

FIGURE 5.9: Events.

by the occursAt role. The location of an event may be precise, as in an action, or it may be a general region, as in a battle. The ontology of events is shown in figure 5.9. The primary distinction, as per Chatman, is between Acts and Happenings. An Act is simply an occurence of an action (a single action), whereas a Happening is a more general event that is not attributed to a particular actor.

A number of general roles are defined on events to connect them to other events and existents. Firstly, we incorporate all of Allen's temporal relations as roles between events (see table 2.1), allowing the ordering of events to be captured. These relations can be defined entirely in terms of the start and end times of events, but this is again not expressible in $\mathcal{SROIQ}(D)$, so we have to resort to an extension of the language which includes these relations as explicit inference rules. We further include a notion of *causation* in the ontology, by providing causes and isCausedBy roles, which are specialisations of the temporal before and after roles respectively. These roles capture the stronger notion that the preceeding event was directly responsible for the succeeding event. We also allow events to have arbitrary sub-events, indicated by the hasSubEvent role, which can be specialised for particular event types. Finally, we associate with each event the set of all actors that are involved in the event using a hasParticipant role, and its inverse, isParticipantIn. Being a participant in an event is a complex notion. In particular, we wish to state that the participants in an event include all of the participants in any sub-events. Again, this is not expressible in OWL, so we extend the language with a particular inference rule. We also capture the inference that the actor responsible for an action, and any actor targets of that action, are also participants.

The roles described in last two paragraphs together address the six main *wh-questions* that a report of an event should answer:

What? The class(es) of the event individuals gives this information;

Who? The hasParticipant role;

When? The startsAt and endsAt roles;

Where? The occursAt role;

How? The hasSubEvent role links an event to more detailed events describing how the event happened;

Why? The isCausedBy role links an event to its cause, where that is known. A deeper notion of motive or reason is not currently developed in the ontology.

The extent to which these roles answer the given questions depends to a large part on the degree of sophistication of representing different situations present in the ontology, and also in how well the agents are able to observe or infer this information. The former concern is addressed in this chapter, by developing a sophisticated ontology of events in virtual environments. The latter concern is addressed in the next chapter, when we look at how the agents are actually implemented.

5.7.1 Observations

An Observation is an event that grounds the fluents described in section 5.4, by connecting an existent individual (with its associated fluents) to a concrete time. This captures the notion that an existent was observed to have those fluent valuations at a particular time interval. An observation is simply an event with an associated existent, indicated by the of role. A particular observation event therefore captures the following information: that an existent was observed at a given time interval and a given location to have the particular properties indicated by the fluents associated with that existent. The main purpose of an observation event is to allow a story to refer to an existent *as it was when the event occurred*. This is important as stories may not be immediately told, but may instead by retold several times at different reference times.

5.7.2 Performing Actions

The performance of actions is captured by the Act concept. An act is the most primitive type of event, and has no sub-events. All complex events (i.e., those events with sub-events) are ultimately grounded in acts. An act is an event that indicates the performance of an action, via the performs role. In addition, each act also records the actor (singular) that performed the action using a hasActor role. The complete definition of an act is thus:

$$\mathsf{Act} \equiv \mathsf{Event} \sqcap \exists \mathsf{performs}.\mathsf{Action}$$

$$\mathsf{Act} \sqsubseteq \exists \mathsf{hasActor}.\mathsf{Actor}$$

$$\begin{aligned}
\mathsf{Assassination} &\equiv \mathsf{Act} \sqcap \exists \mathsf{performs}.(\mathsf{Kill} \sqcap \exists \mathsf{hasTarget}.\mathsf{Leader}) \\
\mathsf{CombatAct} &\equiv \mathsf{Act} \sqcap \exists \mathsf{performs}.\mathsf{Attack} \\
\mathsf{Crime} &\equiv \mathsf{Act} \sqcap \exists \mathsf{performs}.(\mathsf{Action} \sqcap \exists \mathsf{isIllegalIn}.\mathsf{Region}) \\
\mathsf{Theft} &\equiv \mathsf{Act} \sqcap \exists \mathsf{performs}.\mathsf{Steal} \\
\mathsf{Theft} &\sqsubseteq \mathsf{Crime} \\
\mathsf{Trade} &\equiv \exists \mathsf{performs}.(\mathsf{Buy} \sqcup \mathsf{Sell} \sqcup \mathsf{Exchange})
\end{aligned}$$

FIGURE 5.10: Definition of complex acts.

A number of specific types of acts can also be defined, with sufficient conditions to describe them. The set of acts defined in the base ontology are as follows:

- An Assassination is an act which performs a Kill action and whose target is the leader of some group;
- A CombatAct is any act which involves performing an Attack on some other actor;
- A Crime is an act that isIllegalIn the region in which it is performed;
- Trade is defined as any buy or sell or exchange action.

The definition of these acts is shown in figure 5.10[9].

5.7.3 Achievements

An Achievement is any event that achieves some Objective. This is indicated by the achieves role that links the event to the objective (or objectives) that it achieves. How this is ascertained is not specified in the ontology.

5.7.4 Conflict

A Conflict describes any event in which there is a conflict between two or more actors (or groups of actors). Conflict includes both combat as well as more general disputes.

[9] Leader is defined as Actor $\sqcap$ $\exists$isLeaderOf.Group.

$$
\begin{aligned}
\mathsf{Combat} &\sqsubseteq \mathsf{Conflict} \\
\mathsf{Dispute} &\sqsubseteq \mathsf{Conflict} \\
\mathsf{War} &\sqsubseteq \mathsf{Conflict} \\
\mathsf{Combat} &\equiv \mathsf{Event} \sqcap \forall \mathsf{hasSubEvent}.\mathsf{CombatAct} \\
\mathsf{War} &\equiv \mathsf{Event} \sqcap \exists \mathsf{hasSubEvent}.\mathsf{Battle} \\
\mathsf{Duel} &\equiv \mathsf{Combat} \sqcap = 2\ \mathsf{hasParticipant}.\mathsf{Actor} \\
\mathsf{Skirmish} &\equiv \mathsf{Combat} \sqcap \geq 3\ \mathsf{hasParticipant}.\mathsf{Actor} \sqcap \leq 9\ \mathsf{hasParticipant}.\mathsf{Actor} \\
\mathsf{Battle} &\equiv \mathsf{Combat} \sqcap \geq 10\ \mathsf{hasParticipant}.\mathsf{Actor} \\
\mathsf{Argument} &\sqsubseteq \mathsf{Dispute} \\
\mathsf{Argument} &\equiv \mathsf{Conversation} \sqcap \exists \mathsf{hasSubEvent}.(\mathsf{Act} \sqcap \exists \mathsf{performs}.(\mathsf{Taunt} \sqcup \mathsf{Threaten}))
\end{aligned}
$$

FIGURE 5.11: Definition of conflicts.

Combat consists of events whose sub-events are only CombatActs. We distinguish three main types of combat, based on their scale: a Duel involves exactly two individuals as participants; a Skirmish involves between 3 and 9 individuals; and a Battle involves 10 or more individuals. A Dispute is used to indicate non-combat disputes, although it may be the *cause* of some combat. A particular sub-class of disputes is that of Arguments, which are conversations between individuals that involve a dispute. Finally, we also include the notion of a War, which is an event whose sub-events include at least one battle. These concepts are defined in figure 5.11.

5.7.5 Crime

Crimes are defined as shown in figure 5.10. The definition of a crime in description logic is much weaker than we would like, stating only that a crime is performing an action that is illegal in *some* region. We would like to formalise the stronger notion that a crime is an act that is illegal within the region in which it is actually performed. This kind of definition is not expressible within the $\mathcal{SROIQ}$ description logic that underlies OWL, however, so we settle for the weaker definition in the ontology and then strengthen this in the rule layer of

the implementation.

5.7.6 Commerce

The Commerce event class is intended to capture events that are related to buying, selling, and exchanging merchandise and monies. Currently this is limited to Trade events as described in figure 5.10. In future it may be possible to expand this area of the ontology to include such things as trade agreements and economic events.

5.7.7 Disasters

Disasters are happenings that have a catastrophic effect on some group of actors. The ontology currently distinguishes between man-made disasters, such as Wars, and natural disasters, such as floods, earthquakes and so forth. These are not modelled in much depth currently as they are not typical in current RPGs.

5.7.8 Social and Political Events

The ontology contains a preliminary taxonomy of social and political events, although this area is also rather undeveloped due to the lack of such events in our target research environments. Under social events, we distinguish Contests, Conversations, and Parties. The only contest we currently model is that of an Election. A contest can have a winner, which is indicated by the hasWinner role that relates it to the winning actor. An election is currently limited to being for the leadership of some group, and is indicated by the forGroup role. Both of these roles are functional. A conversation is an event whose sub-events are all acts that perform speech, and an argument is a conversation which includes threaten or taunt speech acts. A Party is a celebration of some other event, indicated by the inCelebrationOf role (which is a sub-role of isCausedBy). For instance, we could define a BirthdayParty as Party $\sqcap \exists$inCelebrationOf.Birthday, where Birthday is defined appropriately as an event class.

5.8 Stories and Plots

Finally, we are now in a position to define the content of a story according to the theory that our ontology represents. We first begin by defining the notion of a Plot, which is taken to be

a particular sort of Plan. The reason for this is that a plot typically consists of a set of *story objectives* which are met to create a story with the appropriate plot structure and dramatic arc. A PlotObjective is then defined analogously to the MissionObjectives defined earlier. We do not define particular types of plots in the current ontology, but leave that open for customisation in particular environments. A story can now be defined by associating a Story concept with the following relations:

1. ∃hasCharacter.Actor: relates a story to the characters that are involved in it;
2. ∃hasSetting.Setting: relates the story to its setting;
3. ∃hasPlot.Plot: links a story to its plot;
4. ∃hasEpisode.Event: links a story to the events that satisfy its plot structure.

We also define appropriate inverse roles for each of these, such as isCharacterIn. We can also now properly define the notion of Character mentioned earlier in this chapter. The definition is:

$$\mathsf{Character} \equiv \mathsf{Actor} \sqcap \exists \mathsf{isCharacterIn}.\mathsf{Story}$$

A degenerate case of a story is one in which there is only a single episode (i.e., a single step in the plot). Such a story is known as a Report in the ontology, and corresponds to the simple reports produced by the reporting agent framework of Daniel Fielding, described in Section 2.5.2 (page 45). The more complex story ontology developed in this thesis, building on the event and existent ontology, is capable of describing much more sophisticated narrative structures. For instance, the story grammars described in [10] or the theory of Plot Units [72] can both be encoded using the story ontology we have developed, allowing for a wide range of narrative structures to potentially be encoded.

5.9 Case Study: *Neverwinter Nights*

In this section, we develop the general ontology described above and demonstrate how it was adapted and extended to cover the *Neverwinter Nights* (NWN) game environment. In later chapters we will describe in detail how the agent technology we developed was integrated with the game world.

5.9.1 Setting

Neverwinter Nights (NWN) is based on the popular *Dungeons and Dragons* (D&D) table-top role-playing game, and has inherited a complex model of characters and their attributes. Dungeons and Dragons games, as the name suggests, are usually set in a fantasy world, inhabited by various mythical creatures and magical artefacts. *Neverwinter Nights* is set in a world that to some degree resembles medieval Europe—as it appears in myths and legends more than reality. Knights and wizards do battle with fantastical creatures and dragons, while the general population live in fortified cities or rural villages. The game world is populated with a variety of real and fantastical creatures, such as bears, eagles, and wolves, as well as dragons, trolls, and goblins. Settlements are populated with computer-controlled *non-player characters* (NPCs) that give information and quests to player characters (PCs). The behaviour of these NPCs is usually quite simple; typically they mill around waiting for a player to interact with them. Socially, NPCs are grouped into *factions*, which are simple groups. A simple reputation system affects how NPCs react to other NPCs and players, and is based on a numerical score each faction assigns to each other faction: a positive score results in a favourable and friendly reaction, whereas as large negative score will result in hostility and possibly even violence. Players' actions towards NPCs will alter these scores. Descriptions of characters and locations in NWN often mention groupings such as professional guilds, societies, and militias. However, these are usually only informally described with the group not being represented as a separate faction (factions are usually used to group inhabitants of a particular region, or more generally into 'commoner' or 'hostile' groupings).

The developers of *Neverwinter Nights* deliberately allow players to create their own game worlds, with new locations, props, and creatures, and to customise the behaviour of these worlds, effectively creating entirely new games that can be played by single or multiple players simultaneously. These user-created *modules* can differ significantly from the original content provided with the game, but usually include most of the same elements in terms of setting and creatures. The specific world module we have used for testing our framework is the *Land of Rhun*, which is a persistent world module designed to support many simultaneous players (up to 60 or so concurrently), playing over an extended period of time (weeks or months).

The location model of NWN consists of a set of distinct regions, which are con-

nected to each other via doors and magical portals. Each region is entirely self-contained and is roughly rectangular in layout. Regions range in size from individual rooms in a building, up to entire towns or large areas of countryside. Regions are not positioned absolutely or relatively in relation to each other, and the only spatial points of reference between regions are the locations of the doors that connect them. However, there is no constraint made on the placement of doors within a region, and so a sub-region may be laid out quite differently from how the external doors to it would suggest. Occassionally this may be done for effect (e.g., to disorientate the player), but usually level designers aim to keep things consistent. The world model of NWN can easily be described by the existing ontology, and the only adaptations needed are to add new classes for specific region types (wizards' towers, dungeons, etc.) and to populate an ontology with assertions about particular regions in the world of *Rhun*. We omit these details here. Similarly, the props and objects of NWN are also quite easily described within the base ontology we have already covered.

5.9.2 Magic

As with most fantasy settings, *Neverwinter Nights* includes a system of magic and the supernatural, inherited from *Dungeons and Dragons*. Magic in NWN is divided into two distinct classes: 'divine' and 'arcane'. Arcane magic draws its power from the world around the caster, whereas divine magic comes from the gods. In practice, this distinction just means that certain spells are only available to certain classes of characters, and there are some restrictions on how the different types of spell are cast. These differences are unlikely to be of interest to the reader of a narrative about NWN, and experienced players will be able to distinguish the types of magic anyway, so we do not model this distinction in the ontology. Likewise, arcane spellcasting is divided into a number of different 'schools', such as conjuration, illusion, necromancy and so forth, but we also do not model this aspect. Instead, we concentrate on the spells themselves, and their effects. The ontology is extended with a Spell concept with sub-concepts OffensiveSpell and DefensiveSpell. A list of spells can then be extracted from the environment and asserted as individuals in the ontology. Each spell is not described beyond having a name and being classified as either offensive or defensive in nature. The set of actions is then also extended to include a CastSpell action, with a spell role associating it to the spell that is cast, and a hasTarget role

as for other interactions. Finally, a SpellAttack concept is defined as casting an offensive spell against somebody, and this concept is subsumed by Attack.

5.9.3 Characters, Classes, and Attributes

Neverwinter Nights includes quite a large and complex statistical character system. Characters in the game can earn experience points, which periodically allow the character to 'level-up'. This process involves assigning points and skills to one of a number of different character classes, such as *Wizard*, *Druid*, *Fighter* and so on. These character classes define the capabilities of each character. For instance, wizards and sorcerers are able to cast arcane spells, whereas a fighter is able to wear heavy armour or use powerful weapons. Again, the ontology is extended to incorporate these aspects, but we chose to only model the most basic features rather than the full model. In particular, the ontology is extended with new concepts CharacterLevel and CharacterClass. Each actor can then be associated with a class using a hasCharacterLevel role which ranges over CharacterLevel instances. A CharacterLevel is then connected to a CharacterClass and a numeric level. For instance, to specify that a particular actor was a level 4 sorcerer, the following assertion could be used: $\mathsf{hasCharacterLevel}(x, c) \sqcap \mathsf{inClass}(c, \mathit{sorcerer}) \sqcap \mathsf{atLevel}(c, 4)$. As the levels and character classes of an actor may change over time, these are modeled as fluents, as described previously in this chapter. A new LevelUp event is also introduced as a form of Achievement, which can be used to report on individual experience increases.

CHAPTER 6

MULTI-AGENT IMPLEMENTATION

6.1 Introduction

The implementation of the framework closely follows the multi-agent design described in Chapter 4. In particular, the two use-cases shown in Figure 4.1 are implemented as two specific agent configurations: external *commentator* agents that relate narratives to an external audience, and embodied *witness-narrator* agents that observe events in the environment and produce the narratives (potentially also narrating them to participants within the game). Each such agent is built up from a number of basic agent *capabilities*, such as reporting, editing, and presenting of events, as well as navigation and route-following for embodied agents. In the following chapter we will often use abbreviations such as 'reporter agent' to refer to an agent that has a reporting capability. It should be noted that such agents are not usually dedicated reporters—as in the framework of Section 2.5.2 (page 45)—but instead support reporting as one particular capability.

The agents are implemented in the Jason agent-oriented programming language [11, 12], with some Java used to customise the implementation. Jason is a BDI agent programming language based on an extended version of the AgentSpeak(L) language [95]. The Jason extensions to AgentSpeak(L) include extending the belief base to support arbitrary PROLOG clauses, support for both strong and default negation, and allowing annotations on plans and beliefs (e.g., stating the source of a belief). Jason was chosen as it provides a relatively high level of abstraction and facilities (such as persistent belief bases) that are well-suited to our design. Compared to other recent agent-oriented languages, such as 2APL [33], the implementation of Jason was relatively well-advanced when the present work was commenced, with a working interpreter and a number of tools, such as

a debugger, RDBMS-backed persistent belief bases, and so on.

The design of the narrative agents system comprises a number of different aspects. There are a number of *domain-specific* considerations, dealing with the process of collecting and processing reports of events in our chosen environment, and a number of relatively domain-independent considerations, such as knowledge representation and reasoning for the agents, and coordination of the agent team as a whole. These considerations cannot be entirely separated, and domain-specific considerations always play some part in the overall design of the architecture.

6.2 Ontology Integration

An important aspect of the implementation is the integration of the ontology developed in Chapter 5 into the agent architecture. While an approach to integrating ontological reasoning into Jason had been described in the literature [86] at the time work began on the implementation, a practical implementation of this design was not available until much later [65]. For this reason, it was necessary to implement ontological reasoning by translation of the ontology into equivalent (or near-equivalent) PROLOG facts and rules that could be processed by the Jason belief base. A number of such translations have been described in the literature [118, 117, 104, 59], targetting either PROLOG or (disjunctive) DATALOG, in order to re-use existing optimised implementations of these languages. The approach adopted in this work is based on that described for Bubo [118], in which a subset of OWL-DL features are translated into equivalent PROLOG rules (suitably adapted for the syntax of Jason). The translation is summarised in Table 6.1. OWL classes (concepts) are translated into unary predicates, and properties (roles) into binary predicates. Note that Jason uses '`&`' rather than '`,`' to indicate conjunction in rules, and '`|`' for disjunction.

We differ from Volz et al in being able to make use of built-in Jason internal actions, such as `.count`, and strong negation, which simplify some aspects of the rules. Our treatment also departs from Volz in preferring translations that avoid infinite loops. For example, we translate transitive roles into two distinct predicates: an immediate predicate (with the same name as the role), and a transitive predicate, with a $_{tc}$ suffix. This avoids an infinite loop that would result from the direct translation when executed with the PROLOG operational semantics that Jason inherits (note that DATALOG is able to handle these cases). Some infinite loops can still remain from these rules, however, and so care had

OWL Class Expression	**Equivalent Jason Rules**
C `rdfs:subClassOf` D	$D(X) :- C(X).$
R `rdfs:subPropertyOf` S	$S(X,Y) :- R(X,Y).$
R `rdfs:domain` C	$C(X) :- R(X,Y).$
R `rdfs:range` C	$C(Y) :- R(X,Y).$
C `owl:sameClassAs` D	$D(X) :- C(X).$
	$C(X) :- D(X).$
R `owl:samePropertyAs` S	$S(X,Y) :- R(X,Y).$
	$R(X,Y) :- S(X,Y).$
C `owl:disjointWith` D	$\neg C(X) :- D(X).$
	$\neg D(X) :- C(X).$
`owl:TransitiveProperty` R	$R_{tc}(X,Y) :- R(X,Y).$
	$R_{tc}(X,Z) :- R(X,Y)$ & $R_{tc}(Y,Z).$
`owl:SymmetricProperty` R	$R(X,Y) :- R(Y,X).$
R `owl:inverseOf` S	$R(X,Y) :- S(Y,X).$
	$S(X,Y) :- R(Y,X).$
R `owl:hasValue` v	$R(X,v).$
R `owl:minCardinality` N	.count($R(X,\_)$,M) & $M \geq N.$
R `owl:maxCardinality` N	.count($R(X,\_)$,M) & $M \leq N.$
R `owl:allValuesFrom` C	$C(Y) :- R(X,Y).$
R `owl:someValuesFrom` C	$R(X,Y)$ & $C(Y).$
`owl:intersectionOf` $C, D, \ldots$	$(C(X)$ & $D(X)$ & $\ldots).$
`owl:unionOf` $C, D, \ldots$	$(C(X) \mid D(X) \mid \ldots).$
C `owl:complementOf` D	$C(X) :- \neg D(X).$
C `owl:oneOf` $\{a, b, \ldots\}$	$C(X) :- (X == a \mid X == b \mid \ldots).$

TABLE 6.1: Translation of OWL constructors into Jason rules, based on [118].

to be taken when constructing plans and rules making use of these ontology axioms that such cases were avoided. This was also helped somewhat by implementing a custom belief base for Jason, which always prefers ground facts over rules, regardless of the order in which they appear in the source code. This modification is also needed to support efficient ABox reasoning. The reason for this is that most ABox assertions are made at run-time as the agents encounter new individuals and receive new observations of those individuals. Conceptually these new assertions are added to the end of the belief base and so would be considered last in the default Jason and PROLOG operational semantics (i.e., *after* any rules). It is much more efficient to consider these ground facts first in a PROLOG style of execution and also reduces the likelihood of the infinite loops discussed. It should be noted that the operational semantics of DATALOG is much more suitable for this purpose, as it ensures efficient operation and eliminates the possibility of infinite recursion due to the stratified recursion restrictions. Such a DATALOG belief base would also permit much more efficient ABox reasoning in general (which is one of the reasons for translations of OWL into DATALOG [59]). Unfortunately, there was not enough time to implement such a belief base for Jason, or to formalise the resulting language. These rules were not intended to be a complete and faithful translation of OWL-DL into Jason, but instead to act as a stop-gap until more comprehensive support for ontological reasoning could be added. The translation rules given only approximate the semantics of OWL-DL. In particular, we do not model the `owl:sameIndividualAs` constructor (it is not currently used in the ontology), and the handling of some constructs, such as `owl:oneOf`, only supports reasoning in one direction. For example, the rules allow us to include that an individual belongs to some class C from a `oneOf` definition, but we cannot assert that only those individuals are members of C (which would require disjunction in the head of a rule, which is not expressible in Horn clauses).

The translation has been implemented as a plugin for the Protégé OWL ontology editor (version 3). All class and property predicates are defined in Jason using a lower-case prefix based on their XML prefix in the ontology. This requires using standard prefixes within the ontology editor, otherwise the same translation of the same ontology may result in differently named predicates, if the prefix mappings happen to have changed. The benefit of this approach is that it avoids name clashes and also ensures that all predicates begin with a lower-case letter, as required by the syntax of Jason (and PROLOG). An example of a translation produced by this tool is shown in Figure 6.1, showing both the

$$\mathsf{Assassination} \equiv \mathsf{Act} \sqcap \exists \mathsf{performs}.(\mathsf{Kill} \sqcap \exists \mathsf{hasTarget}.\mathsf{Leader})$$

```
/*
 * Class: http://www.agents.cs.nott.ac.uk/witness/-
 *        event.owl#Assassination
 */
event_Assassination(X) :-
    event_Act(X) &
    event_performs(X,Y) &
    action_Kill(Y) &
    action_hasTarget(Y,Z) &
    character_Leader(Z).
```

FIGURE 6.1: Translation of an example OWL-DL concept into Jason.

original OWL-DL (in DL syntax) definition and the resulting translation into a Jason belief base rule. The resulting rule allows Jason's backward-chaining belief inference process to closely approximate the ontological semantics specified in the OWL-DL ontology. Indeed, in this case the semantics of the two definitions is identical.

6.3 *Neverwinter Nights* Environment

As well as designing and implementing the agents themselves, it was also necessary to implement an interface to a suitable environment in which to test and evaluate the agents. As previously described, the *Neverwinter Nights* computer role-playing game was selected for the implementation, providing a reasonably rich and complex environment that is similar to current massively-multiplayer online role-playing games (MMORPGs), but with a more manageable number of participants. The main advantage of NWN was the support provided for user-created custom worlds and characters. This has lead to the development of a number of so-called 'persistent worlds' (PWs), which aim to provide a constantly running ongoing game experience for a community of players (typically ranging from dozens up to perhaps a hundred or so active participants). The test environment was developed by customising an existing persistent world environment ('module'), adding support for remote agents to create avatars in the world, perform actions, and receive notifications of events occurring nearby.

The integration with *Neverwinter Nights* was achieved by intercepting events delivered to NPCs (non-player characters) in-game using the built in *NWScript* language, which is used to implement the in-game character behaviour. The game was not designed to support the sort of third-party observation of events that is required for a reporting agent, so we instead 'hook' into the events as they are delivered to each non-player character and arrange for them to be also delivered to nearby reporter agents. This is accomplished by first formatting a message describing the action or event that is occurring and then scanning the environment to find reporters that have a line-of-sight to the object that originates the event. The message is then sent to these reporter agents using the *Neverwinter Nights Extender* (NWNX)[1] and the MNX plugin that enables UDP network communication. The overall setup is shown in Figure 6.2. While this works reasonably well for NPC events, it is not possible to intercept events for player characters in this way, as they are obviously not implemented as scripts but by the players themselves. In order to intercept player events it is necessary to perform polling. Each embodied reporter scans the environment approximately every six seconds (in a 'heartbeat' script) for nearby player characters and tries to determine what action they are currently performing. The lists of events and properties that can be determined for both player and non-player characters are described in the following sub-sections. For NPCs, the events that can be intercepted include when the NPC is attacked, when it suffers damage, and when something is taken from its inventory (e.g., if the NPC is the victim of pick-pocketing). In general, properties can be reliably observed for both players and NPCs, but event detection for players is much less reliable, as it relies on the player being in the middle of performing an action at the time at which polling occurs (which, as mentioned, is only once every 6 seconds). This approach is not ideal, being both rather inefficient and somewhat unreliable, but has sufficed for the current research.

6.3.1 Message Protocol

The message protocol used to communicate messages between the environment is a simple *ad hoc* textual protocol based on Jason literals. Each message from the environment is a string consisting of the name of the agent it is for, followed by a colon, followed by the message content as a Jason literal. For example, each agent is sent a status update once

[1] `http://www.nwnx.org/`

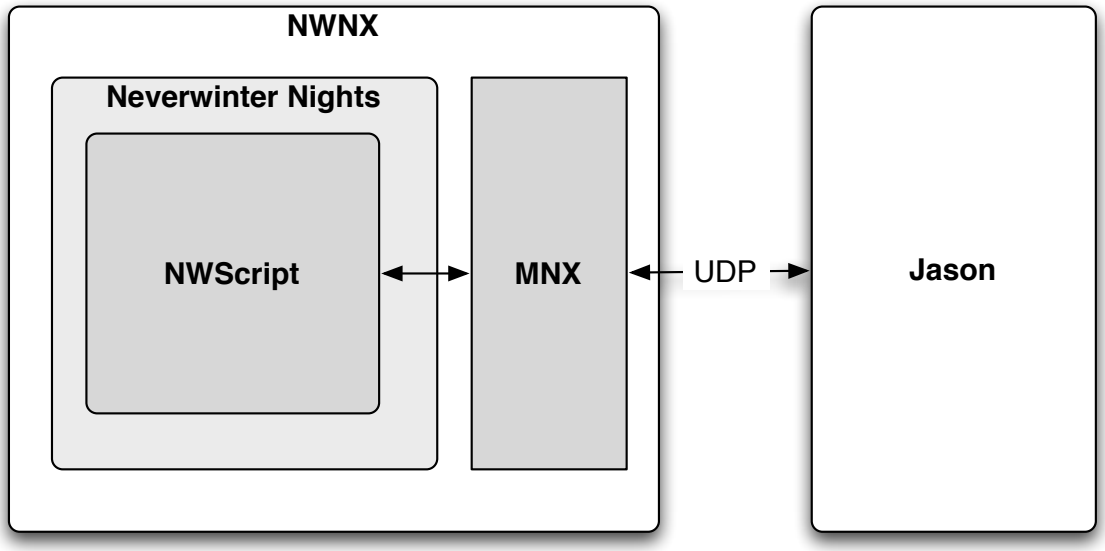

FIGURE 6.2: Integration with *Neverwinter Nights*.

every 6 seconds containing the agent's current position and health, so for an agent called 'wna12' this message would be as follows:

```
wna12:status(position([4.0,34.5,19.7],"EtumCastleDistrict"),34)
```

A custom Jason Environment object then decodes this message, parses the literal, and adds it to the specified agent's percept queue (which Jason will then add to the agent's belief base on the next cycle). A set of simple Jason rules then unpack these messages into a set of beliefs matching the ontology. Some data types, such as positions, are not actually unpacked into individuals in the belief base, but instead are used directly (as literals), with appropriate Jason rules used to treat them as if they were individuals. For example, the rules in Figure 6.3 show how positions are implemented in this fashion. This approach allows us to treat positions (and other data types) as individuals at the knowledge level, while the actual implementation at the symbol level is more efficient. The decision as to whether to implement a particular concept in the ontology as a data-type or as an individual was made on a per-concept basis. Most concepts are unpacked into individuals, and data types are only used for simple concepts that are likely to have lots of unique instances.

```
// Implementation of location ontology using position data types
location_Position(position(_,_)).
location_isWithin(position(_,Region),Region).
location_hasX(position([X,_,_],_),X).
location_hasY(position([_,Y,_],_),Y).
location_hasZ(position([_,_,Z],_),Z).
```

FIGURE 6.3: Implementation of positions as a datatype in Jason.

Percepts and Event Messages

The full list of messages sent from the environment are as follows:

status(Position,Health) The agent's current position and health.

blocked(Object) The agent has collided with the given Object and so has stopped moving.

vanished(ID) A previously seen creature has now disappeared from view.

creature(ID,Name,Species,Gender,IsPlayer) Sent when an agent first encounters a character (either a player or a NPC) in the game. The ID field is a unique identifier for this creature, Name is their name (a string), Species is the species ('race'), Gender is 'male', 'female', or 'neuter', and IsPlayer is a Boolean indicating whether the creature is controlled by a player or not.

observation(ID,Age,Health,Classes,Position,Time) Sent each time a player encounters a creature. This observation event describes the current values of any fluents associated with that character: their current age, health, character classes, and the position and time at which they were observed. The Classes field is a list of literals of the form 'classname(level)'. Note that we currently only send observation events for creatures and not other existents.

death(ID,Position,Time) Event indicating the death of a creature (indicated by the ID field).

levelup(ID,Class,Position,Time) Indicates that a character has achieved a new level in some character class. The Class field contains the class and new level achieved.

performs(Actor,Action,Position,Time) Indicates that a character performed an action.

The values for the Action field are as follows:

attack(Target,Weapons)

kill(Target,Weapons) Sent when a creature attacks another creature. The Target field is the identifier of the target creature, and the Weapons field is a (possibly empty) list of weapons used. Weapons are described only as simple data types of the form 'objecttype(Name)', for example `sword("Magic Sword")`.

damage(Target,Amount,DamageTypes) This is typically sent after an attack event to indicate how much damage was done to the victim. DamageTypes is a list of symbols indicating the type of damage inflicted (e.g., 'electrical', 'fire', 'magical', etc.).

say(Message)

tell(Audience,Message) Indicates that a creature said something, either to anyone or to a specific character.

rest The creature rested.

pickpocket(Target,Object) Indicates that a creature picked somebody's pocket, taking Object (which is described as for weapons).

travel(Destination) The creature moved to the given destination (a position).

castspell(Target,Spell) The creature cast the given spell at the given target. The spell is a simple symbol identifying the spell. All spells are known in advance and encoded into the ontology for NWN.

These messages are serialisations of elements of the ontology, as adapted for *Neverwinter Nights*. A more standardised interface could use the OWL XML serialisation format for these messages, but this was not implemented due to the expected overhead it would introduce, both in the size of the messages, and in the more complex parsing needed to decode messages for the agents. The protocol also avoids sending messages for objects and existents that can be described ahead of time by analysing the module. The format of

Neverwinter Nights modules is described in detail by Bioware[2], and this was used to automatically extract details of creatures, regions, objects, and spells that are present in the game and to populate each agent's belief base at start-up. The same technique was also used to precompute shortest paths between regions to allow fast path finding in the game: an agent can lookup a route between two regions in a database table and then relies on the in-game route finding for local navigation within a region. There are slightly less than 500 regions in the game module used for testing, resulting in 105,460 routes in the table (the region map is not completely connected).

Environment Actions

The actions that agents themselves can perform in the environment are as follows:

spawn(Position) Called initially to spawn the agent's avatar in the environment at the given position.

destroy Called to destroy the avatar.

explore Called to cause the avatar to randomly explore the current region.

follow(ID) Causes the agent's avatar to follow the given creature.

travel(Destination) Travel to the given destination position.

say(Msg) Speak a simple text string.

These are all of the basic actions that the agents are able to perform, and higher-level behaviours are built on top of these, as described in the remaining sections in this chapter.

Player Interaction

Embodied agents also support direct interaction with participants through *Neverwinter Nights*'s built-in converstion abilities. A player in the game can approach an embodied agent and engage them in conversation, bringing up a simple menu-driven conversation system, as for other NPCs. The currently implemented conversation system for embodied agents presents the following choices to the user:

[2]See `http://nwn.bioware.com/developers/`

Follow me! This choice indicates that the player is about to do something interesting and wants the event to be reported on. In response to this event a special **follow(ID)** message is sent to the agent, indicating the ID of the player making the request.

Stop following me. This choice can be used by a player to indicate that they do not want the agent to follow them or report on their current actions. A **nofollow(ID)** message is sent to the agent, and to any other agents in the near vicinity.

Who are you? This choice leads the agent to explain what it is and how the player can interact with it. The response is a stock help message, informing the player of the website. No message is sent to the agent.

What's going on around here? This results in a **present(ID,Position,Time)** message being sent to the agent, requesting that it present a report of recent activity in this area. The agent will typically respond by querying it's belief base for recent interesting events in this area that it has directly witnessed, and then presenting any matching report in-game (using the **say** action). If nothing interesting has happened recently, or the agent hasn't seen anything, then the agent responds with a stock message.

Players can also interact more bluntly with an agent by attacking it. The agent will respond in the same way as if it had been told not to follow that player. The agents are currently marked as immortal in the game, and so cannot actually die. This reaction to damage merely provides a convenient way for a player to indicate that they do not wish to be observed without having to engage an agent in conversation, for instance if the player is currently involved in combat.

6.4 Capabilities and Modules

Each agent in the witness-narrator framework provides a variety of different *capabilities*. A "capability" is a description of some functionality that an agent offers, such as *reporting*, *editing* or *presenting* reports. Agents communicate their capability descriptions to each other to facilitate team formation in response to broadcast focus goals. A capability is described in the system by a ground atomic formula. Each capability can be specialised by providing parameters that specify particular details. For instance, an agent may advertise that it has a capability to report from a particular region of the environment or to publish

to a particular IRC chatroom. Capabilities are used to decide which agents will perform which *roles* in a particular agent team.

Capabilities are implemented as independent components, or *modules*, which can be plugged together in various configurations to create an agent. The details of how these capability-specific modules are plugged together and interact with each other is described in the next section. In this section we describe the three basic capabilities provided by agents in the witness-narrator agent framework: reporting, editing and presenting reports.

6.4.1 Tasks and Deliberation

The purpose of the deliberation module is to decide which tasks the agent should adopt, and then to decide how best to accomplish those tasks. For simplicity, the agents are limited to only adopting a single task at a time. The agents are also not proactive: they only adopt tasks that they can start working on immediately, and do not schedule or plan for future goals or tasks. Indeed, deliberation is also kept to a minimum, resulting in agents that are largely reactive in operation, although keeping quite a large amount of information about the current and past world states for the purpose of narrative generation. The module supports just two basic interfaces: proposing a new task (objective), and dropping a task for some reason. The deliberative module implements some simple rules to determine whether a new objective should be adopted or not. If the task is adopted then it is recorded and adopted as a new intention using the underlying Jason BDI architecture. Otherwise, the task is simply silently ignored. The source of the task (which can be either the agent itself, the environment, or another agent) is used to determine the priority of the task. For instance, requests from players within the environment will appear to come from the environment itself (due to an implementation detail), therefore requests via the environment are given maximum priority (so as to avoid ignoring direct requests from players), whereas requests from other agents are given lower priority. These priorities are hard-coded in the rules of the deliberative module and cannot currently be adjusted. The rules as implemented ensure that requests from players are always immediately handled, whereas requests from other agents (for instance, team formation requests) are only adopted if the agent has no other more pressing task to accomplish. Tasks can also be dropped, either because it has been accomplished, or is no longer relevant. The deliberative module then simply removes the task so that it does not prevent other tasks from

subsequently being adopted. When teamwork is enabled, this event will also trigger co-operative rules to ensure that other members of a team are notified when an agent drops a task it was assigned.

6.4.2 Embodied Agent Capabilities

A single module is responsible for dealing with capabilities for agents that are embodied in the environment (witness-narrator agents). This module deals with movement, collision detection, and other low-level capabilities. The module is organised much like classic three-layer agent architectures. The top-most layer of this module deals with route planning: deciding where the agent will be and how to get there. This layer is driven by the deliberation layer, which calls on the module to plan a route when deciding whether to adopt a goal or to join a team. In the current implementation the agents actually perform no dynamic route planning. Instead, a large static table of routes between all connected regions of the environment was precalculated. This information is then stored in a database which the agents can query directly to determine the fastest route between two regions, and to retrieve a total cost estimate for that route (based on the straight-line distance between connecting doors). This results in a simple (near) constant-time lookup for route planning between regions. Within a region the agents rely on the existing local route planning algorithm implemented in *Neverwinter Nights*.

Once a route plan has been adopted, it is passed to a simple behaviour sequencing layer that takes care of the actual route following. This layer determines what behaviours should be active at what times in order to carry out the plan. The behaviours themselves, which form the lowest layer of the module, consist of collections of simple rules that interact directly with the environment (proposing actions and detecting obstacles) to achieve a simple task. The behaviours that have been implemented are as follows:

- **Explore**: the agent wanders around the environment in search of events. This is the default behaviour if there are no more specific tasks to accomplish, and is implemented as a random walk within the current area of the environment.

- **Follow**: the agent tracks a particular player, in the expectation that they will do something interesting. It is assumed that interesting events typically occur around players, so it is useful to be able to follow a player.

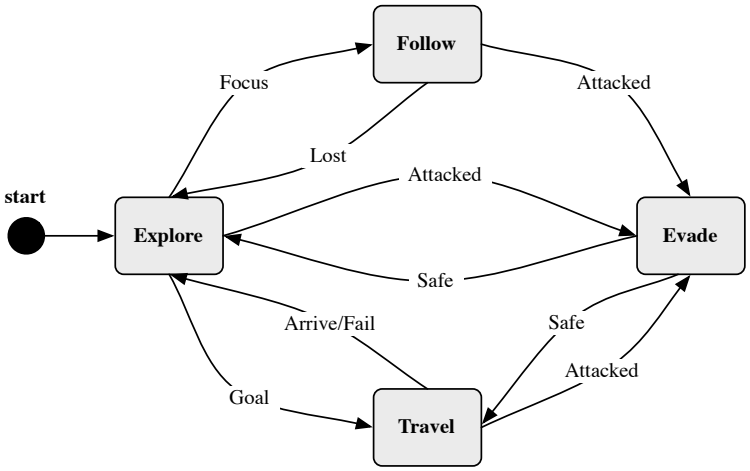

FIGURE 6.4: Reporter event finding behaviours and state machine.

- **Travel**: the agent navigates to a particular location, avoiding obstacles.
- **Evade**: if the agent is attacked or told to go away, it should take steps to avoid the participant.

Only a single behaviour is active at a time, and the scheduler uses a simple finite state machine, depicted in Figure 6.4.2, to determine the current active behaviour. The transitions between behaviour states are triggered by external events and goals. For instance, if the agent is attacked, it enters the *evade* state and flees from the attacker. Once the agent is safe again, then it reverts to the default *explore* state. The *travel* and *follow* states are entered according to the plan being executed. In addition, the scheduler can be interrupted and returned to the start state if a new plan is to be executed.

Each embodied agent is also provided with an *avatar* which is the in-world representation of the agent. The avatar is provided by the virtual environment (in this case, the *Neverwinter Nights* game environment), and is controlled by the witness-narrator agent sending commands to the game engine to perform actions on behalf of the agent, and to retrieve information about what the agent can currently sense of the environment. The avatar of the agents appears as a small 'gnome' character in bright purple and green clothing. This ensures that the agents are easily recogniseable (no other creatures or players have the same appearance), and also their small stature helps to minimise the interference caused by their presence (i.e., they are less likely to obscure the view or get in the way). Figure 6.5 shows a screen-shot of a witness-narrator agent in the game environment.

FIGURE 6.5: Screenshot of *Neverwinter Nights* showing witness-narrator agent.

6.4.3 Reporting

The reporting module is responsible for observing and recognising low-level events and actions as they occur. The task of reporting involves two related tasks:

1. Finding people and places that are likely to cause interesting events;
2. Recognising and reporting on events that actually occur.

The first of these tasks is primarily goal-directed: there are a number of focus goals describing what sorts of events to report on, and the task is to determine *where* to go (planning) in order to observe these types of events, and *how* to get there. The second main task, that of actually reporting on the events observed, is primarily data-directed: the agent must filter and process the stream of percepts coming from the environment in order to recognise and report on events. The goals for the event finding task come from two sources: (i) from presenter agents, and (ii) from the event recognition task which may spot interesting events in need of further investigation. Conversely, the event recognition module depends on the movement module in order to observe interesting events. This leads to a loosely coupled architecture with two independent modules. Coordination between the two modules is done through new focus goals communicated to the deliberative layer, and by perception of the environment.

The reporting module constantly monitors incoming percepts from the environment and attempts to classify them according to the low-level event ontology that is mostly specific to a particular environment. Once an event has been recognised it is then matched against active focus goals to see if it is of interest. If so, then a report is formed from the event and immediately dispatched to interested agents, otherwise it is discarded. The pseudo-code is shown in Algorithm 6.1.

6.4.4 Editing

The editing module is responsible for taking low-level reports from reporter agents and combining them into higher-level reports and story structures by matching them against the component events described in the event ontology. The editor module also acts like a buffer between the mostly data-driven reporting agents, and the interval-based presenting agents. This is achieved by buffering up incoming event reports and performing ongoing incremental activity recognition in a manner similar to that performed for classical plan

Algorithm 6.1 Reporting module.

```
REPORT(percept)
    input:   percept, an event percept.
    state:   G, the current set of focus goals.
1   for g ∈ G
2       do if matches-focus(percept, g)
3          then e ← get-editor-for(g)
4               SEND-TO(e, form-report(g, percept))


matches-focus(e, g)
     input:   e, an event
              g, a focus goal.
 1   r ← focus-region(g)
 2   t ← focus-time(g)
 3   c ← focus-class(g)
 4   if ¬ within(location-of(e), r)
 5      then return FALSE
 6   if ¬ during(time-of(e), t)
 7      then return FALSE
 8   if is-a(e, c)
 9      then return TRUE
10   elseif has-component(e, c)
11      then return TRUE
12   else return FALSE
```

recognition described in Section 2.4.1 (page 37). When the interval for a focus goal expires, the editor module checks to see if it has any event reports matching the conditions specified in the focus goal and then sends a description of those events to any presenter agents registered for the associated team. Events are only sent if the temporal interval of the event overlaps the temporal interval of the focus goal. For instance, if the focus goal specifies an interval of one hour between reports, then the editing module will only send event descriptions that have an endsAt value within the last hour. This overall process achieves a number of objectives. Firstly, the interval-based batch reporting ensures that presenter agents are not swamped with lots of low-level reports and partial event descriptions. On the other hand, it also ensures that a report will eventually be produced, within a reasonable time of the event actually occurring.

In Kautz's work (see Section 2.4.1), observations are compiled into *explanation graphs* that compactly encode the different possible interpretations of an event. A simpler approach was adopted in the implementation of the editing module. Instead of handling disjunctive conclusions explicitly, we instead only use forward-chaining rules where the conclusions follow deductively from the observations. For example, we can conclude that some Combat event is occurring from a single occurrence of an Attack action, but we could not conclude whether this was a Battle or a Duel and so forth. Instead, such judgements are delayed until a final report has to be generated to be sent to a presenter. At this point, backwards-chaining rules in the belief base (derived from the ontology) are used to determine which of the interpretations is correct (if this can be determined) at that point in time. Effectively, in this approach we decide ahead of time how to uniquely classify each possible observation against the ontology, and then later perform a more fine-grained distinction between possible interpretations. See Algorithm 6.2 for a pseudo-code description of this process. The REVEIVEREPORT procedure is called whenever a new report arrives from a reporter. The report is split into individual events and then each event is compared against the current state stored in the editor agent's belief base. If the event matches a component of an existing event (e.g., is a step in a current quest) that has not already been filled (or can be filled multiple times, as in the case with combat sub-events of battles) then the event is asserted as a component of that event. The matches function incorporates checks against the ontology to ensure that the class of event is correct, but also checks that the time and location of the event are suitable. This currently involves checking that the region in which the event occurs is identical, and that it occurs within a small time of the end of the larger event. If these conditions are satisfied, then the ASSERTCOMPONENT procedure is called. This asserts that the new event fills the given sub-event role in the larger event and also updates the start and end times of the larger event to take into account the newer event. If the new event does not match any existing larger event then it is asserted as a new event in it's own right. The ASSERTEVENT procedure examines the ontology to determine if the event indicates a larger event type. For instance, we know from the subsumption hierarchy that an Attack action indicates some form of Combat event. The GENERATEREPORT procedure is then called when the time interval for a focus goal has expired. At this point the editor tries to classify all existing events against the ontology using backward-chaining reasoning (the classify function). This function uses more complete reasoning against the ontology to make finer distinctions between different classes of events using all currently

available evidence. Each such classified event is then compared against the focus goal to see if it matches. If so, then a fresh report is formed and sent to the presenter agent for this focus goal.

This approach has both limitations and advantages. On the positive side, the implementation is straight-forward and simple to understand. It can also handle revising the interpretation over time: an earlier report might describe a battle as a duel, whereas at a later time (when more combatants have joined in) it might be described as a skirmish or a battle. This is handled automatically by the PROLOG-style ontology rules, and doesn't need any complex reorganising of data structures. However, the approach does require that any ambiguous events in the ontology (i.e., events which share a common sub-event classification) have some common parent concept which can be used to classify incoming events until a more fine-grained distinction can be made. Additionally, the approach cannot handle some more complex plan recognition problems in which different sequences of events might lead to entirely different interpretations. In practice, for the domain we have focussed on, these drawbacks have not been much of a problem. The most common type of event in our test environment, by far, is that of combat, which can be handled by classifying all aggressive actions and events as sub-events of some general combat event, and then later reclassifying that event to a more specific sub-class using the ontology as described. The other most common complex event type is that of quests, which in general can have arbitrarily complex plan-like structures. However, in most computer role-playing games, and in particular in *Neverwinter Nights*, quests usually involve distinct and unique sequences of actions to perform, and so can be directly classified without ambiguity.

6.4.5 Presenting

The presenting module is implemented as shown in Figure 6.6 and Algorithm 6.3. The high-level reports produced as a result of editing are declarative structures describing a particular event. Each structure contains fields describing what happened, where, when, and who was involved. Each event description may also have a number of sub-events which describe in finer detail how the event unfolded. In general then, an event description forms a tree structure, with leaves representing the lowest-level details of what happened (for instance, movements and actions of individuals) while the root of the tree represents a high-level overview of the event.

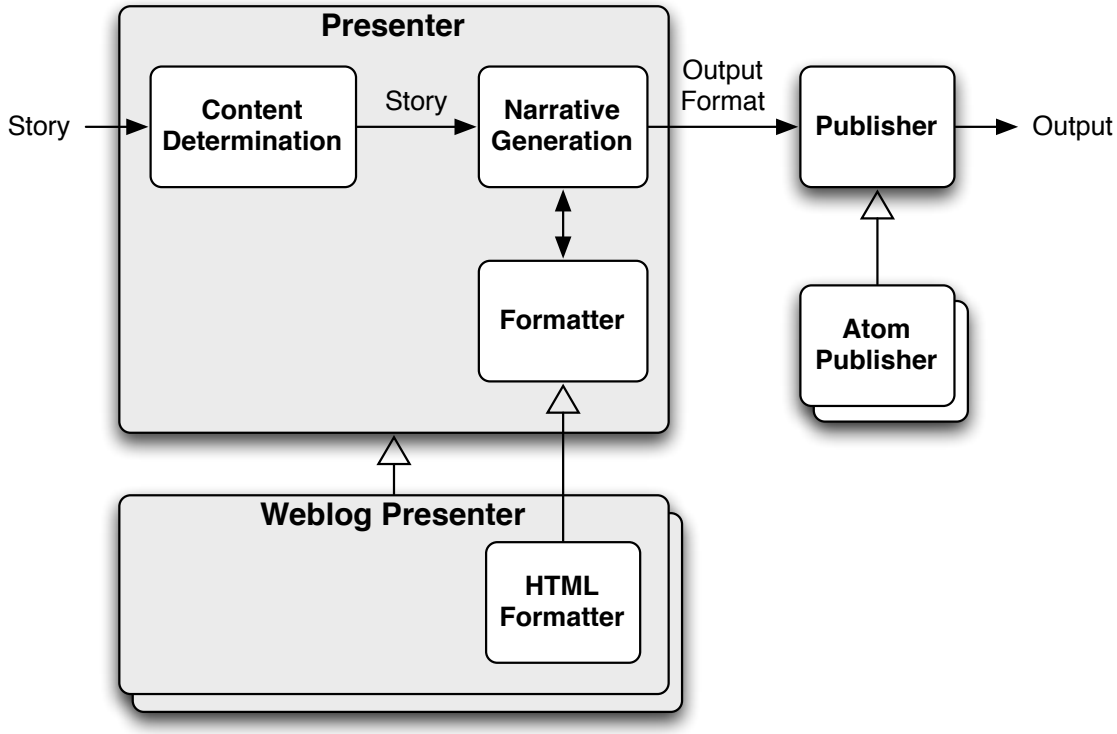

FIGURE 6.6: Presenter module workflow and main components.

Algorithm 6.2 Editing module.

RECEIVEREPORT($report$)

```
    input:   report, an event report from a reporter.
1   for event ∈ report
2       do if ∃e, s. has-component(e, s) ∧ matches(s, event)
3             then ASSERTCOMPONENT(e, s, event)
4             else ASSERTEVENT(event)
```

GENERATEREPORT(g)

```
    input:   g, a focus goal.
1   t ← time-since-last-report(g)
2   for e ∈ classify(events-since(t))
3       do
4          if matches-focus(e, g)
5             then p ← get-presenter-for(g)
6                  SEND-TO(p, form-report(g, e))
```

In the content determination phase, the presenting capability first matches received reports against the focus goal for which a narrative is being created. Once events have been selected, the level of detail appropriate for the narrative is determined. For example, output to a weblog may involve using the entire event structure in the report, whereas a SMS message will require much less detail. The level of detail is specified as a simple limit in the depth to which an event description is traversed to extract information. Narrative generation currently makes use of a relatively simple text template scheme. The tree of event descriptions is traversed to successive depths (up to the depth limit) and each level is matched against a number of rules which extract relevant information and plug it into pre-designed text templates. To avoid overly repetitive text, some variation is allowed in choosing words to describe the entities that are referred to in the event description. After this narrative prose has been constructed, a final phase adds appropriate formatting for a particular output medium. For instance, one output formatter wraps the prose in an Atom XML news entry description which can then be published to a variety of web publishing platforms using a standard Atom publishing API. The current implementation uses the popular WordPress[3] web publishing platform for this purpose, which provides

[3] http://wordpress.org/

Algorithm 6.3 Presenting module.

PRESENTREPORT(*report*)

input: *report*, an event report from a editor.

```
1 r ← determine-content(report)
2 title ← generate-title(r)
3 cats ← generate-categories(r)
4 body ← format(generate-narrative(r))
5 PUBLISH(title, cats, body)
```

a flexible HTML web page for reports, as well as automatically providing RSS and Atom news feeds.

Witness-narrator agents are also capable of presenting simple one-sentence summary reports of recent activity in-game. This is done by reusing the AtomPub presenter module but discarding the main body of the report and only relating the one-sentence headline to the player that requested a report. At present this capability is limited to only reporting on the single most recent event that has occurred in the region in which the request was made. The reason for this limitation is simply that the in-game speech interface is only useable for short speech. Longer conversations can be encoded using the built-in menu-based conversation system, but such conversations have to be statically encoded and so could not be used for dynamic narration of recent events.

The presenting capability also forms the interface for the generation of focus goals in response to user requests and for rating the resulting narrative presentation. How users communicate requests is specific to each output medium. For instance, a weblog presenter may provide a form where members of the audience can submit details of types of events they are interested in receiving reports about. Similarly, participants can approach a witness-narrator agent and engage it in conversation. A simple menu-based dialogue is conducted in which the user can request reports of, e.g., particular (past) events, or receive the agent's personal take on recent events that it has witnessed directly. The current WordPress based front end allows users to rate individual reports for interest and accuracy using a simple form at the bottom of each report. This information is used only for evaluation purposes at present, and is not fed back in to the system to influence future focus goals. In game witness-narrator agents can generate new focus goals in response to direct requests from players, but these are limited to 'follow me' type requests at present.

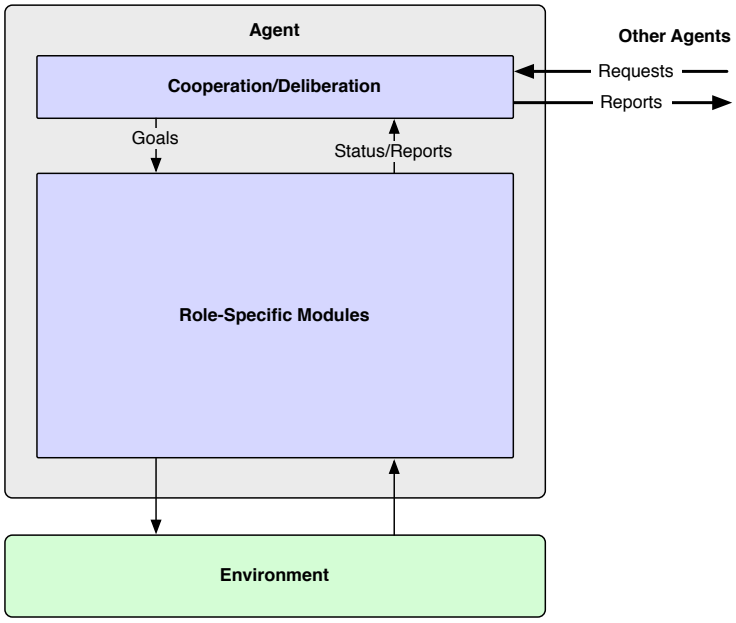

FIGURE 6.7: General agent architecture.

6.5 Agent Architecture

The narrative agents framework employs both data-driven and goal-driven behaviour. Embodied agents reporting on events are mostly data-driven. They detect events occurring in the environment and transmit this low-level information directly to editors. Editor agents then collect these reports and combine them into higher-level event reports ready for transmission to presenter agents at intervals specified in their focus goals. While an editor is waiting to transmit a report description it can collect new information and constantly update and revise the drafts of reports of current events. This allows for a process of revision and updating to occur while the interval-based goal-directed behaviour ensures that a best-effort report is produced on a timely basis. Without this goal-oriented behaviour it would be difficult in general to detect when an event has actually finished, and so we have the danger of editor agents constantly buffering up new information about reports and never being able to tell when all the facts are in.

The basic architecture of the narrative agents is shown in Figure 6.7, and is modelled loosely on common three-layer architectures [46]. The top-most layer of the architecture deals with cooperative and deliberative processes. Only this highest layer is generic

to every agent, and implements the coordination strategies used by the organisation as a whole. Once this layer has decided to adopt a goal and selected a plan for how to achieve it, it passes this information down to the lower layers of the architecture. These lower layers are not generically implemented, but are instead divided into a number of capability-specific modules. There are four such modules in the framework as a whole:

- A *reporter* module takes care of event detection and recognition. This module can spot events from low-level perception events and record important details as events happen.
- An *editor* module that is capable of combining reports from multiple sources, assessing accuracy, and carrying out higher level event recognition.
- A *presenter* module that is responsible for communicating generated narratives to particular output medium and target audience.
- A *movement* module for agents that are embedded in an environment. This module takes care of navigating reporters into relevant positions, and following players, etc..

The movement module implements the other two layers found in a traditional three-layer architecture: a *sequencer* layer that executes the plan created by the deliberative layer, and a *behaviour* layer that contains a number of low-level reactive behaviours that work as tight feedback loops with the environment. The other capability-specific modules do not perform actions in the environment and so have a much simpler architecture. The individual modules and how they are combined in particular types of agents are described in the following sections.

The modules are implemented as collections of Jason plans and rules. For example, the reporting module contains a set of plans and rules for detecting and recognising events occurring in the game world. These modules are combined in various configurations to create individual agents. As stated in the introduction, there are two basic agent configurations currently used in the witness-narrator framework: embodied witness-narrator agents (WNAs), shown in Figure 6.8, and non-embodied commentator agents (CAs), shown in Figure 6.9.

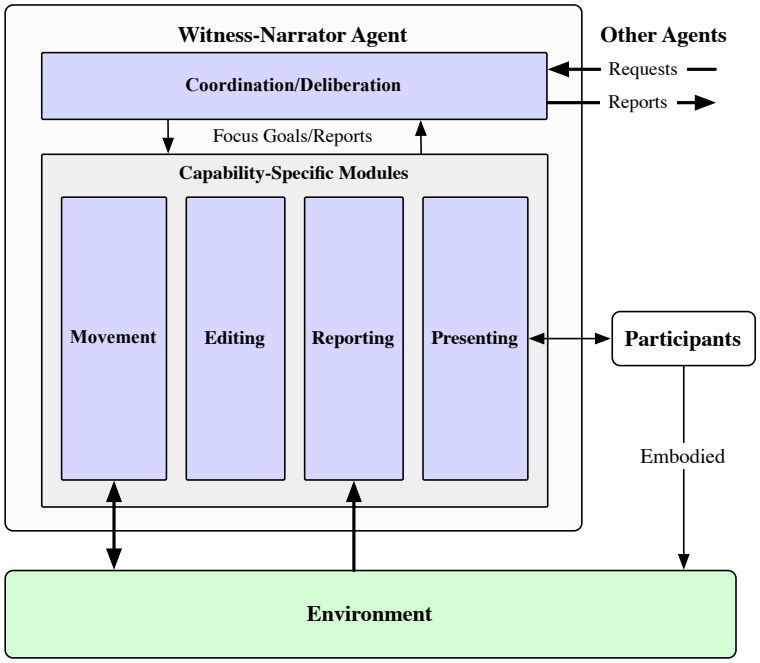

FIGURE 6.8: Witness-Narrator Agent architecture.

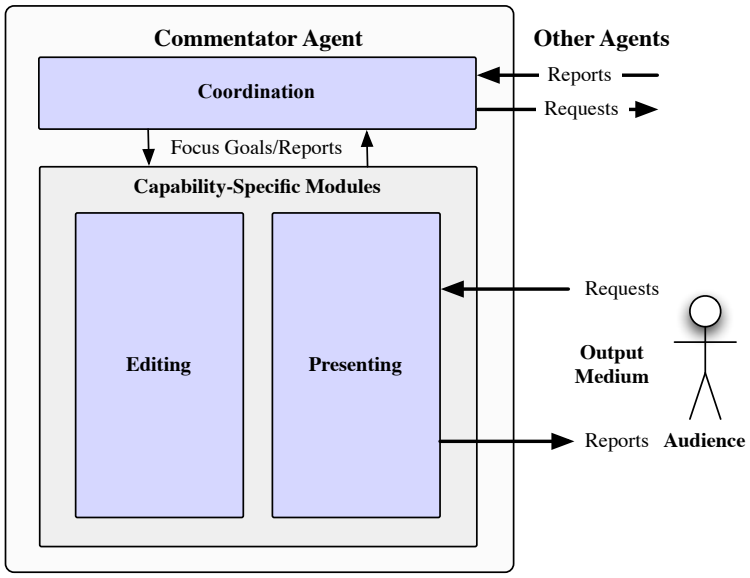

FIGURE 6.9: External Commentator Agent architecture.

6.6 Multi-Agent Cooperation

In order to generate compelling narrative from a large-scale environment it is necessary to ensure adequate coverage of events occurring within all areas of the game world, and to adapt to changing conditions to ensure that each event is covered by a sufficient number of agents. This is the function of the coordination layer in the architecture.

6.6.1 Focus Goals

As described in the Chapter 4, the primary mechanism for coordination of the multi-agent system is through the use of focus goals. A focus goal is a simple specification of the types of events that an audience is currently interested in, and the time and location at which that goal should be active. Each focus goal consists of a named concept class from the ontology (e.g., Battle for battles), together with a possible region to restrict the focus, and a numeric priority to help agents decide which goals are most important. Currently the priority is set to 50 for the initial goals of the system, 75 for any battles that the agents witness occurring, and 100 for direct requests from a player. This simple set-up ensures decent coverage of the basic events that we are most interested in. A more sophisticated system would use audience numbers or some other metric to calculate an appropriate priority for a focus goal. For instance, currently a direct request from a player always takes priority over focus goals from presenter agents, even though the audience that the presenter represents may consist of a large number of players.

In addition to specifying the location and type of events that should form a focus for the MAS, a focus goal also specifies the time at which such a goal should be active. This is specified as an interval stating how often the presenter agent wants to receive updates for this particular goal. There are two forms such an interval can take: either immediate or every(n), where n is an integer interval in minutes. The meaning of immediate is that the editor should forward on any matching reports to interested presenters immediately, as they occur, whereas the second form sets up a periodic timer task to produce a report every n minutes to send to the presenter. This is achieved using Jason's built-in `.at` internal action, which can be used to schedule a delayed intention. In this way the system can support both data-driven goals, which are useful for reporting breaking news (this is used in the test environment to provide notifications that battles have broken out, and also to report character achievements, such as levelling-up), and timed, goal-directed reports. The

latter type of reports are useful for long-running events, such as large battles, in which case it can sometimes be difficult for the agents to detect when the event is over. By using a timer, the editor can report on the battle at a fixed time, and then will simply stop reporting on it when no more events have occurred since the last report.

6.6.2 Organisation

We can identify a number of *roles* within the reporting agent team. A role is simply a collection of behaviours that a particular agent in a team is responsible for. Typically, a role will be associated with a collection of rules and plans in the Jason implementation. These roles are as follows:

1. **Reporter**: Embedded within the environment and reponsible for gathering information about events as they occur.

2. **Editor**: Responsible for aggregating reports from multiple reporters, checking them for consistency, and combining them into higher-level reports.

3. **Presenter**: Responsible for relating reports to an audience via some output medium, such as an HTML webpage, or an IRC chat session.

These roles correspond to the three agent types in Dan Fielding's reporting agents framework. By separating out the notion of a role from that of an agent we can explore different configurations for assigning roles to agents. Firstly, we can choose to assign roles either statically, so that an agent is permanently assigned a particular role, or dynamically, changing role allocations as required. Secondly, we can allow a single agent to take on multiple roles (such as reporting and editing), or we can require that each agent is dedicated to a single role. The current framework uses two agent configurations. A witness-narrator agent is capable of performing all three roles, but is mostly restricted to reporting and editing duties. The presenting capability is used only when a player directly requests an update on recent events from a WN agent. External commentator agents are primarily presenters, but could also function as editors, and so are equipped with just the editor and presenter capabilities. The test system has just a single external commentator agent, responsible for publishing reports to the WordPress publishing application. Other configurations are of course possible, for instance a team of external commentators could be

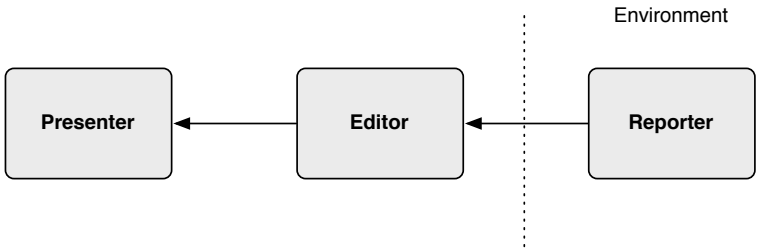

FIGURE 6.10: Basic workflow organisation.

used, each with responsibility for different types of events, or for covering different regions of the environment. This would allow the workload for producing reports to be distributed over several agents (potentially on different machines) and allow much more effort to be put into producing polished reports. At present the single commentator agent only produces quite simple stories and does not attempt to perform much detailed natural language generation, instead adopting a simple text templating system. This approach is reasonable for the test environment, but tends to produce somewhat repetitive narratives.

Each focus goal requires that some agent team commits to it. A goal cannot be fulfilled if no agents attend to it. An agent may belong to several different teams, working on separate goals, or may be constrained to belong to a single team working on a single goal. In addition, a single team may take on multiple goals.

6.6.3 Coordination

The basic workflow of the framework is shown in figure 6.10. Focus goals are generated by presenter agents responding to audience desires[4], and these are filtered down through the framework to editor and reporter agents, which use these goals to determine how best to cover the environment. There are a number of ways in which this workflow could be achieved. In the reporting agents framework, described in Section 2.5.2 (page 45), the various agents in the system were linked together statically. Typically, there was a single well-known editor agent, and all reporters and presenters connected to this agent, shown in figure 6.6.3. While this system was simple and effective in the relatively small environments of *Unreal Tournament*, more flexibility is desireable in larger environments involving moderately sized teams of reporters, editors and presenters.

[4]Focus goals can also be generated spontaneously by a reporter in response to observed events, but these are always compatible with an existing presenter-generated focus goal

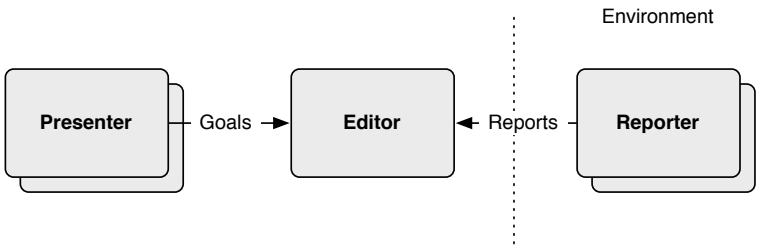

FIGURE 6.11: Coordination through a single well-known editor agent.

In [121], cooperative problem solving is broken down into four distinct phases:

1. *Recognition* of the potential for cooperation with other agents on a particular task.
2. *Team formation*, where an agent recruits other agents to help with the task. This involves collective agreement to work on the task, but does not (yet) imply a commitment to any specific means of achieving the objective.
3. *Plan formation*, where agents collectively agree the means by which the objective should be achieved, and how sub-tasks should be allocated amongst the team members.
4. *Execution* of the agreed plan, possibly involving mechanisms to reconsider commitments or reassign roles as the situation changes.

In the witness-narrator agents framework recognition and initiation of team formation is performed by presenter agents. These agents generate focus goals in response to the interests of their target audiences. Each focus goal requires some team to commit to it, and so generation of a focus goal leads immediately to a team formation phase. For the most part, once a team has been assigned, the process of plan formation and execution is the same for each team. Reporter agents must move themselves into positions where they can observe relevant events, whereas editor and presenter agents merely have to wait for reports to arrive and then process them. The agent that originates the focus goal is responsible for recruiting other agents, and for deciding what roles each should perform. This includes deciding which areas each reporting agent should cover, and how many reporters are needed to cover the region specified in the focus goal.

A focus goal requires some non-empty set of agents to commit to it, and within that team at least one agent must be commited to each of the major roles: presenter, editor,

and reporter. Without a presenter, there is no point producing reports, and without a reporter there will be no reports at all. The editor role is perhaps less well justified, but it justifies itself on three grounds:

1. Firstly, editor agents can combine information from multiple sources, checking for accuracy and recognising larger-scale events that may not be discernable to individual reporters.

2. Editor agents can act as a buffer between data-driven reporters (that produce reports as events occur), and demand-driven presenters that require best-effort reports at regular intervals.

3. Editor agents are well placed to act as overall coordinators, directing the other reporter agents in the team, should such a strategy be employed.

For these reasons we require that all three roles be present.

6.6.4 Teamwork

There are a number of options for team formation. In Fielding's approach, there were no dynamically generated focus goals, and a simple fixed team structure was employed. Reporter agents could also be assigned 'roles' which were particular areas to watch, such as a team's flag base, and an editor agent would reassign agents to cover any gaps. However, all agents belonged to a single team and worked towards the same goals. With the introduction of focus goals and a larger environment, it becomes more important to divide responsibility for different goals among sub-teams of agents. There are a number of ways in which this could be done:

1. By employing a fixed hierarchical structure, with different sub-teams permanently assigned to a particular sub-set of events. For instance, this could be done geographically by region of the environment, or by dividing up the ontology (e.g., having a team which covers all speeches and another that covers all combat events).

2. By employing a dynamic hierarchical structure in which a coordinator agent is assigned for each focus goal and this coordinator then allocates a dedicated sub-team to work on the task.

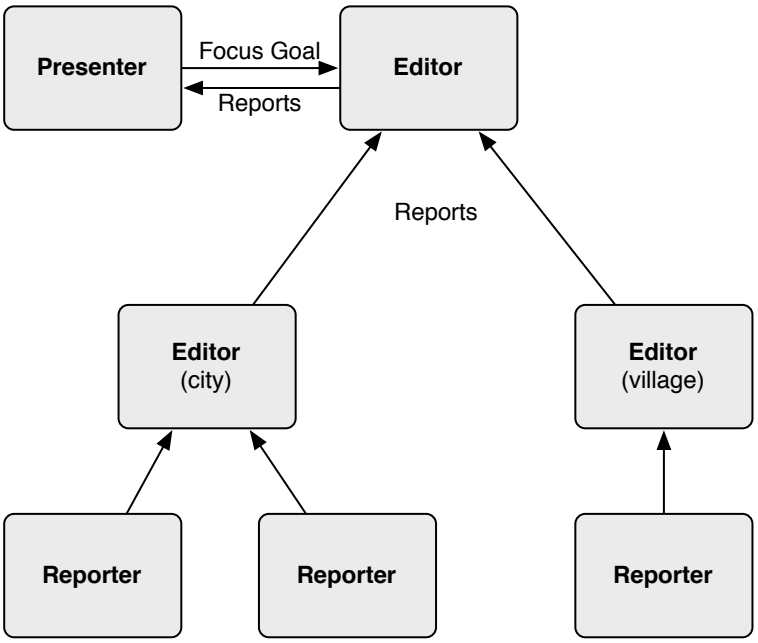

FIGURE 6.12: Fixed team hierarchy, based on region.

3. By negotiation where individual agents bid for roles within a team. This would be a Contract Net [109] style system.

4. By negotiation where agents form teams between themselves and collectively bid for the entire focus goal.

The hierarchical options are shown in figure 6.6.4, while the Contract Net-style option is shown in figure 6.6.4. Each has advantages. The Contract Net is resistant to failures of individual agents, as these agents simply won't be available to bid. The hierarchical solutions on the other hand will fail completely if the top-most editor agent stops working. There are ways to limit the damage in a hierarchical system, such as having fail-over mechanisms whereby one agent will take over the role of a missing editor. However, these add to the complexity of the solution. The Contract Net proposal is simpler, but suffers from excessive network usage if many presenter agents are often trying to recruit team members. This can be lessened by limiting the scope of broadcast messages, but this again complicates the framework.

Initially, we will employ the simple Contract Net scheme whereby a presenter agent broadcasts an open call for agents to work on a particular focus goal. Individual

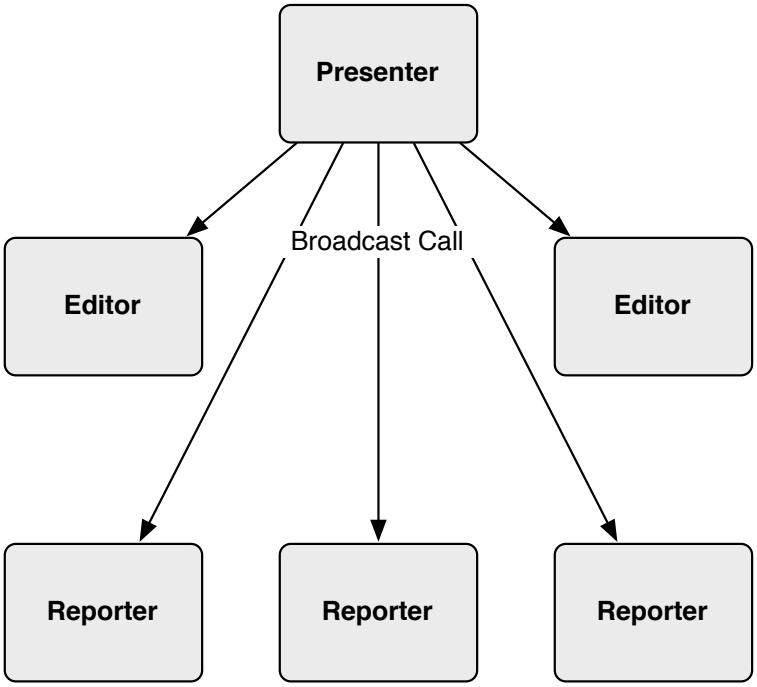

FIGURE 6.13: Broadcast call for team members in first stage of Contract Net style team formation.

agents can then respond to this call, and the presenter will select a team of agents. This is a simple scheme which should be effective for the scale of experiments we wish to perform. There will not be too many presenter agents, and the frequency with which they produce focus goals should be low enough that the relative network inefficiency of the protocol shouldn't be a problem. It should be possible to later move to a more hierarchical model where broadcasts are made only to known coordinator agents, which then form a team on behalf of the presenter (either through further limited broadcast, or by direct delegation).

Apart from focus goals generated by presenter agents, it may be possible for reporter or editor agents to also generate focus goals in response to events occurring within the environment. For instance, if a reporter spots a large-scale event occurring that it is not able to cover itself, then it may generate a new focus goal and attempt to recruit a team to cover the event. This should only happen if the event already matches some focus goal the reporter has, otherwise no presenters will be interested in it. Therefore, the sub-team will be recruited as normal and will send all reports to this reporter. The original reporter will then pass on these reports to interested agents of the original focus goal in the normal way.

6.6.5 Team Formation

A team of agents is formed to handle each broadcast focus goal which specifies future events. Each focus goal requires that some team of agents commit to it. A single team can take on multiple focus goals, but typically a new team will be created for each focus goal. Individual agents can belong to multiple teams, and teams are hierarchically structured with members of a subteam also being members of a larger team. Initially, there is a single team that includes all agents in the system and attends to some general focus goals (such as reporting on all deaths that occur anywhere in the environment).

The agent that generated the focus goal is known as the coordinator, and is responsible for recruiting agents to work towards the goal, and for ongoing coordination of the agent team. This is the case even if the originating agent is not itself able to contribute to the team. For instance, if an agent is committed to covering some event but notices another interesting event en route, then it will attempt to recruit other agents to cover the goal while still carrying on to its original destination. This ensures that all noticed events are covered (if possible) while avoiding an agent having to drop a commitment.

The initial team formation phase involves determining which agents are available to work on the goal and what capabilities they can offer. To achieve this we use a Contract Net based protocol. Firstly, the coordinator broadcasts a general call for participation, including the details of the focus goal. Each agent must then decide if it can commit to the goal and whether to make a bid to be on the team. In its bid, an agent includes a list of its capability descriptions along with the times at which it is available to work for the team.

Agents determine if they are available to work on a particular focus goal using a simple goal arbitration scheme. Each agent considers only its position within the environment (if it is embodied) over time. An embodied agent keeps track of the locations it is committed to being in and during which time intervals, and uses this information to determine if a new focus goal is compatible with its existing commitments. If an agent can attend to a focus goal at any time when that goal is active, then it will submit a proposal to join the team, including information on when it is available and what capabilities it can offer (presenting, editing, reporting). An agent may commit to as many focus goals in whatever combination of roles that it believes it can achieve.

6.6.6 Role Assignment

Once all bids have been received (or the proposal deadline is reached), the coordinator then moves to assigning *roles* to team members. To do this, it generates a set of role requirements consisting of a particular capability pattern that an agent must perform, along with an *ideal* number of agents required for that particular role. Role requirements are patterns which can be matched against capability descriptions to determine if a particular agent is suitable for a particular role.

Team formation is approached on a best-effort basis. The only hard requirement is that at least one agent must commit to each of the three role types (reporter, editor, presenter). The coordinator is responsible for initial role assignment, and also for on-going coordination of the team, such as arranging cover for agents that become unavailable, or recruiting new agents that become available over time.

The coordinator agent tries to assign agents to roles to ensure the greatest possible coverage of the focus goal (measured by time at which agents are available), up to the ideal number of agents specified in the role requirement. Once roles have been assigned, each agent is informed of its expected task by a message including the specific role information and the times at which the agent is expected to commit to the role. At this stage, each agent must recheck its commitments (in case they have changed) and can either confirm the commitment (perhaps with a slightly altered schedule) or can refuse (in which case, the coordinator will attempt to reassign the role). Once the final role assignments have been agreed, the coordinator broadcasts the information to all members of the team so that they know who is responsible for what.

CHAPTER 7

EVALUATION

7.1 Introduction

The main aim of this work has been to create a technology that can be used to increase the sense of participation in online role-playing games, and to foster a feeling amongst players that their participation contributes meaningfully to the evolution of an ongoing shared narrative experience. In order to evaluate whether we have managed to achieve this aim involves answering a number of basic questions:

- do players play the game more when witness-narrator agents are present?
- does the presence of the agents affect the way people play the game?
- are the agents disruptive to normal gameplay in any manner?
- do players and others find the reports interesting or useful?
- do players find the reports accurate?
- finally, does appearing in a generated story make the game more enjoyable?

7.2 Evaluation Outline

Due to the relative novelty of the task and the approach we have taken it was decided that these questions should be addressed in order to develop an idea of how the technology is used and what its strengths and weaknesses are. We therefore chose to evaluate the technology primarily by an extended live participation study. This involved running the agent technology in a public *Neverwinter Nights* game server for an extended period and

Machine	CPU	RAM	Role
giotto	3.2GHz	1GB	Neverwinter Nights server
uccello	3.2GHz	2GB	Jason agent server (100 agents) and MySQL server
sleepy	2.4GHz	2GB	Public WordPress server (Apache, PHP, MySQL)

TABLE 7.1: Configuration of machines during testing and evaluation.

carefully observing how players use the system in order to gain an understanding of how the technology is used and to direct future research. The data collected is mostly qualitative in nature, consisting of comments and questionnaire responses from participants, and observations about their use of the technology. The setup and results from this study are presented in section 7.7.

In addition to the live study, we also performed a quantitative evaluation of the basic technical aspects of the framework. In particular we present results on coverage achieved by the system both with and without teamwork enabled. We also provide some figures which show how the framework performs under load. These performance and coverage test were entirely automatic, using scripted bots to generate events in the environment which the witness-narrators will then report on, matching as closely as possible the conditions present in an environment using human players. The setup and results from the performance and coverage tests are presented in sections 7.4, 7.5 and 7.6.

7.3 Equipment

The tests were carried out on a cluster owned by the intelligent agents research group at Nottingham. The configuration of these machines is shown in table 7.1. Three machines from the cluster were used to host the main servers for the study. These are the public *Neverwinter Nights* server, the agents themselves (running as a single Jason instance), and a MySQL relational database management system running on the same machine, which held the persistent beliefs of the agents (such as archives of previously produced reports and information about players who have been encountered). Finally, the existing public webserver for the research group (sleepy) hosted the public web pages produced by an AtomPub commentator agent. This public server ran the WordPress publishing application (version 2.2.1) that allowed an agent to publish a report of an event using the Atom

Publishing Protocol (AtomPub).

The *Neverwinter Nights* server runs the standard Linux version of the NWN standalone server. The Neverwinter Nights Extender (NWNX) application is used to integrate the agents with the game environment. The interface between the Jason agents and the Neverwinter Nights server makes use of the MNX module which allows communication with a running game via UDP. The *Neverwinter Nights* server ran a slightly adapted version of the popular *Rhun* persistent world module. The alterations consisted of a few hundred lines of NWScript code to enable external agents to connect to the environment via UDP and to receive notifications when actions were performed in their near vicinity. Some other minor alterations were made to ensure that each area in the environment had a unique tag assigned to it in the game, to avoid any problems recognising each region. None of the changes made had any affect on the gameplay or quest elements of the module.

The agents themselves ran using a single instance of the Jason agent runtime, using the centralised infrastructure rather than the distributed SACI infrastructure. It was found that a dedicated machine was capable of running up to 100 Jason agents and the associated MySQL database, and so a distributed infrastructure was not required. The use of MySQL provided a number of advantages, most notably reducing the memory overhead required for the agents, as they store large numbers of beliefs related to previous events, and these can be stored in the database rather than kept resident in memory. The other large advantage of using a RDBMS was that in the event of a catastrophic failure of the agent system it could simply be restarted without the agents losing any belief context beyond their current goals and intentions (which would be reset).

7.4 Performance Tests

7.4.1 Method

The first round of technical testing simply measured the CPU and memory requirements of the agents in a number of configurations. The purpose of these tests is to firstly check that the implementation and the general approach taken is feasible when applied to a real-world setting, and also to gain an idea of the resources that would be required for the live evaluation study. Each test measured the performance of the framework as the number of witness-narrator agents in the environment was increased from zero up to 100 agents. The number of agents was increased in 10 agent increments and measurements taken and

averaged over a 5 minute interval. Each test measured total CPU and memory usage (as measured by the `vmstat` tool) on both the *Neverwinter Nights* server and on the Jason agents server. From these figures we then derived the mean CPU and memory usage per agent for each configuration. The tests that were run are:

P1: The first test measured the performance change as agents are added in a random spread across the environment. Each agent spawned was randomly assigned to a region of the environment. Each agent remained static once it has spawned but continued to report on any events that it observed. This test should simulate the least load conditions for the agents and allow us to measure a lower bound for the CPU and memory.

P2: The second test then measured the performance when the agents are spawned in close proximity to each other. This increased the load on that particular region of the environment, and also simulated increased load for the witness-narrator agents as they will be receiving percepts for each other agent around them, therefore simulating a very crowded environment. This test should approach an upper bound for the CPU and memory.

7.4.2 Results and Discussion

The results for experiments P1 and P2 are shown in Table 7.2 and Table 7.3 respectively. As can be seen from both tables, the impact on the *Neverwinter Nights* server of running more agents is negligible. Indeed the CPU and memory usage for the *Neverwinter Nights* server remains largely unaffected as further agents are added. The reason for this is due to the way in which the agents are integrated with the environment. Most of the extra work involved on the server to support the witness-narrator agents is performed no matter how many agents are present. This work consists mostly of intercepting events occurring within the environment and then scanning to see if any embodied agents are nearby and within line-of-sight to perceive the event. Clearly, this work is performed regardless of whether an agent actually turns out to be present or not. The cost of actually transmitting the percept via the UDP interface seems to be minimal (messages are typically only a few hundreds bytes in length). There is however a much larger difference between the lower and upper bound performance figures for the Jason agents. When the agents are more densely packed, the number of events each agent has to process increases considerably

# of Agents	NWN CPU %	NWN Memory (kB)	Jason CPU %	Jason Mem (kB)
0	46.2	148528	0	0
10	47.4	148752	0.1	28188
20	48.8	148576	0.2	34860
30	48.0	148796	0.3	41792
40	46.6	148864	0.3	48260
50	48.7	148732	0.3	53980
60	48.0	148904	0.4	57428
70	48.9	148792	0.4	63660
80	48.3	148736	0.4	64884
90	50.5	148724	0.5	66160
100	47.7	148692	0.5	67180

TABLE 7.2: Results of experiment P1: lower-bound performance.

# of Agents	NWN CPU %	NWN Memory (kB)	Jason CPU %	Jason Mem (kB)
0	46.4	148579	0	0
10	47.4	148752	9.2	37192
20	49.0	148824	18.4	41322
30	49.4	148764	26.2	48720
40	50.0	148590	34.6	52036
50	50.4	148902	39.4	56146
60	49.8	148958	45.2	62344
70	50.2	148622	49.4	66364
80	51.2	149104	53.8	71702
90	50.4	148944	55.6	73204
100	50.2	148736	59.4	77684

TABLE 7.3: Results of experiment P2: upper-bound performance.

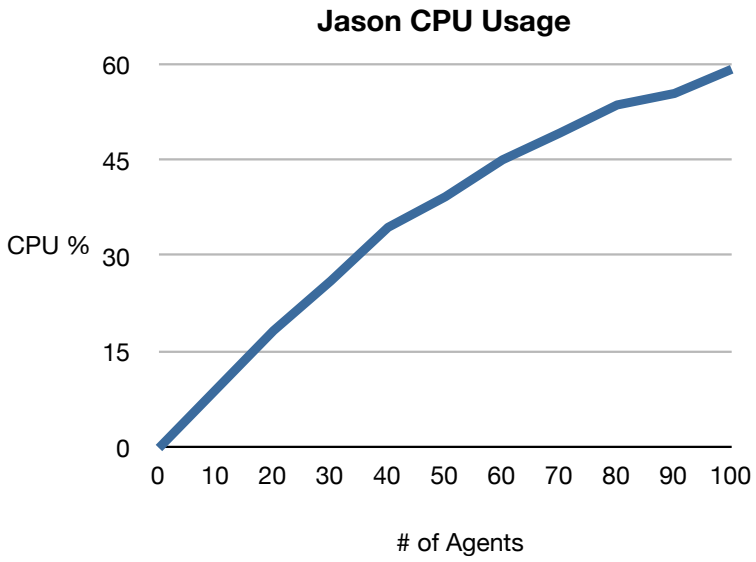

FIGURE 7.1: P2 upper-bound CPU usage for Jason agents.

as the agents receive percepts relating to each other's activities. While all of these events from other witness-narrator agents will not be considered interesting, and so not form part of a report, the witness-narrators still have to process each event and match it against the event ontology. This uses multiple reasoning cycles per observed event, resulting in a much higher CPU and memory usage. The CPU usage for the Jason agents for experiment P2 is plotted in Figure 7.1, and shows a near linear increase in CPU usage as the number of agents in the environment is increased. The memory usage also appears to increase approximately linearly with the number of witness-narrator agents in both tests.

7.5 Coverage Tests

7.5.1 Method

The purpose of the coverage tests was to determine how coverage varies with the number of witness-narrator agents, and with the ratio of participants to witness-narrators. The figures produced were used to determine how many agents to use in the live evaluation study, and how best to distribute them over the large environment. Coverage was measured by using scripted participants ("bots") to simulate human players. Each bot was programmed to produce events in the area where they were spawned. Specifically, each bot would randomly wander around the area it was spawned in and attack any creature that it encountered (except the witness-narrator agents). All actions performed by the bots were recorded by the *Neverwinter Nights* server to create a log of all the low-level events that were generated. The witness-narrator agents then fed reports to a special log presenter agent that also created a log file in the same format. These two log files were then compared to evaluate coverage (how may events were actually reported). For each of these experiments, teamworking capabilities were disabled so that agents effectively freelanced. The evaluation of the effect of teamwork on coverage is reported in the next section. Accuracy (i.e., whether the reported events are also factually correct) has not been evaluated, as for the low-level events considered in this evaluation the accuracy should always be 100%—no inferences are involved.

C1: This experiment measured the effect of increasing the number of witness-narrator agents has on the coverage. It was expected that increasing the number of agents would have an initially beneficial effect on coverage, but that this would reach a limit

# of Agents	Events Generated	Events Reported	Percentage
2	560	45	8.036 %
4	676	69	10.207%
6	744	118	15.860%
8	319	59	18.495%
10	780	118	15.128%
12	842	146	17.340%
14	517	168	32.495%
16	729	184	25.240%
18	839	201	23.957%
20	706	202	28.612%

TABLE 7.4: C1 Results

beyond which the number of reporters in a single region would no longer be beneficial. For this experiment, agents and bots were confined to a single region of the environment, and there was a fixed number of bots (10). The number of agents was varied from 1–20 in increments of 2 agents and measurements taken over 5 minute intervals.

C2: This experiment measured the coverage over a larger area as the ratio of participants (bots) to reporting agents was varied. A fixed size environment of 10 regions was used, with 3 witness-narrator agents in each region. The number of bots per region was varied from 1–10 in 1 bot increments. The bots were confined to the region they started in. This also effectively confined the witness-narrator agents to the same region, as in the absence of teamwork or roaming participants they will not spontaneously travel between regions.

7.5.2 Results and Discussion

The results for **C1** are presented in table 7.4 and figure 7.2. These results have been averaged over 3 runs of the test setup. However, it is clear that there is still a large amount of variation in the data. The coverage levels out at around 25–30% when $N \geq 14$. The results

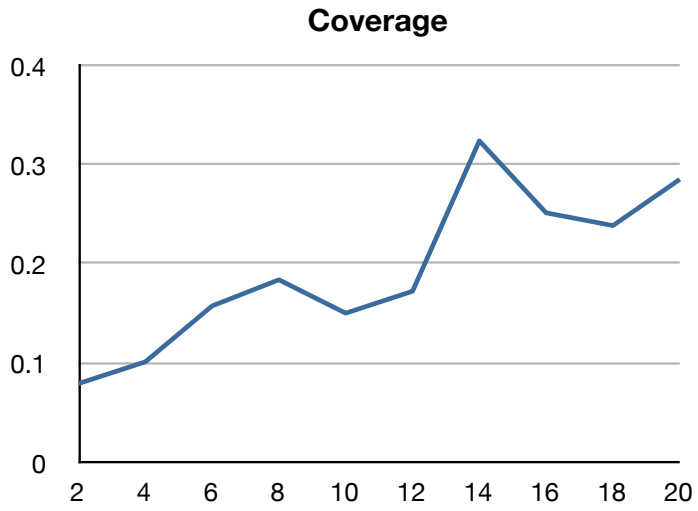

FIGURE 7.2: Experiment C1 Results

for **C2** are presented in table 7.5 and figure 7.3. Again, these results have been averaged over 3 runs.

The most striking aspect of the results is how little coverage is achieved using standard freelancing agents. Each agent follows some simple rules in the absence of teamwork. Firstly, they explore the environment in a random-walk from some starting point. Given that the agents start off at distinct way-points spread over the environment, the random walk behaviour tends to avoid agents bunching together in a particular part of the environment. Each agent is also programmed to follow any character of interest it encounters. This includes human player-characters and also testing bots (which are treated as if they were PCs by the agents). The results show that even in quite favourable conditions, the sheer volume of events occurring is too much for the agents to handle. Part of the reason for the poor coverage results seems to be that the code within *Neverwinter Nights* for detecting events and passing them on to nearby witness-narrator agents is not particularly reliable, and often agents will fail to be notified of an event that occurs in their very near vicinity. It is not clear why exactly this is, but it is possible that the *Neverwinter Nights* server avoids running some scripts or delivering some events when the load on the server is high. In **C2** we see that the best coverage achieved (28.371%) occurs when the number of agents matches the number of bots in each area (3). Before this point, there appears to be too few bots in the environment so that the agents are less likely to see one. After this

# Bots per Area	Events Generated	Events Reported	Percentage
1	640	105	16.406 %
2	799	219	27.409 %
3	927	263	28.371 %
4	1353	238	17.591 %
5	1580	228	14.430 %
6	2222	263	11.836 %
7	2229	257	11.530 %
8	3180	249	7.830 %
9	2332	230	9.863 %
10	3236	323	9.981 %

TABLE 7.5: C2 Results

number, the agents become overwhelmed and cannot cover all of the events that are occurring. The coverage quickly deteriorates after this point. The results of the two experiments show similar results for the cases where the ratio of participants (bots) to witness-narrator agents is the same. In particular, the coverage seems to be around 25% in both cases when the ratio of witness-narrators to participants is around 3:2.

7.6 Teamwork Tests

7.6.1 Method

The teamwork tests measured the effectiveness of the teamwork strategy in improving the coverage of events provided by the agents. The measure used was the number of agents required to achieve a certain level of coverage both with and without teamwork enabled. The hypothesis was that teamwork will enable the same level of coverage to be achieved with fewer agents than without teamwork.

T1: This experiment used a fixed size environment (10 areas) and a fixed number of bots (5 per area). We measured the minimum number of agents required to achieve a certain level of coverage in two conditions: with and without teamwork enabled. The coverage levels are expressed as percentages (events observed / events generated $\times$

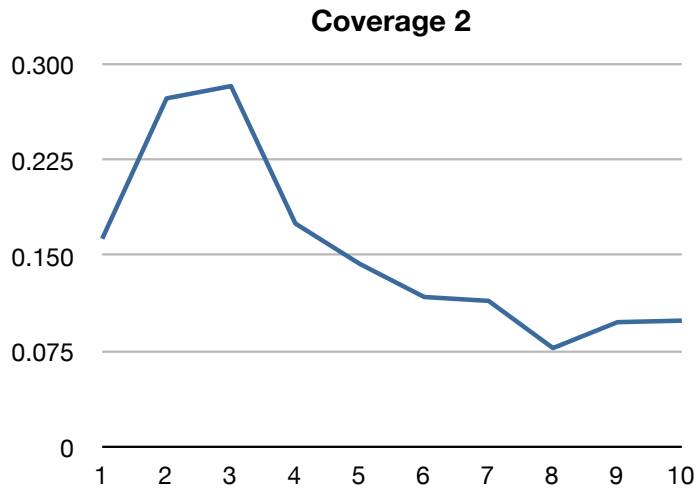

FIGURE 7.3: Experiment C2 Results

100). We took measurements at coverage levels from 10% to 90% in 10% increments. Measurements were taken by gradually increasing the number of agents in the environment until we would reliably achieve $x\%$ coverage over a 5 minute interval. A maximum of 100 agents was used for each condition.

7.6.2 Results and Discussion

Table 7.6 and Figure 7.4 show the results of these experiments. Neither condition, either with or without teamwork, was capable of performing at better than 60% coverage, which is a slightly disappointing result. However, within the results that were achieved, it is clear that the teamwork strategy does make a significant difference to coverage levels, allowing significantly better coverage results to be achieved for the same number of agents in the environment.

7.7 Live Evaluation

Once the initial technical tests had been carried out, the main live evaluation study was run in order to evaluate how real players interact with the framework, and how well the technology performs in a real world situation.

Target Coverage (%)	Without Teamwork	With Teamwork
10	10	10
20	40	35
30	70	50
40	100	65
50	-	95
60	-	100
70	-	-
80	-	-
90	-	-

TABLE 7.6: T1 Results

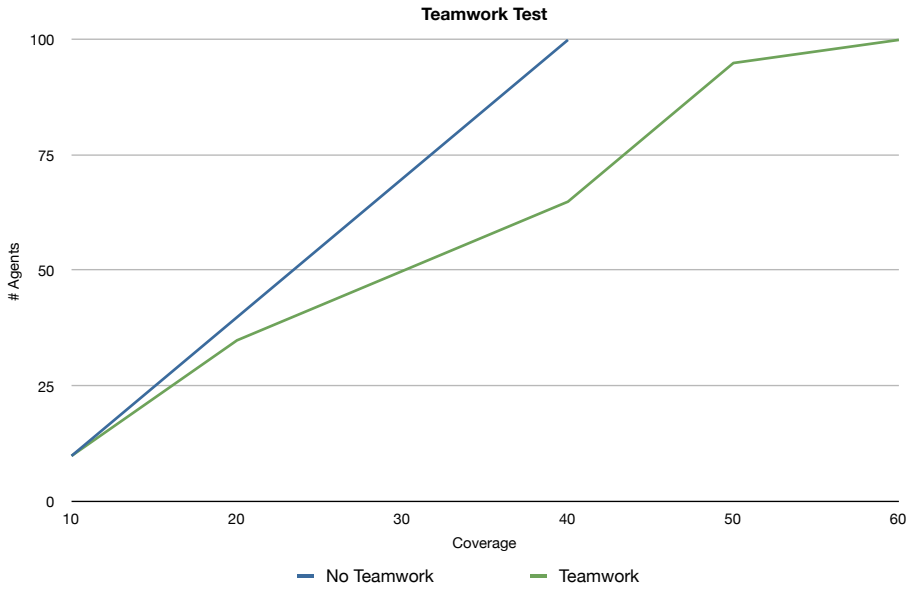

FIGURE 7.4: Teamwork test results

7.7.1 Method

The approach taken to the live evaluation study was to host a public *Neverwinter Nights* server on University of Nottingham equipment and to then recruit players from the general *Neverwinter Nights* community to play on the server, interact with witness-narrator agents and an external commentator agent, and to collect data on the performance of the system through comments, feedback forms, and a final questionnaire.

Recruitment

Participants were initially recruited from the existing population of *Neverwinter Nights* players by advertising on the official NWN message forums at the Bioware website[1], and also by advertising on a NWN group on the popular Facebook social networking site. In addition, once the study was live, it was advertised via the in-game GameSpy system that is used for advertising running games. An initial attempt to recruit players via these mechanisms failed to attract any respondents, so a second attempt was made, this time offering a £20 Amazon.co.uk gift voucher to compensate participants for their time, limited to a maximum of 25 participants. Despite this incentive, the uptake from the general internet population was still quite slow, and so further participants were recruited from the local Nottingham population, mostly undergraduate and graduate students. It is not clear why the response to the study was so poor initially, but possibly is related to the age of the game (*Neverwinter Nights* was originally released in 2002) and the amount of time required for the study.

There are a number of issues related to using online questionnaires and surveys for the evaluation. Firstly, there are sampling issues. Our target population for taking part in the study is existing Neverwinter Nights players. We cannot develop a sample frame of this population as there is no list of such users. Therefore we must use a non-probabilistic sampling method. Coomber [32] suggests that online self-selection may be appropriate when researching a particular group of internet users (such as our target population). We therefore solicited participation from the existing population of Neverwinter Nights players using the existing public message boards. The guidelines given in [76] also recommend getting ethical clearance for such research and that respondents should have given their informed consent. In addition, there are guidelines from the Higher Educa-

[1] http://nwn.bioware.com/

tion and Research Organisation (HERO) on ethics and equal opportunities in relation to online questionnaires. Finally, [76] provides a checklist of items to consider to maximise response to an online questionnaire. We followed these guidelines, such as providing an institutionally sanctioned project website to verify our own identity and affiliation.

Ethics

In planning a large publicly recruited evaluation study it is necessary to consider the ethical implications of the study. In designing this evaluation study we have followed the guidelines described in [77] as well as following the ethics guidelines of the Mixed Reality Laboratory at the University of Nottingham, who funded the study. An ethics checklist was completed as per University policy and approved by the Head of School and an ethics officer within the School of Computer Science.

All participants were free to withdraw from the study at any time, and were informed of their rights before the study commenced, via an online webpage. Due to the way in which participants were recruited from the general Internet population, it was not practical to obtained signed written consent for each participant. Instead, participants were presented with two online consent forms. Firstly, when entering the persistent game world for the first time, players were presented with a short introduction to the study and asked whether they understood the terms of the study and agreed to take part. Secondly, at the end of the study participants were asked to complete a short questionnaire. This questionnaire also informed participants of their rights (including their right to withdraw from the study at any time) and again asked participants to indicate that they gave their informed consent for their results to be used in the study.

Data Collection

The data collected from participants during the study included game logs detailing the times at which they logged in to the game, and their activities in the game world. Participants were also able to access the website with reports of their activities, and could read and rate reports for interest and accuracy as well as completing the online questionnaire. All data collected from participants was anonymised once we had been able to verify that they had fulfilled the requirements of the study. This included removing references to both participants' email addresses and also the user-names they used in-game (which may form

part of their online identity). The data collected has been kept only in anonymous form, and cannot be linked back to individual participants. All data was used only in aggregate form, except for some quotes given in questionnaire responses, for which participants were asked to give explicit consent for their use. Participants were informed of all of the details of the data that would be collected on their participation, and their rights with regards to the storage and use of this data.

7.7.2 Questionnaire

At the end of the study a questionnaire was placed on the website and participants were invited to fill it in. The questionnaire asked the following questions:

Personal information

The following questions will be used in the analysis of your data. Answering these questions will not compromise the anonymity of your data.

1. How old are you in years? (10–19,20–29,30–39,40–49,50+)
2. Are you male or female?
3. What user-name did you use in-game?
4. If you wish to be considered for the Amazon.co.uk gift voucher, please enter your email address here:
5. Do you agree to be contacted with any follow-up questions?
6. Do you agree to any quotes you provide being used in any publications that result from this study?

Neverwinter Nights and the Reporting Agents

1. How often do you play *Neverwinter Nights*? (Never, Hardly ever, Once or twice a week, Every day)
2. When you do play *Neverwinter Nights*, how long do you usually play for? (Less than an hour, 1–2 hours, 3+ hours)
3. Do you think you played *Neverwinter Nights* more or less when the reporting agents were present? (More, About the same, Less)
4. Why was this? (Free text input box)
5. How would you rate the overall interestingness of the reports produced? (Very Interesting, Interesting, Not Interesting, Dull)
6. Why was this? (Free text input box)
7. How would you rate the overall accuracy of the reports produced? (Very Accurate, Accurate, Inaccurate, Very Inaccurate)
8. Why was this? (Free text input box)

9. Did you find the agents disruptive or intrusive in any way? (Very Intrusive, Slightly Intrusive, Not Intrusive)
10. Why was this? (Free text input box)
11. Did the agents increase your overall enjoyment of the game? (Yes, No)
12. Why was this? (Free text input box)
13. Finally, do you have any other comments about the study or the system? (Free text input box)

The questionnaire also had a final link to the rights and obligations of taking part in the study, and respondents were asked to indicate their informed consent before submitting the questionnaire.

7.7.3 Results and Discussion

In total, 11 participants completed the study and answered the questionnaire questions. It was originally hoped to collect data on individual reports, allowing participants to rate each report for both interest and accuracy on a simple 5-point scale. However, in the event, no participants chose to evaluate individual reports.

The study was carried out over approximately 2 weeks, with participants mostly logging in to the game for an hour or two on a single day and then not returning to the game later, except to fill in the questionnaire. This meant that almost all of participants played the game on their own. The only exceptions to this were some locally recruited participants, who were able to play together in small groups using supplied equipment. The lack of time spent by participants in the game also limited the activities which they could perform. A newly created player in the game is initially quite weak as a character, and so generally is not able to undertake any of the quests in the game as they would not be able to complete them. This resulted in the vast majority of reports produced being simple accounts of battles fought between players and various creatures in the environment. The only exception to this was a single occassion in which a player managed to 'level-up' (i.e., gain an experience level).

Personal Information

Figure 7.5 and Figure 7.6 show the distribution of respondents in terms of age and gender, respectively. Nine of the respondents were in their twenties, with the other 2 in their

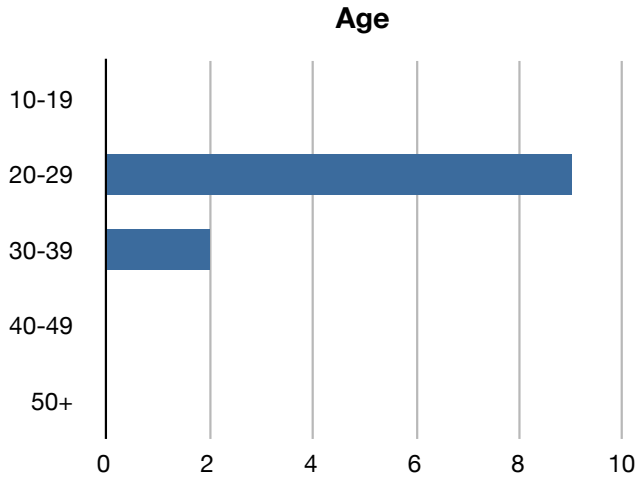

FIGURE 7.5: Age distribution of respondents.

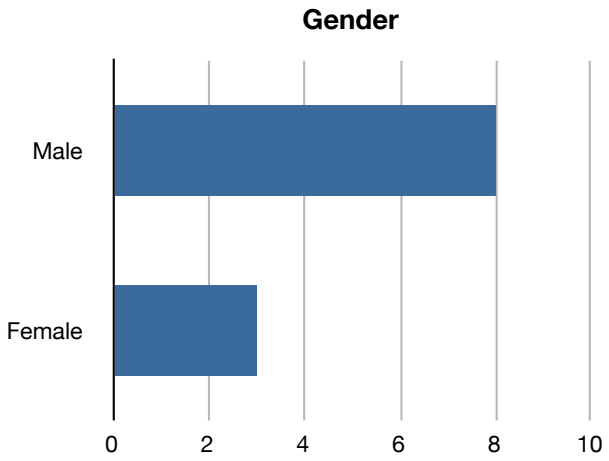

FIGURE 7.6: Gender distribution of respondents.

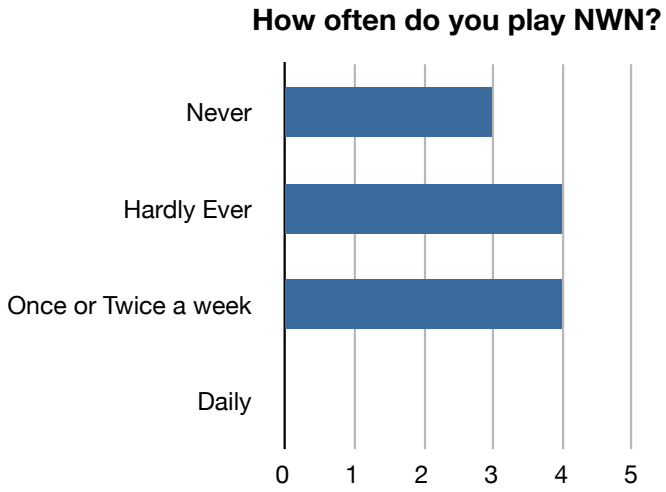

FIGURE 7.7: How often respondents play NWN.

thirties. Only three respondents were female, and these were all recruited locally. All respondents recruited from the internet were male and in their twenties. The sample population shows a clear bias towards young male adults, which is to be expected given the self-selected population of computer game players.

Previous Experience with Neverwinter Nights

Figures 7.7 and 7.8 show the previous experience of the participants with *Neverwinter Nights*. Due to the difficulty in recruiting participants from the general NWN population via the internet, the majority of respondents (who were recruited locally) had limited experience with the game. Due to this unforeseen lack of uptake from experienced NWN players, locally recruited players were asked to indicate their general experience with similar games, rather than with NWN specifically. For instance, a number of locally recruited players had experience with *World of Warcraft* or other popular role-playing games.

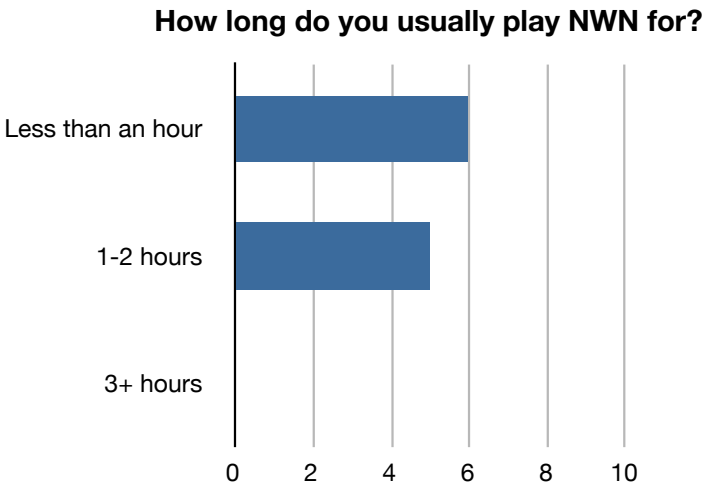

FIGURE 7.8: How long respondents usually play NWN for.

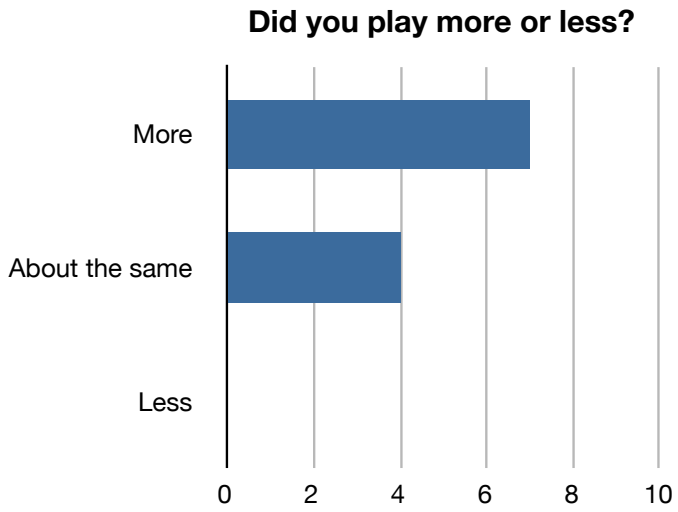

FIGURE 7.9: Did respondents play more or less when the agents were present?

Participation

Figure 7.9 shows the response of the participants to whether they felt they played the game more or less when their actions were being reported on by the witness-narrator agents. The results show a clearly expressed preference for when the agents were present, with 7 respondents preferring to play with the agents present, and only 4 being indifferent to the agents. No respondents said that they would play less when the agents are present. However, the actual amount of time spent in the game world during the study by individual particpants was rather low. The most time spent in the game was a single participant who played on several consecutive days and for a total of around 5 hours. None of the other participants played in the game world for much more than about an hour, so it is difficult to draw conclusions as to whether the responses to the questionnaire would match up with reality over a longer study period. The lack of participation also made it impossible to conduct a rigorous control study to see if players did indeed play more when agents

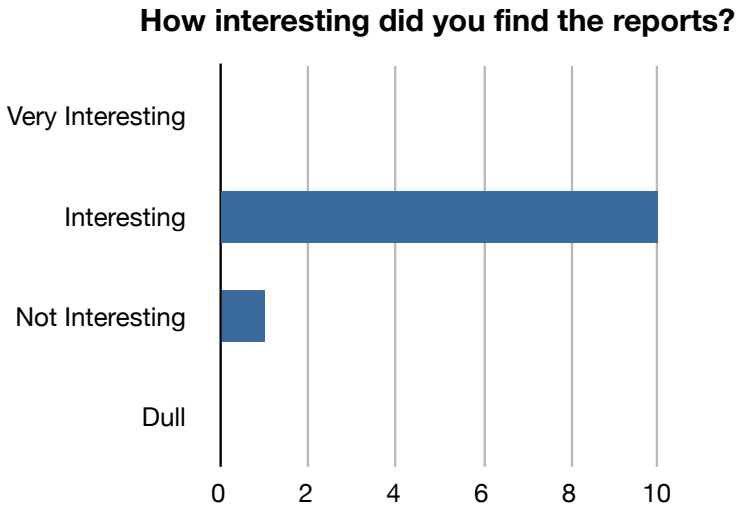

FIGURE 7.10: How interesting were the reports?

were present.

Only a handful of respondents gave reasons as to why they would play more. Of these, the main reason seemed to be that players enjoyed being able to read reports of what they had done after the fact. One respondent wrote *"I liked to be able to look back at what I had done"*, while another wrote *"The purpose of the agents is cool, but sometimes they would get in my way. I wasn't too concerned with them as I was playing."*

Interest

Figure 7.10 shows how respondents rated the overall interest of the reports that were produced. As previously noted, no respondents chose to rate individual reports, and the vast majority of reports were quite similar in nature due to the limited participation. As such, it is not possible to evaluate whether there was a difference in interest between different types of reports produced by the system. The battle reports produced during game-play

Skirmish in Galum Hills!

There was a battle in Galum Hills today, involving Kale Veuthian, a Goblin and a Hobgoblin.

It all started when a Goblin attacked Kale Veuthian with their light mace. Then, Kale Veuthian attacked Goblin with their shortsword. The Goblin attacked Kale Veuthian with their light mace. Kale Veuthian attacked the Goblin with their shortsword. The Goblin attacked Kale Veuthian with their morningstar. Kale Veuthian attacked the Goblin with their shortsword. The Goblin was slain by Kale Veuthian. Kale Veuthian attacked a Hobgoblin with their shortsword. The Hobgoblin attacked Kale Veuthian with their longsword. Kale Veuthian attacked the Hobgoblin with their shortsword. The Hobgoblin attacked Kale Veuthian with their longsword. Finally, the Hobgoblin was slain by Kale Veuthian.

FIGURE 7.11: An example battle report from the live evaluation.

are perhaps the least interesting of the reports produced. Figure 7.11 shows an example battle report produced by the software during the study. As can be seen from the output, battle reports mostly result in a blow-by-blow account of the action, with little effort put in to making the report interesting from a narrative point of view. Despite these drawbacks, the majority of respondents (10) rated the reports as "interesting", with only a single participant rating the reports as "not interesting". The individual responses to this question are quite revealing. A selection of some of the responses are as follows:

- *"It's interesting to get some updates on what is going on, but the reports themselves are a bit dull."*
- *"They seem to focus on combat and the sentences are literally a blow by blow account of the battle. They are interesting [but] maybe some reports on loot or talking to NPCs would help vary things."*
- *"I liked to read about my victories."*
- *"They reported on my interactions in the game which was usefull [sic] in backtracking over events and detailed my killinmg [sic] spree."*

From these responses, we can see that the main interest for respondents lay in the chronicling of their individual accomplishments in battle. However, several respondents picked

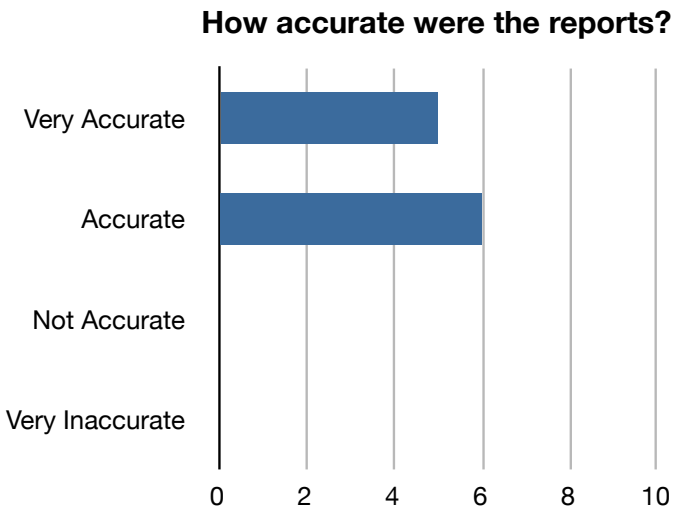

FIGURE 7.12: How accurate were the reports?

up on the repetitive nature of the reports and their 'blow-by-blow' appearance. This is clearly a limitation of the current framework, and an area which would require more work in future. As mentioned in the previous chapters, the prose generation aspect of the presenter capability was not developed beyond basic text templating. This lack of variation in the generated reports can clearly be seen to be a major drawback in the evaluation.

Accuracy

A corollary of the relatively dull and formulaic reports produced by the software during the live study was that the resulting reports were rated as being quite accurate. All respondents felt that the reports were accurate, with almost half (5) rating the reports as "very accurate". This is most likely due to the low level of reporting that is used in battle reports. Beyond recognising a battle is occurring, most of the details of the reports consist of individual actions performed by individuals. Such directly observed actions are always

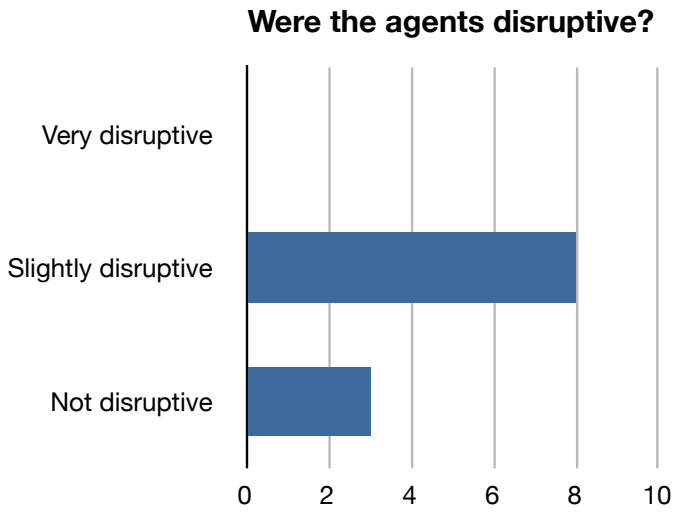

FIGURE 7.13: Were the agents disruptive in any way?

entirely accurate within the framework, as if an agent observes the event at all it will correctly observe all details of that event, and no inference is needed to fill in any gaps. This is in contrast to reports of quests or other more complex events, in which case the system employs a less reliable inference process. Some sample responses to this question included:

- *"I didn't see any discrepancies with the reports and what actually happened as far as I can tell."*
- *"[T]hey reported on everything I did."*
- *"They missed a little bit of my fighting but what they did see was accurately reported."*
- *"[The agents] seem to leave out some events and the order is not always accurate."*

Disruption

Figure 7.13 shows the responses to whether the witness-narrator agents were disruptive to the game-play in any way. Despite their small size, the agents were found to be "slightly disruptive" by the majority of respondents, with only 3 people stating that the agents were not at all disruptive to the gameplay. However, no respondents felt that the agents represented a major source of disruption to their usual gameplay. From the free-text responses to this question, it is apparent that the major source of complaint with regards to the agents is that they sometimes get in the way of the players. From observing players during the game, it appears that this is partially the result of teamwork: when a battle breaks out the scene can quickly become flooded as a number of nearby witness-narrator agents immediately move into the vicinity in order to ensure good coverage of the battle. Another source of complaint was that when an agent is following a player, the agent will sometimes 'jump' directly into the path of the player. This seems to be an artefact of the implementation of the follow behaviour in *Neverwinter Nights*: if a player moves too far away from the agent, the game will simply 'teleport' the agent back to a position near the player, which sometimes happens to be directly in the way. Some sample comments from this question include:

- *"They get in the way sometimes, and they tend to crowd any battle."*
- *"Sometimes there would be multiple agents and they would block my way, but this did not happen very often."*
- *"When they were following me they sometimes got in the way, but I didn't really mind."*
- *"They're not intrusive byt [sic] they sometimes get in the way. They sometimes jump right in front of you when you are moving."*

Overall Enjoyment

The results of the most important question—whether the presence of the agents actually increased participants' enjoyment of the game—are shown in Figure 7.14. These results are very promising, with all respondents (11) indicating that they enjoyed the game more because of the witness-narrator agents and having their actions reported on. The responses to this question reveal some of the reasons for this:

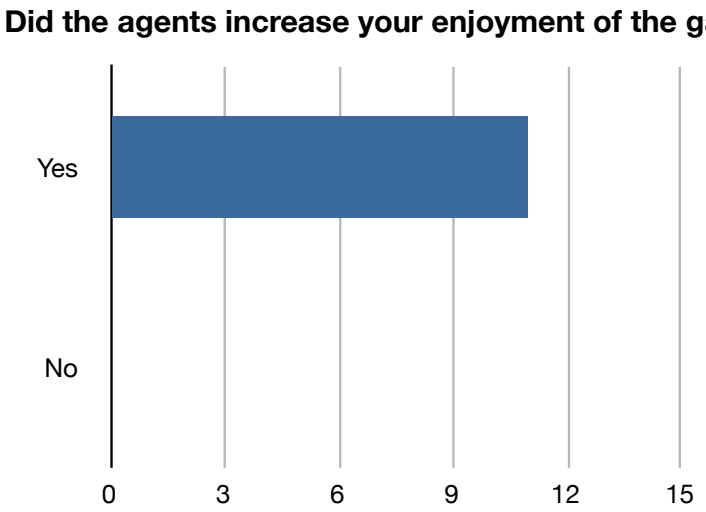

FIGURE 7.14: Did the agents increase your enjoyment of the game?

- *"It was cool to feel like what I did in the game was recorded and didn't end when I logged off."*
- *"[B]ecause I like to read about winning battles after I played."*
- *"Screenshots would be nice if possible."*
- *"Yes the reporters reacted to my actions much like a paparazzi scrum which was quite fun when on a killing spree."*

General Comments

Only two respondents left general comments at the end of the questionnaire. One respondent agreed with a respondent to the previous question that screenshots would make the reports more interesting. The other respondent was disappointed with the in-game presenting capability of the witness-narrator agents, stating *"nothing ever seems to be going on when I ask them."* This latter complaint is likely due to the lack of participation in the study. It is expected that with a larger group of participants it would be more likely that the witness-narrator agents would have something to report when asked to do so in-game.

CHAPTER 8

CONCLUSIONS

8.1 Conclusions

We have described witness-narrator agents, a framework for narrative generation in persistent virtual environments and its implementation as a multi-agent system. Our approach is distinguished by two key factors. First, there is no single overarching narrative, but rather the narrative consists of many strands. Some narrative strands, e.g., those relating to major conflicts or relating to the 'backstory' of the environment, may be widely shared, while others, e.g., an account of an individual quest, may be only of interest to a single user. Second, the generation of narrative is collaborative, in that users have both direct and indirect control over which events are ultimately narrated. In particular, users have control over:

- which events are performed — each participant determines which events are potentially observable
- which events are observed — participants have negative control over which events get reported, in that they can ask the agents to go away, or simply avoid them
- which events get reported — users have positive control in determining which events the agents will search out and report
- which events are remembered and hence can form part of future narratives — users can rate reports, which determines which reports (and hence which actions/events) become part of the "official" collective narrative or backstory of the environment.

More generally, one role of narrative is to develop a sense of shared values and experience, e.g., "what it means to be a player of *Neverwinter Nights*". In this case, the function of the narrative is not just to entertain, but to help foster a sense of community, through participation in and shared experience of the narrative. We believe the witness-narrator agent framework can be seen as a first step towards this goal for the domain of persistent virtual worlds.

In this thesis we have described the design and implementation of a framework that can produce simple narratives from the actions of players in a persistent role-playing game, *Neverwinter Nights*. The work has been evaluated both technically and qualitatively. The technical tests have shown that the framework is capable of handling a reasonably complex environment, with up to several dozen simultaneous participants, while the qualitative live evaluation has shown that producing narratives based on players' experiences increases enjoyment in the game, and users found the generated stories interesting. The evaluation has also highlighted both the strengths and weaknesses of the current approach. On the positive side, responses to the evaluation have justified the approach taken and the idea of having events in a persistent role-playing game reported on automatically seems to be popular. On the less positive side, the variety and quality of the generated reports seems to require further work to avoid appearing repetitive or dull, at least in the case of the battles and low-level actions that were produced during the evaluation.

8.2 Summary of Contributions

In the introduction, a number of research objectives were identified. Throughout the thesis we have concentrated on providing solutions to these identified problems. In particular, the research presented has achieved the following aims:

1. A framework for representing and reasoning about events occurring in a diverse range of persistent virtual environments has been developed and successfully applied to an existing commercial role-playing game;

2. Existing theories for knowledge representation and multi-agent systems have been adapted and extended to deal with the specific task of reporting on events in virtual environments;

3. The system has been demonstrated and evaluated in an environment supporting a

reasonably large number of simultaneous participants, and we feel confident that the system could scale up to larger environments;

4. We have evaluated the system in a realistic scenario and the responses have been on the whole positive, reinforcing the view that reporting on virtual environments is a worthwhile direction for research, and the approach taken in this thesis is a good approach.

A number of key research contributions have also been identified:

- Development of a framework that can be applied to a wide variety of different online persistent environments;
- Providing a formal basis for describing events, incorporating elements from narrative theory and a number of knowledge representation formalisms;
- Integration of a number of different facets of knowledge representation: temporal knowledge, actions and events, groups and teams, and general ontological knowledge;
- Application of multi-agent development techniques to large-scale persistent environments running for extended periods of time.

In the course of developing the framework we have also had to adapt and incorporate a number of technologies. For instance, we provide a simple and efficient means of incorporating temporal knowledge into a description logic ontology in a manner that requires little adaptation of the existing semantics of the logic.

8.3 Reflections

8.3.1 Strengths

The main strengths of the system as implemented lie in its ability to be adapted to different operating environments due to the use of the 'pluggable' ontology, and in the relative scalability of the system compared to previous work. While the system has currently only been applied to the *Neverwinter Nights* environment, it is believed that it can be adapted in a straight-forward manner to other similar environments, such as existing MMORPGs,

and even to more diverse environments. An interesting experiment would be to adapt the system to the *Unreal Tournament* game used in Daniel Fielding's earlier work (Section 2.5.2 (page 45)). In comparison with this earlier work, the current system presents a more comprehensive approach, with each area of the task (reporting, editing, and presenting) developed in greater depth, and with more consideration of existing technologies and methodologies, as outlined in the literature review (chapters 2 and 3). While there are clearly still many obstacles to be overcome, and much future work is needed to develop a truly production-quality system, the thesis provides a useful guide to the sorts of questions that need to be answered, and the types of approaches that are likely to yield success. In the following sections we describe some of the remaining limitations in more depth, and reflect on the technology choices made in the current implementation, before describing planned future work to further develop the technology to address these limitations.

8.3.2 Limitations

There are a number of clear limitations in the system that can be identified, and in its evaluation in this thesis. Firstly, and perhaps most importantly, the system makes no commitment to a particular notion of what is an 'interesting' event, instead delegating this task through the mechanism of focus goals. To the system, what consitutes an interesting event is simply what matches a currently active focus goal. This approach, while practical and flexible, fails to build on the literature of narrative theory and previously implemented storytelling systems, where the concept of an interesting story is much more comprehensively defined, through notions such as 'dramatic arc'. This lack of commitment to a particular notion of interestingness can be seen as contributing to the relative lack of entertainment value in the resulting narratives, and the correspondingly poor assessment given by the live evaluation participants. The ontology actually contains a flexible notion of Story, which considers notions such as character, setting, and plot. This framework is sufficiently expressive to represent a number of different story description and generation frameworks described in the literature, but currently lacks any theory of how such stories should be constructed from individual event descriptions so as to maximise the dramatic impact of the resulting narrative. In addition to this theoretical difficulty, the system as implemented also fails to take advantage of the existing capabilities to the full. The vast majority of narratives that can be generated from the system are really just reports of a sin-

gle event, possibly with some largely unstructured sub-events, such as battles. In reality, such battles would be expected to consist of various plot-lines, as different sides attempt to carry out individual tactics and schemes, which are either successful or thwarted. To an extent, this is a result of the particular environment chosen for the evaluation, in which battles are usually too short-lived to involve any particularly complex tactical maneuveurs or other interesting plot twists. Recognising complex interactions between groups of players, such as double-crossing and betrayal, is also extremely difficult currently, as there are few clues from the environment that such activities are occurring. However, even with the relative lack of information currently available it should be possible to construct some examples of more story-like reports and implement these in the system.

In addition to the lack of dramatic narrative structure in the produced narratives, the quality of the generated prose is also of lesser quality than could be produced with state-of-the-art techniques. The current system uses only a very basic text templating system, with a few extra rules to tidy up punctuation and capitalisation, and some attempts at using appropriate pronouns rather than full names everywhere. This results in rather dull prose generation, with little variety in the output for similar events. Modern natural language generation techniques could improve this output to a certain degree. The most important aspects of these improvements would be:

- better content detemination, to avoid including every small detail in the final output;
- incorporation of a summarisation component, that can combine multiple instances of similar events into brief summaries, such as those discussed by Genette (Section 2.1.2 (page 9));
- more use of context to eliminate describing characters and settings multiple times in detail, and to generate more variety in phrasing.

These aspects are perhaps best addressed by directly incorporating an existing NLG component and then specialising it for the particular ontology being employed.

A final limitation of the current system is the lack of exploration of the '*Why?*' aspect of events. While there is a shallow notion of 'cause' present in the system, that can address the direct question as to why an event occurred, there is no deeper notion of motive or reason behind a character performing an action. We term the process of inferring these deeper reasons behind players' actions *motive recognition*. It was initially hoped that

the system would incorporate just such a capability, using a more comprehensive plan recognition facility to determine the plans and plots occurring within the environment, but unfortunately time constraints prevented this work from being completed. It remains a major piece of future work.

8.3.3 Methodology

The methodology adopted in the designing the system presented in this thesis was largely 'bottom-up' in character. The approach proceeding as follows:

1. investigate all of the low-level actions that can be directly observed from the environment;

2. develop a number of potentially interesting higher-level event descriptions from these low-level actions;

3. attempt to manufacture some interesting reports based on these event descriptions.

While this approach ensured that the system was always capable of actually observing sufficient information to produce each story type that it could produce, it was perhaps not the most appropriate methodology for ensuring that the resulting narratives were of interest to users of the system and demonstrated the full range of the system. A better methodology would have adopted a top-down approach, starting from some example stories that we would like the system to be able to produce, and then using a process of iterative prototyping to adapt these example stories to the actual actions and event descriptions that are observable from the environment.

In developing a complex ontology of the sort we developed for this application, which spans a wide variety of different subject areas, it is easy to get carried away with attempting to generalise to the point at which the ontology is capable of expressing a large part of human experience. Our experience with developing this system suggests that such desires should be suppressed as much as possible. In attempting to capture the essence of every possible story, the system has ended up being clumsy to use in specifying any *particular* story. A useful guideline for anybody attempting to replicate this work would be to develop the ontology to be *as general as your requirements dictate, but no more so*. In particular, addition of new concepts to the ontology should always be directly driven by the requirements for representing one or other of the corpus of example stories that are

driving developments. No concept should be introduced unless it results in a significant and desireable new distinction in the resulting story generation process. This is not to say that there is no utility in general high-level ontologies, particularly when they enable the sharing of ideas between diverse agents, but that such generalisation is extremely difficult to get right. Application-oriented ontologies should be tightly constrained by the actual purpose of the final system, and then relaxed only as further requirements dictate.

8.3.4 Choice of Technology

An early decision in the implementation of this work was to use the Web Ontology Language (OWL) as the representation language for the ontology that underlies all of the agents' reasoning about events, stories, characters and settings. This choice was justified mainly by the fact that OWL offers a good trade-off between expressivity and computational complexity. The choice has proved adequate for the majority of the ontology, and the availability of good modern tools for developing in OWL lead to an efficient development process in which it was easy to experiment with different conceptualisations of different concepts. However, the choice was not without significant drawbacks. In particular, insufficient consideration was paid to the temporal and procedural aspects of the task at hand. The main back-bone of the ontology is the description of events, actions, and stories with plan-like structures—such as quests. Each of these requires some consideration of temporal aspects of representation, such as the temporal ordering and overlap of events, the effect of actions on properties (fluents) of objects, and the constraints relating different steps of a plan. OWL itself has no explicit support for any temporal reasoning, and indeed many *ad-hoc* extensions had to be made to incorporate these aspects of the design.

In retrospect, more time should have been invested at the outset in clearly specifying the temporal and procedural representation requirements of the domain and then either selecting a more appropriate representation medium, or more carefully formulating solutions to these problems as extensions to OWL. For example, one problem was in the specification of complex events that require coreference constraints between different properties. The example that repeatedly presented itself was how to adequately represent a crime as being an action that is illegal in the region in which it is performed. Even this rather simplistic definition of a crime is not expressible within OWL, as it requires equating the region of an action (one property) with the jurisdiction of a law (another property).

OWL lacks the logical machinery to make such an assertion, and we had to resort to the more general definition that a crime is an action that is illegal *somewhere*, and then augment this definition with specialised rules. Such assertions are easily expressible in various rule languages, such as PROLOG or DATALOG. Another example of the difficulties encountered was the lack of a proper subsumption relationship between plan-like entities. It is not possible for the system to infer that one plan is a sub-set of another plan, because OWL is not aware of the special nature of steps in plans. It was also clumsy to specify the various types of constraints between steps of a plan, such as that all steps should be carried out by the same agent. Again, these constraints were hard-coded into the agent implementation to make up for the lack of expressivity in OWL. Given the importance of these types of concepts in the overall system, it may have been more appropriate to adopt a more specialised representation, such as the description logic with plans of T-REX (Section 2.4.1 (page 41)) or a plan representation language such as PDDL (Planning Domain Definition Language).

Apart from OWL, the other major technology adopted in the implementation of the system was *Jason*, an agent-oriented programming language based on a BDI (belief-desire-intention) architecture. Jason was chosen because it offered a concise and powerful notation for specify multi-agent systems based on a sound theoretical basis (AgentSpeak(L) and BDI logics). As for OWL, this choice turned out to have both advantages and disadvantages. The advantages of Jason were the relatively simple way in which quite complex agent behaviours could be implemented, as well as the ease of integration with custom components written in Java, which were essential for connecting to *Neverwinter Nights*. However, there were a number of clear disadvantages to the use of Jason, some of which related to the maturity of the current implementation, but others which reflect design weaknesses:

- The implementation was relatively untested when the work was commenced. This thesis work was, to the best of our knowledge, the single largest application of Jason that had been attempted. The live evaluation pushed the system beyond most previous uses, and a number of bugs and stability problems were encountered that had to be worked around or fixed.

- The implementation was also evolving quite quickly, with updates and potential incompatibilities being introduced frequently. Eventually, we had to abandon keeping up-to-date with the latest version and instead maintained our own local copy of the

source-code as a stable base.

- Jason lacks support for modular construction of large code-bases, at a level intermediate between that of 'agent' and 'plan'. Our agents were developed by including different source code files, which led to problems due to a lack of compositionality and modularity in this process.

- The use of PROLOG for the belief base language, while flexible, resulted in an inefficient interface to the underlying relational database that was used to store beliefs. In particular, the reasoning engine frequently retrieved large numbers of irrelevant beliefs that were then almost immediately discarded. Given that our live evaluation study produced beliefs numbering into the hundreds of thousands, and even millions, this resulted in significant wasted processing and network use.

Despite these drawbacks, Jason still seems like a promising approach to constructing such systems. However, much work remains to be done in plugging the gaps in the specification of the language to better address the needs of large-scale modular agent construction, while also allowing for much more efficient and scaleable implementations to be developed.

8.4 Future Work

The work that has been presented encompasses a large number of different areas of artificial intelligence research. In order to develop the framework presented, research from diverse areas such as narrative theory, knowledge representation and reasoning, plan and activity recognition, intelligent agents and multi-agent systems have had to be woven together. The resulting framework, while powerful, in some cases only incorporates quite basic versions of the various technologies discussed. The evaluation has highlighted perhaps the biggest current drawback, which is the comparatively poor narrative prose generation that is performed. As previously discussed, improvement of this capability requires work in at least two areas:

1. development of a more comprehensive description of a story, incorporating ideas of dramatic arc in order to produce more compelling content;

2. incorporation of more sophisticated natural language generation (NLG) in order to create more variation and conciseness in the generated prose.

The current approach relies on simple text templates and some very rudimentary rules for generating some basic variation in the text produced. These techniques, while adequate, are clearly noticeable in the quality of the produced output. However, good natural language generation (NLG) is a difficult task, and was not considered to be an area in which this thesis was expected to produce novel research. It was therefore decided to not concentrate too much effort on this task. A clear area for future work on the framework would therefore have to begin with improving the text output capabilities, perhaps incorporating a complete NLG component.

The other major area for future research is in the development of the idea of *motive recognition* for inferring the deeper reasons behind players' actions. We believe that a significant aspect of what constitutes an interesting story relates to the motives, schemes, and tactics that underly character interactions. These aspects are currently woefully underexplored in this work. Future work will look at exploring more comprehensive plan recognition, as well as looking at deontological and social aspects of characters to reveal conflicts of interest, double-crossing, revenge, betrayal, and other motivations that could potentially significantly improve the narratives that are produced.

Work is also currently underway to address the problems that were discovered in the representation technologies that we used, namely OWL and Jason, discussed in the previous section. We intend to address the problems in Jason by extending and adapting the language to better support the needs of modular agent development. For OWL, we intend to properly characterise the additional features that are required in order to correctly specify the types of events and plans that are featured in the ontology. Further work would then attempt to determine the increase in computational complexity that would result from the incorporation of these features.

The multi-agent teamwork approach taken has been largely successful, but more research could be performed to better understand how to coordinate such large-scale teams as we have employed. The use of focus goals is also a new area of research, and the rules for forming and disbanding teams in response to new and ongoing focus goals could be further investigated, for instance to develop a clear approach for when a focus goal can be considered achieved or otherwise able to be dropped.

APPENDIX A

ONTOLOGY AXIOMS

A.1 Introduction

This chapter contains the complete axiomatisation of the formal ontology developed in chapter 5. The source of this chapter was generated automatically by the LaTeX exporter in Protégé 4.

A.2 Classes

Achievement

Achievement $\equiv$ Event $\sqcap \exists$ achieves Objective
Achievement $\sqsubseteq$ Happening

Acquire

Acquire $\sqsubseteq$ Manipulate

AcquirePower

AcquirePower $\sqsubseteq$ PersonalObjective

AcquireProperty

AcquireProperty $\sqsubseteq$ PersonalObjective

AcquireWealth

$\text{AcquireWealth} \sqsubseteq \text{AcquireProperty}$

Act

$\text{Act} \equiv \text{Event} \sqcap = \text{performs Action}$
$\text{Act} \sqsubseteq = \text{hasActor Actor}$

Action

$\text{Action} \equiv \text{Interact} \sqcup \text{Manipulate} \sqcup \text{PersonalAction}$
$\text{Action} \sqsubseteq \text{Thing}$

Actor

$\text{Actor} \equiv \text{Group} \sqcup \text{Individual}$
$\text{Actor} \sqsubseteq \text{Existent}$

Advise

$\text{Advise} \sqsubseteq \text{Say}$

Agent

$\text{Agent} \equiv \text{Actor} \sqcap \exists \text{ hasObjective Objective}$
$\text{Agent} \sqsubseteq \text{Actor}$

AgriculturalBuilding

$\text{AgriculturalBuilding} \sqsubseteq \text{Building}$
$\text{AgriculturalBuilding} \sqsubseteq \text{AgriculturalRegion}$

AgriculturalRegion

$\text{AgriculturalRegion} \sqsubseteq \text{RuralRegion}$

Ammunition

Ammunition $\sqsubseteq$ Prop

Argument

Argument $\equiv$ Conversation $\sqcap \exists$ hasSubEvent (Act $\sqcap \exists$ performs (Taunt $\sqcup$ Threaten))
Argument $\sqsubseteq$ Conversation
Argument $\sqsubseteq \forall$ hasParticipant Individual
Argument $\sqsubseteq$ Dispute

Ask

Ask $\equiv$ Say $\sqcap \exists$ hasMessage Request

Assassination

Assassination $\equiv$ Act $\sqcap \exists$ performs (Kill $\sqcap \exists$ hasTarget Leader)
Assassination $\sqsubseteq$ Act

Attack

Attack $\sqsubseteq$ Interact

Bag

Bag $\sqsubseteq$ Container

BallisticWeapon

BallisticWeapon $\sqsubseteq$ Weapon

Barn

Barn $\sqsubseteq$ StorageBuilding
Barn $\sqsubseteq$ AgriculturalBuilding

Barracks

Barracks $\sqsubseteq$ MilitaryBuilding

Battle

Battle $\equiv$ Combat $\sqcap \geq$ 10 hasParticipant Actor
Battle $\sqsubseteq$ Combat

Beach

Beach $\sqsubseteq$ Shoreline

Bed

Bed $\sqsubseteq$ Furniture

Birthday

Birthday $\sqsubseteq$ PersonalEvent

BirthdayParty

BirthdayParty $\equiv$ Party $\sqcap \exists$ inCelebrationOf Birthday
BirthdayParty $\sqsubseteq$ Party

BodyOfWater

BodyOfWater $\sqsubseteq$ RuralRegion

Box

Box $\sqsubseteq$ Container

Building

Building $\sqsubseteq$ Structure

Buy

$\text{Buy} \sqsubseteq \text{Take}$

Camp

$\text{Camp} \sqsubseteq \text{Settlement}$

Castle

$\text{Castle} \sqsubseteq \text{Fortress}$
$\text{Castle} \sqsubseteq \text{MilitaryBuilding}$

Cave

$\text{Cave} \sqsubseteq \text{SubterraneanRegion}$
$\text{Cave} \sqsubseteq \text{RuralRegion}$

Chair

$\text{Chair} \sqsubseteq \text{Furniture}$

Character

$\text{Character} \equiv \text{Individual} \sqcap \exists \text{ isCharacterIn Story}$
$\text{Character} \sqsubseteq \text{Individual}$

Chest

$\text{Chest} \sqsubseteq \text{Box}$
$\text{Chest} \sqsubseteq \text{Furniture}$

City

$\text{City} \sqsubseteq \text{Settlement}$

CivilBuilding

$\text{CivilBuilding} \sqsubseteq \text{Building}$

CivilOrganisation

$\text{CivilOrganisation} \sqsubseteq \text{Organisation}$

Cliffs

$\text{Cliffs} \sqsubseteq \text{Coast}$

Close

$\text{Close} \sqsubseteq \text{Manipulate}$

Clothing

$\text{Clothing} \sqsubseteq \text{Prop}$

Coast

$\text{Coast} \sqsubseteq \text{RuralRegion}$

Combat

$\text{Combat} \equiv \text{Event} \sqcap \forall\, \text{hasSubEvent}\, \text{CombatAct}$
$\text{Combat} \sqsubseteq \text{Conflict}$

CombatAct

$\text{CombatAct} \equiv \text{Act} \sqcap \exists\, \text{performs}\, \text{Attack}$
$\text{CombatAct} \sqsubseteq \text{Act}$

Commerce

$\text{Commerce} \sqsubseteq \text{Happening}$

CommercialBuilding

$\text{CommercialBuilding} \sqsubseteq \text{Building}$

CommercialOrganisation

CommercialOrganisation $\sqsubseteq$ Organisation

Conflict

Conflict $\sqsubseteq$ Happening

Container

Container $\sqsubseteq$ Prop

Contest

Contest $\sqsubseteq$ SocialEvent

Conversation

Conversation $\sqsubseteq \forall$ hasSubEvent (Act $\sqcap \exists$ performs Say)
Conversation $\sqsubseteq$ SocialEvent

Copse

Copse $\sqsubseteq$ WoodedRegion

Country

Country $\sqsubseteq$ Territory

CourtHouse

CourtHouse $\sqsubseteq$ CivilBuilding

Create

Create $\sqsubseteq$ Manipulate

Creature

$\text{Creature} \equiv \text{Individual} \sqcap \text{Object}$
$\text{Creature} \sqsubseteq \text{Individual}$
$\text{Creature} \sqsubseteq \exists\, \text{hasGender Gender}$
$\text{Creature} \sqsubseteq \exists\, \text{hasSpecies Species}$

Crime

$\text{Crime} \equiv \text{Act} \sqcap \exists\, \text{performs (Action} \sqcap \exists\, \text{isIllegalIn Region)}$
$\text{Crime} \sqsubseteq \text{Act}$
$\text{Crime} \sqsubseteq \text{Happening}$

Damage

$\text{Damage} \sqsubseteq \text{Manipulate}$

Desert

$\text{Desert} \sqsubseteq \text{RuralRegion}$

Destroy

$\text{Destroy} \sqsubseteq \text{Manipulate}$

Disaster

$\text{Disaster} \sqsubseteq \text{Happening}$

Dispute

$\text{Dispute} \sqsubseteq \text{Conflict}$

District

$\text{District} \sqsubseteq \text{UrbanRegion}$
$\text{District} \sqsubseteq \text{Settlement}$
$\text{District} \sqsubseteq \exists\, \text{isWithin City}$

Door

$\text{Door} \sqsubseteq \text{Furniture}$

Drop

$\text{Drop} \sqsubseteq \text{Unacquire}$

Duel

$\text{Duel} \equiv \text{Combat} \sqcap = \text{hasParticipant Actor}$
$\text{Duel} \sqsubseteq \text{Combat}$

Dwelling

$\text{Dwelling} \sqsubseteq \text{Region}$

Election

$\text{Election} \sqsubseteq \text{PoliticalEvent}$
$\text{Election} \sqsubseteq \text{Contest}$

ElevatedRegion

$\text{ElevatedRegion} \sqsubseteq \text{RuralRegion}$

Empire

$\text{Empire} \sqsubseteq \text{Territory}$

Event

$\text{Event} \sqsubseteq \text{Thing}$
$\text{Event} \sqsubseteq \exists \text{ occursAt Location}$
$\text{Event} \sqsubseteq = \text{startsAt}$
$\text{Event} \sqsubseteq = \text{endsAt}$

Examine

Examine $\sqsubseteq$ Manipulate

Exchange

Exchange $\sqsubseteq$ Take
Exchange $\sqsubseteq$ Give

Existent

Existent $\equiv$ Actor $\sqcup$ Setting
Existent $\sqsubseteq \exists$ hasName
Existent $\sqsubseteq$ Thing

ExpandInfluence

ExpandInfluence $\sqsubseteq$ PoliticalObjective

ExpandTerritory

ExpandTerritory $\sqsubseteq$ PoliticalObjective

ExpandTrade

ExpandTrade $\sqsubseteq$ PoliticalObjective

Explore

Explore $\sqsubseteq$ Search
Explore $\sqsubseteq$ Wander

Factory

Factory $\sqsubseteq$ IndustrialBuilding

FarmHouse

$\text{FarmHouse} \sqsubseteq \text{AgriculturalBuilding}$
$\text{FarmHouse} \sqsubseteq \text{House}$

Field

$\text{Field} \sqsubseteq \text{AgriculturalRegion}$

Follow

$\text{Follow} \sqsubseteq \text{Interact}$

Forest

$\text{Forest} \sqsubseteq \text{WoodedRegion}$

Fortress

$\text{Fortress} \sqsubseteq \text{Building}$

Furniture

$\text{Furniture} \sqsubseteq \text{Prop}$

GainExperience

$\text{GainExperience} \sqsubseteq \text{PersonalObjective}$

Gender

$\text{Gender} \equiv \{\text{male}\} \sqcup \{\text{female}\} \sqcup \{\text{neuter}\}$
$\text{Gender} \sqsubseteq \text{Thing}$

Give

$\text{Give} \sqsubseteq \text{Interact}$
$\text{Give} \sqsubseteq \text{Unacquire}$

GiveOrder

GiveOrder $\equiv$ Say $\sqcap$ $\exists$ hasMessage Order

Group

Group $\sqsubseteq$ Actor
Group $\sqsubseteq \geq 1$ hasMember Actor
Group $\sqsubseteq \neg$ Individual

Hamlet

Hamlet $\sqsubseteq$ Settlement

HandWeapon

HandWeapon $\sqsubseteq$ Weapon

Happening

Happening $\sqsubseteq$ Event

Heal

Heal $\sqsubseteq$ Interact

Hill

Hill $\sqsubseteq$ ElevatedRegion

Hotel

Hotel $\sqsubseteq$ Dwelling
Hotel $\sqsubseteq$ CommercialBuilding

House

House $\sqsubseteq$ Building
House $\sqsubseteq$ Dwelling

Hut

$\text{Hut} \sqsubseteq \text{House}$

Individual

$\text{Individual} \sqsubseteq \text{Actor}$
$\text{Individual} \sqsubseteq \neg\, \text{Group}$

IndustrialBuilding

$\text{IndustrialBuilding} \sqsubseteq \text{Building}$
$\text{IndustrialBuilding} \sqsubseteq \text{IndustrialRegion}$

IndustrialRegion

$\text{IndustrialRegion} \sqsubseteq \text{Region}$

Inn

$\text{Inn} \equiv \text{Hotel} \sqcap \text{Tavern}$

Interact

$\text{Interact} \equiv \text{Action} \sqcap \exists\, \text{hasTarget}\ \text{Actor}$
$\text{Interact} \sqsubseteq \text{Action}$

Jewellery

$\text{Jewellery} \sqsubseteq \text{Clothing}$
$\text{Jewellery} \sqsubseteq \text{Valuable}$

Key

$\text{Key} \sqsubseteq \text{Tool}$

Kill

$\text{Kill} \sqsubseteq \text{Attack}$

Kingdom

Kingdom $\sqsubseteq$ Territory

Lake

Lake $\sqsubseteq$ BodyOfWater

Leader

Leader $\equiv$ Individual $\sqcap \exists$ isLeaderOf Group
Leader $\sqsubseteq$ Individual

LearnSkill

LearnSkill $\sqsubseteq$ PersonalObjective

Location

Location $\equiv$ Region $\sqcup$ Position
Location $\sqsubseteq$ Thing

Lock

Lock $\sqsubseteq$ Manipulate

Manipulate

Manipulate $\equiv$ Action $\sqcap \exists$ hasObject Object
Manipulate $\sqsubseteq$ Action

Marsh

Marsh $\sqsubseteq$ BodyOfWater

Meadow

Meadow $\sqsubseteq$ Field

MedicalTool

MedicalTool $\sqsubseteq$ Tool

Message

Message $\sqsubseteq$ Thing

MilitaryBuilding

MilitaryBuilding $\sqsubseteq$ Building

MilitaryOrganisation

MilitaryOrganisation $\sqsubseteq$ Organisation

Mine

Mine $\sqsubseteq$ IndustrialRegion
Mine $\sqsubseteq$ SubterraneanRegion

Mission

Mission $\sqsubseteq \exists$ isGivenBy Actor
Mission $\sqsubseteq$ Plan

MissionObjective

MissionObjective $\equiv$ PlanObjective $\sqcap \exists$ isObjectiveOf (Step $\sqcap \exists$ isStepOf Mission)
MissionObjective $\sqsubseteq$ PlanObjective

Money

Money $\sqsubseteq$ Valuable

Mountain

Mountain $\sqsubseteq$ ElevatedRegion

Move

Move $\sqsubseteq$ Manipulate

NaturalDisaster

NaturalDisaster $\sqsubseteq$ Disaster

Object

Object $\sqsubseteq \exists$ hasLocation Location
Object $\sqsubseteq$ Setting

Objective

Objective $\sqsubseteq$ Thing

Observation

Observation $\sqsubseteq \exists$ of Existent
Observation $\sqsubseteq$ Event

Ocean

Ocean $\sqsubseteq$ BodyOfWater

Open

Open $\sqsubseteq$ Manipulate

Order

Order $\sqsubseteq$ Message
Order $\sqsubseteq$ Task

Organisation

Organisation $\sqsubseteq$ Team

Party

Party $\sqsubseteq$ SocialEvent

PersonalAction

PersonalAction $\sqsubseteq$ Action

PersonalEvent

PersonalEvent $\sqsubseteq$ Happening

PersonalObjective

PersonalObjective $\sqsubseteq$ Objective

Persuade

Persuade $\sqsubseteq$ Say

Pickup

Pickup $\sqsubseteq$ Acquire

Plain

Plain $\sqsubseteq$ RuralRegion

Plan

Plan $\sqsubseteq$ Thing
Plan $\sqsubseteq \exists$ hasStep Step

PlanObjective

PlanObjective $\equiv$ Objective $\sqcap \exists$ isObjectiveOf Step
PlanObjective $\sqsubseteq$ Objective

Plot

$\text{Plot} \sqsubseteq \text{Plan}$

PlotObjective

$\text{PlotObjective} \equiv \text{PlanObjective} \sqcap \exists\, \text{isObjectiveOf}\, (\text{Step} \sqcap \exists\, \text{isStepOf}\, \text{Plot})$
$\text{PlotObjective} \sqsubseteq \text{PlanObjective}$

PoliticalEvent

$\text{PoliticalEvent} \sqsubseteq \text{Happening}$

PoliticalObjective

$\text{PoliticalObjective} \sqsubseteq \text{Objective}$

Pond

$\text{Pond} \sqsubseteq \text{BodyOfWater}$

PortableStructure

$\text{PortableStructure} \sqsubseteq \text{Structure}$

Position

$\text{Position} \equiv \geq 1\ \text{hasCoordinate}$
$\text{Position} \sqsubseteq \text{Location}$
$\text{Position} \sqsubseteq \neg\, \text{Region}$

Post

$\text{Post} \sqsubseteq \text{Settlement}$

Prison

$\text{Prison} \sqsubseteq \text{Fortress}$
$\text{Prison} \sqsubseteq \text{CivilBuilding}$

Prop

Prop $\sqsubseteq$ Object

Proposition

Proposition $\sqsubseteq$ Message

Region

Region $\sqsubseteq$ Location
Region $\sqsubseteq$ Setting
Region $\sqsubseteq \neg$ Position

ReligiousBuilding

ReligiousBuilding $\sqsubseteq$ Building

Repair

Repair $\sqsubseteq$ Manipulate

Request

Request $\sqsubseteq$ Message

Resolution

Resolution $\sqsubseteq$ Happening

Rest

Rest $\sqsubseteq$ PersonalAction

River

River $\sqsubseteq$ BodyOfWater

Room

$\text{Room} \sqsubseteq \text{Structure}$
$\text{Room} \sqsubseteq \exists\, \text{isPartOf}\, \text{Building}$

RuralRegion

$\text{RuralRegion} \sqsubseteq \text{Region}$
$\text{RuralRegion} \sqsubseteq \neg\, \text{UrbanRegion}$

Say

$\text{Say} \sqsubseteq \exists\, \text{hasMessage}\, \text{Message}$
$\text{Say} \sqsubseteq \text{Interact}$

Sea

$\text{Sea} \sqsubseteq \text{BodyOfWater}$

Search

$\text{Search} \sqsubseteq \text{PersonalAction}$

Sell

$\text{Sell} \sqsubseteq \text{Give}$

SentientSpecies

$\text{SentientSpecies} \sqsubseteq \text{Species}$

Setting

$\text{Setting} \equiv \text{Object} \sqcup \text{Region}$
$\text{Setting} \sqsubseteq \text{Existent}$

Settlement

Settlement $\sqsubseteq$ Group
Settlement $\sqsubseteq$ Region

Sewer

Sewer $\sqsubseteq$ SubterraneanRegion
Sewer $\sqsubseteq$ UrbanRegion

Shop

Shop $\sqsubseteq$ CommercialBuilding

Shoreline

Shoreline $\sqsubseteq$ Coast

Shout

Shout $\sqsubseteq$ Say

Sit

Sit $\sqsubseteq$ PersonalAction

Skirmish

Skirmish $\equiv$ Combat $\sqcap \geq 3$ hasParticipant Actor $\sqcap \leq 9$ hasParticipant Actor
Skirmish $\sqsubseteq$ Combat

SocialEvent

SocialEvent $\sqsubseteq$ Happening

Species

Species $\sqsubseteq$ Thing

Stand

Stand $\sqsubseteq$ PersonalAction

Steal

Steal $\sqsubseteq$ Take

Step

Step $\sqsubseteq =$ hasSubObjective Objective
Step $\sqsubseteq$ Thing

StorageBuilding

StorageBuilding $\sqsubseteq$ Building

Story

Story $\sqsubseteq$ Thing
Story $\sqsubseteq \exists$ hasPlot Plot
Story $\sqsubseteq \exists$ hasSetting Setting
Story $\sqsubseteq \exists$ hasCharacter Actor
Story $\sqsubseteq \exists$ hasEpisode Event

Structure

Structure $\sqsubseteq$ Object
Structure $\sqsubseteq$ Region

SubterraneanRegion

SubterraneanRegion $\sqsubseteq$ Region

Swamp

Swamp $\sqsubseteq$ BodyOfWater

Take

$\text{Take} \sqsubseteq \text{Interact}$
$\text{Take} \sqsubseteq \text{Acquire}$

Task

$\text{Task} \equiv \text{Objective} \sqcap \exists \text{ toPerform Action}$

Taunt

$\text{Taunt} \sqsubseteq \text{Say}$

Tavern

$\text{Tavern} \sqsubseteq \text{CommercialBuilding}$

Team

$\text{Team} \equiv \text{Group} \sqcap \text{Agent}$

Tell

$\text{Tell} \equiv \text{Say} \sqcap \exists \text{ hasMessage Proposition}$
$\text{Tell} \sqsubseteq \text{Say}$

Temple

$\text{Temple} \sqsubseteq \text{ReligiousBuilding}$

Tent

$\text{Tent} \sqsubseteq \text{PortableStructure}$

Territory

$\text{Territory} \sqsubseteq \text{Region}$

Theft

$\text{Theft} \equiv \text{Act} \sqcap \exists \text{ performs Steal}$
$\text{Theft} \sqsubseteq \text{Crime}$

Thing

Threat

$\text{Threat} \sqsubseteq \text{Message}$

Threaten

$\text{Threaten} \equiv \text{Say} \sqcap \exists \text{ hasMessage Threat}$

ThrownWeapon

$\text{ThrownWeapon} \sqsubseteq \text{Weapon}$

Tool

$\text{Tool} \sqsubseteq \text{Prop}$

Town

$\text{Town} \sqsubseteq \text{Settlement}$

TownHall

$\text{TownHall} \sqsubseteq \text{CivilBuilding}$

Trade

$\text{Trade} \equiv \text{Act} \sqcap \exists \text{ performs (Buy} \sqcup \text{Exchange} \sqcup \text{Sell)}$
$\text{Trade} \sqsubseteq \text{Act}$
$\text{Trade} \sqsubseteq \text{Commerce}$

Travel

$\text{Travel} \sqsubseteq \text{PersonalAction}$

Tunnel

Tunnel $\sqsubseteq$ SubterraneanRegion

Unacquire

Unacquire $\sqsubseteq$ Manipulate

Unlock

Unlock $\sqsubseteq$ Manipulate

UrbanBuilding

UrbanBuilding $\equiv$ Building $\sqcap$ UrbanRegion
UrbanBuilding $\sqsubseteq$ UrbanRegion
UrbanBuilding $\sqsubseteq$ Building

UrbanRegion

UrbanRegion $\equiv$ Region $\sqcap \exists$ isWithin Settlement
UrbanRegion $\sqsubseteq$ Region
UrbanRegion $\sqsubseteq \neg$ RuralRegion

Valley

Valley $\sqsubseteq$ RuralRegion

Valuable

Valuable $\sqsubseteq$ Prop

Vehicle

Vehicle $\sqsubseteq$ Structure

Village

Village $\sqsubseteq$ Settlement

Wait

Wait $\sqsubseteq$ PersonalAction

Wander

Wander $\sqsubseteq$ PersonalAction

War

War $\equiv$ Event $\sqcap \exists$ hasSubEvent Battle
War $\sqsubseteq$ Disaster
War $\sqsubseteq \forall$ hasParticipant Territory
War $\sqsubseteq$ Conflict

Warehouse

Warehouse $\sqsubseteq$ StorageBuilding
Warehouse $\sqsubseteq$ CommercialBuilding

Warn

Warn $\equiv$ Say $\sqcap \exists$ hasMessage Warning

Warning

Warning $\sqsubseteq$ Message

Weapon

Weapon $\sqsubseteq$ Tool

Whisper

Whisper $\sqsubseteq$ Say

WoodedRegion

WoodedRegion $\sqsubseteq$ RuralRegion

A.3 Object properties

achieves

$\exists$ achieves Thing $\sqsubseteq$ Event
$\top \sqsubseteq \forall$ achieves Objective

after

$\sqsubseteq$ temporalRelation
after $\equiv$ before^{-}

authority

$\top \sqsubseteq \leq 1$ authority Thing
$\exists$ authority Thing $\sqsubseteq$ Territory
$\top \sqsubseteq \forall$ authority Actor

before

$\sqsubseteq$ temporalRelation
after $\equiv$ before^{-}

causes

$\sqsubseteq$ before
causes $\equiv$ isCausedBy^{-}
$\top \sqsubseteq \leq 1$ causes^{-} Thing
$\exists$ causes Thing $\sqsubseteq$ Event
$\top \sqsubseteq \forall$ causes Event

contains

$\sqsubseteq$ temporalRelation
contains $\equiv$ during^{-}

contains

$\text{isWithin} \equiv \text{contains}^{-}$
$\top \sqsubseteq \forall \text{ contains Location}$

dependsOn

$\exists \text{ dependsOn Thing} \sqsubseteq \text{Step}$
$\top \sqsubseteq \forall \text{ dependsOn Step}$

disjoint

$\sqsubseteq \text{temporalRelation}$

during

$\sqsubseteq \text{temporalRelation}$
$\text{contains} \equiv \text{during}^{-}$

finishedBy

$\sqsubseteq \text{temporalRelation}$
$\text{finishes} \equiv \text{finishedBy}^{-}$

finishes

$\sqsubseteq \text{temporalRelation}$
$\text{finishes} \equiv \text{finishedBy}^{-}$

forGroup

$\top \sqsubseteq \leq 1 \text{ forGroup Thing}$
$\exists \text{ forGroup Thing} \sqsubseteq \text{Election}$
$\top \sqsubseteq \forall \text{ forGroup Group}$

hasActor

$\top \sqsubseteq \leq 1 \text{ hasActor Thing}$

$\exists$ hasActor Thing $\sqsubseteq$ Act

$\top \sqsubseteq \forall$ hasActor Actor

hasCharacter

hasCharacter $\equiv$ isCharacterIn$^-$

$\exists$ hasCharacter Thing $\sqsubseteq$ Story

$\top \sqsubseteq \forall$ hasCharacter Actor

hasEpisode

$\exists$ hasEpisode Thing $\sqsubseteq$ Story

$\top \sqsubseteq \forall$ hasEpisode Event

hasGender

$\top \sqsubseteq \leq 1$ hasGender Thing

$\exists$ hasGender Thing $\sqsubseteq$ Creature

$\top \sqsubseteq \forall$ hasGender Gender

hasInhabitant

$\sqsubseteq$ hasMember

$\exists$ hasInhabitant Thing $\sqsubseteq$ Settlement

hasLawAgainst

hasLawAgainst $\equiv$ isIllegalIn$^-$

$\exists$ hasLawAgainst Thing $\sqsubseteq$ Region

$\top \sqsubseteq \forall$ hasLawAgainst Action

hasLeader

$\sqsubseteq$ hasMember

hasLeader $\equiv$ isLeaderOf$^-$

$\top \sqsubseteq \leq 1$ hasLeader Thing

$\exists$ hasLeader Thing $\sqsubseteq$ Group

$\top \sqsubseteq \forall$ hasLeader Individual

hasLocation

$\top \sqsubseteq \leq 1$ hasLocation Thing
$\exists$ hasLocation Thing $\sqsubseteq$ Object
$\top \sqsubseteq \forall$ hasLocation Location

hasMember

$\sqsubseteq$ hasPart
$\exists$ hasMember Thing $\sqsubseteq$ Group
$\top \sqsubseteq \forall$ hasMember Individual
$\top \sqsubseteq \forall$ hasMember Actor

hasMessage

$\top \sqsubseteq \leq 1$ hasMessage Thing
$\exists$ hasMessage Thing $\sqsubseteq$ Say
$\top \sqsubseteq \forall$ hasMessage Message

hasObject

$\exists$ hasObject Thing $\sqsubseteq$ Action
$\top \sqsubseteq \forall$ hasObject Object

hasObjective

$\exists$ hasObjective Thing $\sqsubseteq$ Actor
$\top \sqsubseteq \forall$ hasObjective Objective

hasPart

isPartOf $\equiv$ hasPart^{-}
$\exists$ hasPart Thing $\sqsubseteq$ Existent
$\top \sqsubseteq \forall$ hasPart Existent

hasParticipant

$\text{hasParticipant} \equiv \text{isParticipantIn}^{-}$

$\exists\, \text{hasParticipant Thing} \sqsubseteq \text{Event}$

$\top \sqsubseteq \forall\, \text{hasParticipant Actor}$

hasPlan

$\exists\, \text{hasPlan Thing} \sqsubseteq \text{Actor}$

$\top \sqsubseteq \forall\, \text{hasPlan Plan}$

hasPlot

$\top \sqsubseteq \leq 1\, \text{hasPlot Thing}$

$\exists\, \text{hasPlot Thing} \sqsubseteq \text{Story}$

$\top \sqsubseteq \forall\, \text{hasPlot Plot}$

hasReward

$\exists\, \text{hasReward Thing} \sqsubseteq \text{Mission}$

$\top \sqsubseteq \forall\, \text{hasReward Object}$

hasSetting

$\exists\, \text{hasSetting Thing} \sqsubseteq \text{Story}$

$\top \sqsubseteq \forall\, \text{hasSetting Setting}$

hasSpecies

$\top \sqsubseteq \leq 1\, \text{hasSpecies Thing}$

$\exists\, \text{hasSpecies Thing} \sqsubseteq \text{Creature}$

$\top \sqsubseteq \forall\, \text{hasSpecies Species}$

hasStep

$\text{hasStep} \equiv \text{isStepOf}^{-}$

$\exists\, \text{hasStep Thing} \sqsubseteq \text{Plan}$

$\top \sqsubseteq \forall\, \text{hasStep Step}$

hasSubEvent

$\exists$ hasSubEvent Thing $\sqsubseteq$ Event

$\top \sqsubseteq \forall$ hasSubEvent Event

hasSubObjective

hasSubObjective $\equiv$ isObjectiveOf^{-}

$\top \sqsubseteq \leq 1$ hasSubObjective Thing

$\exists$ hasSubObjective Thing $\sqsubseteq$ Step

$\top \sqsubseteq \forall$ hasSubObjective Objective

hasTarget

$\exists$ hasTarget Thing $\sqsubseteq$ Action

$\top \sqsubseteq \forall$ hasTarget Actor

hasWinner

$\top \sqsubseteq \leq 1$ hasWinner Thing

$\exists$ hasWinner Thing $\sqsubseteq$ Contest

$\top \sqsubseteq \forall$ hasWinner Actor

inCelebrationOf

$\sqsubseteq$ isCausedBy

$\top \sqsubseteq \leq 1$ inCelebrationOf Thing

$\exists$ inCelebrationOf Thing $\sqsubseteq$ Party

$\top \sqsubseteq \forall$ inCelebrationOf Event

inhabits

$\sqsubseteq$ isMemberOf

$\exists$ inhabits Thing $\sqsubseteq$ Individual

$\top \sqsubseteq \forall$ inhabits Settlement

isCausedBy

$\sqsubseteq$ after
causes $\equiv$ isCausedBy^{-}
$\top \sqsubseteq \leq 1$ isCausedBy Thing
$\exists$ isCausedBy Thing $\sqsubseteq$ Event
$\top \sqsubseteq \forall$ isCausedBy Event

isCharacterIn

hasCharacter $\equiv$ isCharacterIn^{-}
$\exists$ isCharacterIn Thing $\sqsubseteq$ Actor
$\top \sqsubseteq \forall$ isCharacterIn Story

isGivenBy

$\top \sqsubseteq \leq 1$ isGivenBy Thing
$\exists$ isGivenBy Thing $\sqsubseteq$ Mission
$\top \sqsubseteq \forall$ isGivenBy Actor

isIllegalIn

hasLawAgainst $\equiv$ isIllegalIn^{-}
$\exists$ isIllegalIn Thing $\sqsubseteq$ Action
$\top \sqsubseteq \forall$ isIllegalIn Region

isLeaderOf

$\sqsubseteq$ isMemberOf
hasLeader $\equiv$ isLeaderOf^{-}
$\top \sqsubseteq \leq 1$ isLeaderOf^{-} Thing
$\exists$ isLeaderOf Thing $\sqsubseteq$ Individual
$\top \sqsubseteq \forall$ isLeaderOf Group

isMemberOf

$\sqsubseteq$ isPartOf

$\exists$ isMemberOf Thing $\sqsubseteq$ Actor

$\top \sqsubseteq \forall$ isMemberOf Group

isObjectiveOf

hasSubObjective $\equiv$ isObjectiveOf^{-}

$\top \sqsubseteq \leq 1$ isObjectiveOf^{-} Thing

$\exists$ isObjectiveOf Thing $\sqsubseteq$ Objective

$\top \sqsubseteq \forall$ isObjectiveOf Step

isPartOf

isPartOf $\equiv$ hasPart^{-}

$\exists$ isPartOf Thing $\sqsubseteq$ Existent

$\top \sqsubseteq \forall$ isPartOf Existent

isParticipantIn

hasParticipant $\equiv$ isParticipantIn^{-}

$\exists$ isParticipantIn Thing $\sqsubseteq$ Actor

$\top \sqsubseteq \forall$ isParticipantIn Event

isProhibitedBy

prohibits $\equiv$ isProhibitedBy^{-}

$\exists$ isProhibitedBy Thing $\sqsubseteq$ Action

$\top \sqsubseteq \forall$ isProhibitedBy Group

isStepOf

hasStep $\equiv$ isStepOf^{-}

$\exists$ isStepOf Thing $\sqsubseteq$ Step

$\top \sqsubseteq \forall$ isStepOf Plan

isWithin

isWithin $\equiv$ contains^{-}

$\exists$ isWithin Thing $\sqsubseteq$ Location

livesIn

$\exists$ livesIn Thing $\sqsubseteq$ Individual
$\top \sqsubseteq \forall$ livesIn Dwelling

meets

$\sqsubseteq$ temporalRelation
metBy $\equiv$ meets$^-$

metBy

$\sqsubseteq$ temporalRelation
metBy $\equiv$ meets$^-$

occursAt

$\top \sqsubseteq \leq 1$ occursAt Thing
$\exists$ occursAt Thing $\sqsubseteq$ Event
$\top \sqsubseteq \forall$ occursAt Location

of

$\top \sqsubseteq \leq 1$ of Thing
$\exists$ of Thing $\sqsubseteq$ Observation
$\top \sqsubseteq \forall$ of Existent

overlappedBy

$\sqsubseteq$ temporalRelation
overlaps $\equiv$ overlappedBy$^-$

overlaps

$\sqsubseteq$ temporalRelation
overlaps $\equiv$ overlappedBy$^-$

performs

$\top \sqsubseteq \leq 1$ performs Thing
$\exists$ performs Thing $\sqsubseteq$ Act
$\top \sqsubseteq \forall$ performs Action

produces

$\exists$ produces Thing $\sqsubseteq$ IndustrialRegion
$\top \sqsubseteq \forall$ produces Object

prohibits

prohibits $\equiv$ isProhibitedBy^{-}
$\exists$ prohibits Thing $\sqsubseteq$ Group
$\top \sqsubseteq \forall$ prohibits Action

sells

$\exists$ sells Thing $\sqsubseteq$ Shop
$\top \sqsubseteq \forall$ sells Object

startedBy

$\sqsubseteq$ temporalRelation
starts $\equiv$ startedBy^{-}

starts

$\sqsubseteq$ temporalRelation
starts $\equiv$ startedBy^{-}

temporalRelation

$\exists$ temporalRelation Thing $\sqsubseteq$ Event
$\top \sqsubseteq \forall$ temporalRelation Event

to

$\top \sqsubseteq \leq 1$ to Thing
$\exists$ to Thing $\sqsubseteq$ Move
$\top \sqsubseteq \forall$ to Location

toAchieve

$\top \sqsubseteq \leq 1$ toAchieve Thing
$\exists$ toAchieve Thing $\sqsubseteq$ Plan
$\top \sqsubseteq \forall$ toAchieve Objective

toDestination

$\top \sqsubseteq \leq 1$ toDestination Thing
$\exists$ toDestination Thing $\sqsubseteq$ Travel
$\top \sqsubseteq \forall$ toDestination Location

toPerform

$\top \sqsubseteq \leq 1$ toPerform Thing
$\exists$ toPerform Thing $\sqsubseteq$ Task
$\top \sqsubseteq \forall$ toPerform Action

using

$\exists$ using Thing $\sqsubseteq$ Action
$\top \sqsubseteq \forall$ using Object

A.4 Data properties

atX

$\top \sqsubseteq \leq 1$ atX

atY

$\top \sqsubseteq \leq 1$ atY

atZ

$\top \sqsubseteq \leq 1$ atZ

endsAt

$\top \sqsubseteq \leq 1$ endsAt

hasContent

$\top \sqsubseteq \leq 1$ hasContent

hasCoordinate

hasName

$\top \sqsubseteq \leq 1$ hasName

startsAt

$\top \sqsubseteq \leq 1$ startsAt

A.5 Individuals

female

female : Gender
$\{\text{male}\} \not\equiv \{\text{female}\} \not\equiv \{\text{neuter}\}$

human

human : SentientSpecies

male

male : Gender
$\{\text{male}\} \not\equiv \{\text{female}\} \not\equiv \{\text{neuter}\}$

neuter

neuter : Gender
$\{\text{male}\} \not\equiv \{\text{female}\} \not\equiv \{\text{neuter}\}$

APPENDIX B

EXAMPLE PRESENTER OUTPUT

B.1 Introduction

This appendix shows a variety of sample inputs to the presenter capability of the system and then shows the output that is generated in each case. In order to keep the presentation short, the input definitions are presented in summary form, rather than fully describing each character, location and object using the full ontology. The output stories have been converted from the original Atom publishing format into appropriate LaTeX markup for incorporation into the thesis: the content of the reports is identical, only the format has been altered.

B.2 Combat

Combat events are by far the most common sorts of events that occur in the game we are narrating (*Neverwinter Nights*), and were by far the most common event that was actually reported in the live evaluation study. In this section we show some example reports that represent combat events and the resulting stories that are produced by the system for those inputs.

B.2.1 Skirmish

This report shows the input that generated the live study report shown in the evaluation (Figure 7.11). The input in symbolic form is as follows:

$$
\begin{aligned}
\mathsf{Skirmish}(e) \wedge\ & \mathsf{hasParticipant}(e, k) \\
\wedge\ & \mathsf{hasParticipant}(e, g) \\
\wedge\ & \mathsf{hasParticipant}(e, h) \\
\wedge\ & \mathsf{hasSubEvent}(e, s_1) \\
\wedge\ & \mathsf{hasSubEvent}(e, s_2) \\
\wedge\ & \mathsf{hasSubEvent}(e, s_3) \ldots \\
\mathsf{PlayerCharacter}(k) \wedge\ & \mathsf{hasName}(k, \text{`KaleVeuthian}') \ldots \\
\mathsf{Act}(s_1) \wedge\ & \mathsf{occursAt}(s_1, \text{`GalumHills}') \\
\wedge\ & \mathsf{startsAt}(s_1, t_1) \wedge \mathsf{endsAt}(s_1, t_2) \\
\wedge\ & \mathsf{hasActor}(s_1, g) \\
\wedge\ & \mathsf{performs}(s_1, a_1) \\
\wedge\ & \mathsf{Attack}(a_1) \\
\wedge\ & \mathsf{hasTarget}(a_1, k) \\
\wedge\ & \mathsf{using}(a_1, \text{`LightMace}') \\
\vdots\ &
\end{aligned}
$$

Which produces the output shown in Figure 7.11, duplicated here:

> **Skirmish in Galum Hills!**
>
> There was a battle in Galum Hills today, involving Kale Veuthian, a Goblin and a Hobgoblin.
>
> It all started when a Goblin attacked Kale Veuthian with their light mace. Then, Kale Veuthian attacked Goblin with their shortsword. The Goblin attacked Kale Veuthian with their light mace. Kale Veuthian attacked the Goblin with their shortsword. The Goblin attacked Kale Veuthian with their morningstar. Kale Veuthian attacked the Goblin with their shortsword. The Goblin was slain by Kale Veuthian. Kale Veuthian attacked a Hobgoblin with their shortsword. The Hobgoblin attacked Kale Veuthian with their longsword. Kale Veuthian attacked the Hobgoblin with their shortsword. The Hobgoblin attacked Kale Veuthian with their longsword. Finally, the Hobgoblin was slain by Kale Veuthian.

Most battles follow the same format, leading to a mostly blow-by-blow account of each separate combat event.

B.2.2 Assassination

A minor form of combat action is an assassination, which is recognised when a leader of some group (as defined in the ontology) is killed. Such events produce brief summary outputs as separate reports, with the assumption that the full battle in which the event occurred will likely appear as a separate battle report. This example shows how an assassination event is represented and narrated. The input:

$$
\begin{aligned}
\mathsf{Assassination}(e) &\wedge \mathsf{hasActor}(e, k) \\
&\wedge \mathsf{performs}(e, a_1) \\
&\wedge \mathsf{occursAt}(e, \text{`}\mathsf{EtumCastleDistrict}') \\
&\wedge \mathsf{startsAt}(e, t_1) \wedge \mathsf{endsAt}(e, t_2) \\
\mathsf{Kill}(a_1) &\wedge \mathsf{hasTarget}(a_1, m) \\
&\wedge \mathsf{using}(a_1, \text{`}\mathsf{ShortSword}') \\
\mathsf{Character}(m) &\wedge \mathsf{hasName}(m, \text{`}\mathsf{MarinSigerCaptainoftheGuards}') \\
&\wedge \mathsf{isLeaderOf}(m, \text{`}\mathsf{RoyalGuards}') \\
&\vdots
\end{aligned}
$$

Produces the following output:

> **Assassination in Etum Castle District!**
>
> Marin Siger Captain of the Guards was assassinated in Etum Castle District today by Kale Veuthian.
>
> The leader of Etum Royal Guards was assassinated in Etum Castle District today by Kale Veuthian, a novice fighter. Kale struck down Marin Siger Captain of the Guards with their shortsword.

B.3 Achievements

Aside from combat, the only other event that was reported in the live evaluation study was an achievement event: a character 'levelled up' (i.e., gained a new experience level).

This event is part of a general class of 'achievement' events represented in the ontology. Achievement events achieve some objective, of which levelling-up is one.
The input that generated this event was as follows:

$$\begin{aligned}
\mathsf{Achievement}(e) &\wedge \mathsf{achieves}(e, o) \\
&\wedge \mathsf{hasActor}(e, h) \\
&\wedge \mathsf{occursAt}(e, x) \ldots \\
\mathsf{LevelUp}(o) &\wedge \mathsf{inCharacterClass}(o, \mathsf{fighter}) \\
&\wedge \mathsf{toLevel}(o, 2) \\
\mathsf{PlayerCharacter}(h) &\wedge \mathsf{hasName}(h, \text{`}\mathsf{HellaKendon}') \\
&\vdots
\end{aligned}$$

This led to the following story being presented on the website:

> **Hella Kendon has levelled up!**
> Hella Kendon, a female fighter, has achieved a new level. Hella is now a level 2 fighter! Congratulations!

B.4 Quests

The most story-like of the different events that are reported on in the current implementation are the various quests that are available to players in the game. A quest is simply a mission given by a non-player-character (NPC) to a player. Typically, quests in *Neverwinter Nights* are short, involving fetching some magical artefact or slaying a monster. In the single-player version of the game, more complex, multi-part quests are available that lead the player through a story-line, but these are largely absent from the multi-player persistent world we used.
This example is typical of the majority of the quests that are present in the *Rhun* module that was used for evaluation of the system. Such quests typically consist of a simple conversation with a non-player character (NPC) who describes the quest. The player then has to travel to a certain region, collect a particular item, and then return it to the original NPC to receive a reward. In this case, the quest is given by a character named 'Pata Brows', a female human in Etum Castle District. Pata requires the player to fetch a magical fairy from Galum Forest. In return the player receives a magical amulet as a reward (plus some

experience points). The input that would be generated on successfully recognising this quest would be as follows (some details elided):

$$
\begin{aligned}
\mathsf{Achievement}(e) &\wedge \mathsf{achieves}(e, o) \\
&\wedge \mathsf{hasActor}(e, k) \\
&\wedge \mathsf{occursAt}(e, x) \ldots \\
\mathsf{CompletedQuest}(o) &\wedge \mathsf{quest}(o, q) \\
\mathsf{Quest}(q) &\wedge \mathsf{isGivenBy}(q, p) \\
&\wedge \mathsf{hasReward}(q, \mathsf{AmuletOfIntellect}) \\
&\wedge \mathsf{hasStep}(q, s_1) \\
&\wedge \mathsf{hasStep}(q, s_2) \\
&\ldots \\
\mathsf{Step}(s_1) &\wedge \mathsf{hasSubObjective}(s_1, o_1) \wedge \ldots \\
\mathsf{QuestObjective}(o_1) &\wedge \mathsf{toPerform}(o_1, a_1) \\
\mathsf{Acquire}(a_1) &\wedge \mathsf{hasObject}(a_1, \mathsf{Fairy}) \\
&\vdots
\end{aligned}
$$

This produces the following output:

> **Kale Veuthian has completed a quest!**
>
> Kale Veuthian completed a quest today in Etum Castle District! Congratulations!
>
> It all started when Kale Veuthian spoke to Pata Brows, a female human, in Etum Castle District. Pata Brows asked Kale Veuthian to embark on a quest to fetch a magical fairy from Galum Forest. Kale Veuthian travelled to Galum Forest, encountering many difficulties on his travels. Kale Veuthian then captured a magical fairy in the forest. Finally, Kale Veuthian gave the magical fairy to Pata Brows, receiving Amulet of Intellect as a reward.

Bibliography

[1] The Atom Publishing Protocol. Technical Report RFC 5023, The Internet Engineering Task Force (IETF), October 2007.

[2] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[3] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[4] James F. Allen and George Ferguson. Actions and events in interval temporal logic. *Journal of Logic and Computation*, 4(5):531–579, 1994.

[5] Aristotle. *Poetics*. The Internet Web Classics Archive, `http://classics.mit.edu/Aristotle/poetics.html`, circa 350 BC. Translated by S. H. Butcher.

[6] Alessandro Artale and Enrico Franconi. A survey of temporal extensions of description logics. *Annals of Mathematics and Artificial Intelligence*, 30:171–210, 2000.

[7] Franz Baader, Diego Calvanese, Deborah McGuiness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[8] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics. In van Harmelen et al. [116], chapter 3, pages 135–180.

[9] Franz Baader and Werner Nutt. Basic description logics. In Baader et al. [7], chapter 2, pages 47–100.

[10] Rebecca Barr, Michael L. Kamil, Peter Mosenthal, and P. David Pearson, editors. volume II. Lawrence Erlbaum Associates, Mahwah, New Jersey, USA, 1991.

[11] Rafael H. Bordini, J. F. Hübner, and R. Vieira. Jason and the golden fleece of agent-oriented programming. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, chapter 1, pages 3–37. Springer-Verlag, 2005.

[12] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons Ltd, 2007.

[13] Ronald J. Brachman and Hector J. Levesque. *Knowledge Representation and Reasoning*. Elsevier/Morgan Kaufmann, San Francisco, CA, USA, 2004.

[14] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355, 1988.

[15] Michael Bratman. Two faces of intention. *The Philosophical Review*, 93(3):375–405, 1984.

[16] Michael Bratman. *Intention, Plans, and Practical Reason*. Center for the Study of Language and Information — The David Hume Series. The University of Chicago Press, 1987.

[17] Rodney A. Brooks. A robust layered control system for a mobile robot. AI Memo 864, Massachusetts Institute of Technology Artificial Intelligence Laboratory, September 1985.

[18] Rodney A. Brooks. Intelligence without reason. In John Myopoulos and Ray Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595, 1991.

[19] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:140–159, 1991.

[20] Brian Cantwell Smith. Reflection and semantics in LISP. In *Proceedings of 11th Annual Symposium on Principles of Programming Languages*, pages 23–35, 1984.

[21] Sandra Carberry. Techniques for plan recognition. *User Modeling and User-Adapted Interaction*, 11(1–2):31–48, 2001.

[22] Cristiano Castelfranchi. Modelling social action for AI agents. *Artificial Intelligence*, 103:157–182, 1998.

[23] Stefana Ceri, Georg Gottlob, and Letizia Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1, 1989.

[24] Seymour Chatman. Towards a theory of narrative. *New Literary History*, 6(2):295–318, 1975.

[25] Seymour Chatman. *Story and Discourse. Narrative Structure in Fiction and Film*. Cornell University Press, 1978.

[26] Edgar F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[27] Edgar F. Codd. *The relational model for database management: version 2*. Addison-Wesley Longman, Boston, MA, USA, 1990.

[28] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.

[29] Philip R. Cohen and Hector J. Levesque. Teamwork. *Noûs*, 25(4):487–512, 1991. Special Issue on Cognitive Science and Artificial Intelligence.

[30] Philip R. Cohen, Hector R. Levesque, and Ira Smith. On team formation. In *Contemporary Action Theory, Volume 2: Social Action*, pages 87–114. Synthese Library, Springer, 1997.

[31] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.

[32] R. Coomber. Using the Internet for survey research. *Sociological Research Online*, 2(2), 1997. `http://www.socresonline.org.uk/2/2/2.html`.

[33] Mehdi Dastani. 2APL: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16:214–248, 2008.

[34] Randall Davis and Reid G. Smith. Negotiation as metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–100, 1983.

[35] Daniel C. Dennett. *The Intentional Stance*. MIT Press, 1989.

[36] Ian Dickinson and Mike Wooldridge. Towards practical reasoning agents for the semantic web. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-03)*, Melbourne, Australia, July 2003.

[37] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. $\mathcal{AL}$-Log: integrating datalog and description logics. *Journal of Intelligent Information Systems*, 10(3):1–27, 1998.

[38] Thomas Eiter, Giobambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. Combining answer set programming with description logics for the semantic web. *Artificial Intelligence*, 172(12–13):1495–1539, 2008.

[39] E. A. Emerson and Srinivasan J. Branching time temporal logic. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 123–172. Springer-Verlag, Berlin, 1989.

[40] Oren Etzioni. Intelligence without robots: A reply to Brooks. *AI Magazine*, 14(4):7–13, 1993.

[41] Dan Fielding, Mike Fraser, Brian Logan, and Steve Benford. Extending game participation with embodied reporting agents. In *Proceedings of the ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE 2004)*, Singapore, June 2004. ACM Press.

[42] FIPA. FIPA contract net interaction protocol specification. Technical Report SC000029H, The Foundation for Intelligent Physical Agents, 2002.

[43] R. James Firby. Task networks for controlling continuous processes. In Kristian J. Hammond, editor, *Proceedings of the Second International Conference on AI Planning Systems*, pages 49–54, Chicago IL, 1994. AAAI.

[44] Antony Galton. Temporal logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Fall 2008.

[45] Gamebots interface to *Unreal Tournament*. `http://www.gamebots.org/`.

[46] Erann Gat. On three-layer architectures. In David Kortenkamp, R. Peter Bonnasso, and Robin Murphy, editors, *Artificial Intelligence and Mobile Robots*. AAAI Press, 1997.

[47] Michael Gelfond. Answer sets. In van Harmelen et al. [116], chapter 7, pages 285–316.

[48] Gerard Genette. *Narrative Discourse: An Essay in Method*. Cornell University Press, Ithaca, NY, USA, 1980. Trans. Jane E. Lewin.

[49] Bernardo Cuenca Grau and Boris Motik. OWL 2 web ontology language: Model-theoretic semantics. Technical report, W3C, April 2008. Working Draft.

[50] Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996.

[51] Barbara J. Grosz and Sarit Kraus. The evolution of SharedPlans. In Mike Wooldridge and Anand Rao, editors, *Foundations of Rational Agency*, pages 227–262. Kluwer Academic, Boston, MA, 1999.

[52] Nicola Guarino and Christopher Welty. Evaluating ontological decisions with OntoClean. *Communications of the ACM*, 45(2):61–65, 2002.

[53] Nicola Guarino and Christopher Welty. An overview of OntoClean. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, chapter 8, pages 151–172. Springer, 2004.

[54] Charlotte Herzeel, Pascal Costanza, and Theo D'Hondt. Reflection for the masses. In *Proceedings of the Workshop on Self-sustaining Systems (S3 2008)*, pages 87–122, Potsdam, Germany, May 2008.

[55] John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and their Relation to Automata*. Addison-Wesley, 1969.

[56] Ian Horrocks. OWL: A description logic based ontology language. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP 2005)*, pages 5–8. Springer, 2005.

[57] Ian Horrocks, Oliver Kutz, and Uli Sattler. The even more irresistible SROIQ. In *Proceedngs of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67. AAAI Press, 2006.

[58] Ian Horrocks, Peter F. Patel-Schneider, Sean Bechhofer, and Dmitry Tsarkov. OWL rules: A proposal and prototype implementation. *Journal of Web Semantics*, 3(1):23–40, 2005.

[59] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reducing $\mathcal{SHIQ}^-$ description logic to disjunctive datalog programs. In *Proceedings of the 9th International Conference on Knowledge Representation and Reasoning (KR 2004)*, pages 152–162, Whistler, Canada, June 2004.

[60] François F. Ingrand, Michael P. Georgeff, and Anand S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 7(6):33–44, 1992.

[61] N. R. Jennings. Specification and implementation of a belief desire joint-intention architecture for collaborative problem solving. *Journal of Intelligent and Cooperative Information Systems*, 2(3):289–318, 1993.

[62] Nick R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75:195–240, 1995.

[63] Henry A. Kautz. *A Formal Theory of Plan Recognition*. PhD thesis, University of Rochester, Rochester, NY, May 1987. Tech. Report TR 215.

[64] Henry A. Kautz. A formal theory of plan recognition and its implementation. In *Reasoning About Plans*, chapter 2, pages 69–126. Morgan Kaufmann Publishers, San Mateo, CA, USA, 1991.

[65] Thomas Klapiscak and Rafael H. Bordini. JASDL: A practical programming approach combining agent and semantic web technologies. In Matteo Baldoni, Tran Cao Son, M. Birna van Riemsdijk, and Michael Winikoff, editors, *Proceedings of the 6th International Workshop on Declarative Agent Languages and Technologies (DALT 2008)*, pages 45–62, Estoril, Portugal, May 2008.

[66] Robert Kowalski. Database updates in the event calculus. *Journal of Logic Programming*, 12:121–146, 1992.

[67] Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, February 1986.

[68] Sanjeev Kumar and Philip R. Cohen. Towards a fault-tolerant multi-agent system architecture. In *Proceedings of the Fourth International Conference on Autonomous Agents (Agents 2000)*, Barcelona, Spain, June 2000.

[69] Sanjeev Kumar and Philip R. Cohen. STAPLE: An agent programming language based on the joint intention theory. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004)*, New York, New York, USA, July 2004. ACM Press.

[70] Sanjeev Kumar, Philip R. Cohen, and Marcus J. Huber. Direct exception of team specifications in STAPLE. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002)*, Bologna, Italy, July 2002. ACM Press.

[71] Sanjeev Kumar, Philip R. Cohen, and Hector J. Levesque. The adaptive agent architecture: Achieving fault-tolerance using persistent broker teams. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS '00)*, page 159, 2000.

[72] Wendy G. Lehnert. Plot units and narrative summarization. *Cognitive Science*, 4:293–331, 1981.

[73] Hector J. Levesque, Philip R. Cohen, and José H. T. Nunes. On acting together. In *Proceedings of the Annual Meeting of the American Association for Artificial Intelligence, AAAI-90*, pages 94–99, 1990.

[74] Brian Logan, Mike Fraser, Dan Fielding, Steve Benford, Chris Greenhalgh, and Pilar Herrero. Keeping in touch: Agents reporting from collaborative virtual environments. In Ken Forbus and Magy Seif El-Nasr, editors, *Artificial Intelligence and Interative Entertainment: Papers from the 2002 AAAI Symposium*, pages 62–68. AAAI Press, March 2002. Technical Report SS–02–01.

[75] Neil Madden and Brian Logan. Collaborative narrative generation in persistent virtual environments. In *Proceedings of the AAAI Fall Symposium on Intelligent Narrative Technologies*, Arlington, Virginia, USA, November 2007.

[76] Clare Madge. Online questionnaires: Sampling issues. Available from `http://www.geog.le.ac.uk/orm/questionnaires/quessampling.htm` [Accessed: 17.07.07].

[77] Clare Madge. Online research ethics. Available from `http://www.geog.le.ac.uk/orm/ethics/ethcontents.htm` [Accessed: 17.07.07].

[78] Claudio Masolo, Stefano Borgo, Aldo Gangemi, Nicola Guarino, Alessandro Oltramari, and Luc Schneider. The WonderWeb library of foundational ontologies. preliminary report. Technical Report D17, WonderWeb Deliverable, 2005.

[79] John McCarthy. Actions and other events in situation calculus. In *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, Toulouse, France, April 2002.

[80] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[81] Jing Mei, Zuoquan Lin, Harold Boley, Jie Li, and Virendrakumar C. Bhavsar. The $Datalog_{DL}$ combination of deduction rules and description logic. *Computational Intelligence*, 23(3), 2007.

[82] Marvin Minsky. A framework for representing knowledge. In P. H. Winston, editor, *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, New York, 1975.

[83] Nick Montfort. Natural language generation and narrative variation in interactive fiction. In *Proceedings of the AAAI Computational Aesthetics Workshop*, 2006.

[84] Nick Montfort. *Generating Narrative Variation in Interactive Fiction*. PhD thesis, University of Pennsylvania, 2007.

[85] Nick Montfort. Ordering events in interactive fiction narratives. In *Proceedings of the AAAI Fall Symposium on Intelligent Narrative Technologies*, 2007.

[86] Álvaro F. Moreira, Renata Vieira, Rafael H. Bordini, and Jomi Hübner. Agent-oriented programming with underlying ontological reasoning. In *Proceedings of the 3rd International Workshop on Declarative Agent Languages and Technologies (DALT)*, pages 132–147, Utrecht, Netherlands, July 2005.

[87] K. L. Myers. A procedural knowledge approach to task-level control. In *Proceedings of the Third International Conference on AI Planning Systems*, 1996.

[88] Ranjit Nair, Milind Tambe, and Stacy Marsella. Role allocation and reallocation in multiagent teams: Towards a practical analysis. In *Proceedings of the Second International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS '03)*, pages 552–559, Melbourne, Australia, July 2003.

[89] Allen Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.

[90] Ian Niles and Adam Pease. Towards a standard upper ontology. In *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS)*, pages 2–9, Ogunquit, Maine, USA, October 2001.

[91] Nils J. Nilsson. Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158, 1994.

[92] Nils J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann, San Francisco, CA, 1998.

[93] Vladimir Propp. *Morphology of the Folktale*. University of Texas Press, 2nd edition, 1968. Translated by Laurence Scott.

[94] D. V. Pynadath, M. Tambe, N. Chauvat, and L. Cavedon. Toward team-oriented programming. In N. R. Jennings and Y. Lespérance, editors, *Intelligent Agents VI: Agent Theories, Architectures and Languages*, pages 233–247. Springer-Verlag, 1999.

[95] Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *MAAMAW '96: Proceedings of the 7th European workshop on modelling autonomous agents in a multi-agent world*, pages 42–55. Springer-Verlag, 1996.

[96] Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-architecture. In R. Fikes and E. Sandewall, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-91)*, pages 473–484, 1991.

[97] Anand S. Rao and Michael P. Georgeff. BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, San Francisco, USA, June 1995.

[98] Anand S. Rao and Michael P. Georgeff. Decision procedures for BDI logics. *Journal of Logic and Computation*, 8(3):293–344, 1998.

[99] Ehud Reiter and Robert Dale. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87, 1997.

[100] Shlomith Rimmon-Kenan. *Narrative Fiction: Contemporary Poetics*. Routledge, 1983.

[101] Peter Van Roy. *Can Logic Programming Execute as Fast as Imperative Programming?* PhD thesis, University of California Berkeley, December 1990. Tech Report UCB/CSD 90/600.

[102] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, New Jersey, 1995.

[103] Fernando Sáenz-Pérez. Datalog educational system. user's manual. Technical Report 139-04, Faculty of Computer Science, Universidad Complutense de Madrid, 2004. Available from `http://des.sourceforge.net`.

[104] Ken Samuel, Leo Obrst, Suzette Stoutenberg, Karen Fox, Paul Franklin, Adrian Johnson, Ken Laskey, Deborah Nichols, Steve Lopez, and Jason Peterson. Translating OWL and semantic web rules into prolog: Moving toward description logic programs. *Theory and Practice of Logic Programming*, Forthcoming, 2008.

[105] C. F. Schmidt, N. S. Sridharan, and J. L. Goodson. The plan recognition problem: An intersection of psychology and artificial intelligence. *Artificial Intelligence*, 11(1,2):45–83, August 1978.

[106] Murray Shanahan. The event calculus explained. In *Artificial Intelligence Today: Recent Trends and Developments (LNAI 1600)*, pages 409–430. Springer, 1999.

[107] Aaron Sloman and Brian Logan. Architectures and tools for human-like agents. In *Proceedings of the Second European Conference on Cognitive Modelling, ECCM-98.*, 1998.

[108] Aaron Sloman and Brian Logan. Building cognitively rich agents using the SIM_AGENT toolkit. *Communications of the ACM*, 42(3):71–77, 1999.

[109] Reid G. Smith. The contract net: A formalism for the control of distributed problem solving. In R. Reddy, editor, *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI)*, page 472, Cambridge, MA, August 1977.

[110] Neil Sorens. Stories from the sandbox. *Gamasutra*, 2008. `http://www.gamasutra.com/view/feature/3539/stories_from_the_sandbox.php?print=1` (checked 15th February 2008).

[111] Ella Tallyn, Boriana Koleva, Brian Logan, Dan Fielding, Steve Benford, Giulia Gelmini, and Neil Madden. Embodied reporting agents as an approach to creating narratives from live virtual worlds. In *Lecture Notes in Computer Science, Proceedings of Virtual Storytelling 2005*, Strasbourg, France, November 2005. Springer.

[112] M. Tambe, D. V. Pynadath, N. Chauvat, A. Das, and G. Kaminka. Adaptive agent integration architectures for heterogenous team members. In *Proceedings of Fourth International Conference on Multi-Agent Systems (ICMAS-2000)*, pages 301–308, 2000.

[113] Milind Tambe. Agent architectures for flexible, practical teamwork. In *Proceedings of National Conference on Artificial Intelligence (AAAI-97)*, pages 22–28, 1997.

[114] Milind Tambe, Wei-Min Shen, Maja Mataric, David V. Pynadath, Dani Goldberg, Pragnesh Jay Modi, Zhun Qiu, and Behnam Salemi. Using TEAMCORE to make agents team-ready. In *Proceedings of the AAAI Spring Symposium on Intelligent Agents in Cyberspace*, pages 136–141, 1999.

[115] Milind Tambe and Weixiong Zhang. Towards flexible teamwork in persistent teams. *Autonomous Agents and Multi-Agent Systems*, 3:159–183, 2000.

[116] Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors. *Handbook of Knowledge Representation*. Elsevier, 2007.

[117] Raphael Volz. *Web Ontology Reasoning with Logic Databases*. PhD thesis, Universität Karlsruhe, 2004.

[118] Raphael Volz, Stefan Decker, and Daniel Oberle. Bubo - implementing OWL in rule-based systems. In *Proceedings of WWW 2003*, Budapest, Hungary, May 2003.

[119] Robert Weida. Knowledge representation for plan recognition. In *Proceedings of the IJCAI-95 Workshop on the Next Generation of Plan Recognition Systems*, pages 119–123, Montreal, Canada, 1995.

[120] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2002.

[121] Mike J. Wooldridge and Nick R. Jennings. The cooperative problem solving process. *Journal of Logic and Computation*, 9(4):563–592, 1999.

[122] John Yen, Jianwen Yin, Thomas R. Ioerger, Michael S. Miller, Dianxiang Xu, and Richard A. Volz. CAST: collaborative agents for simulating teamwork. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1135–1144, 2001.

[123] R. Michael Young. An overview of the Mimesis architecture: Integrating intelligent narrative control into an existing gaming environment. In *Working Notes of the AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, Stanford, CA, March 2001. AAAI Press.

[124] José P. Zagal, Michael Mateas, Clara Fernández-Vara, Brian Hochhalter, and Nolan Lichti. Towards an ontological language for game analysis. In *Proceedings of DiGRA 2005 Conference: Changing Views — Worlds in Play*, Vancouver, Canada, June 2005.