Autonomous Agents and Multi-Agent Systems manuscript No.
(will be inserted by the editor)

A Semantic Approach to Situational Awareness for Intelligent Virtual Agents

Surangika Ranathunga · Stephen Cranefield · Julian Padget · JeeHang Lee

Received: date / Accepted: date

Abstract The use of intelligent agent systems, such as those based on the BDI architecture, is a promising way of implementing believable and intelligent virtual agents. However, the information representation gap that exists between these agent systems and virtual worlds poses a major challenge for improving the awareness of the agent about the dynamic environment around it. There are two facets to this problem: first and foremost, there is no common terminology or ontology to describe the dynamism observed in a virtual environment; second, the heterogeneous nature of the sensor data interfaces provided by virtual worlds, and the

Surangika Ranathunga
Department of Information Science
University of Otago
PO Box 56
Dunedin 9054
New Zealand
Tel.: +64 3 479 8317
Fax: +64 3 479 8311
E-mail: surangika@infoscience.otago.ac.nz

Stephen Cranefield
Department of Information Science
University of Otago
PO Box 56
Dunedin 9054
New Zealand
Tel.: +64 3 479 8083
Fax: +64 3 479 8311
E-mail: scranefield@infoscience.otago.ac.nz

Julian Padget
Department of Computer Science
University of Bath
Bath BA2 7AY, United Kingdom
E-mail: j.a.padget@bath.ac.uk

JeeHang Lee
Department of Computer Science
University of Bath
Bath BA2 7AY, United Kingdom
E-mail: j.lee@bath.ac.uk

different information requirements of virtual agents deployed in different application domains make it difficult to implement a sufficiently generic data processing framework to bridge this information representation gap.

The purpose of this paper is to provide solutions to both these problems. For the first, we provide a dynamic virtual environment ontology in OWL 2, and illustrate how it can be extended with ontologies specific to a virtual world and an application domain (with specific examples for the popular multi-purpose virtual world Second Life and the SecondFootball simulation within it). For the second, we propose a loosely-coupled data processing framework that allows different sensory data processing modules to be used depending on the needs of the agent platform and the virtual world used, and we illustrate this in the specific context of deploying Jason BDI agents in Second Life. We illustrate the flexibility of our approach by showing how additional sensor data processing can be initiated at run time—in particular, to monitor for the progress of durative actions, and we present an evaluation of the communication cost of our sensory data processing architecture.

Keywords Intelligent Virtual Agents · Multi-Agent Systems · Ontology · Middleware

1 Introduction

Intelligent Virtual Agents (IVAs) are interactive characters in 3D virtual worlds controlled by software agents that exhibit human-like behaviours. They are increasingly being used in many virtual world applications. For example, in Massively Multi-player Online Role-Playing Games (MMORPGs) and virtual storytelling applications as Non-Player Characters (NPCs) [44, 53, 47], in multi-purpose virtual worlds to assist and provide information to visitors [73, 79, 36, 46], for simulating different cultures [5], in virtual training applications [39, 32], and as passive observers to monitor human participant behaviour in virtual communities [23, 8].

When interacting with human participants in these different virtual world applications, an IVA is expected to be intelligent, as well as believable. According to the research in the area of believable agents [45], as well as in intelligent agents [78], an important factor that determines the intelligence and believability of an IVA is its capacity to be aware of its environment.

Awareness is a broad and complex subject that encompasses many aspects. While humans have the ability to be subconsciously aware of many different things in the surrounding environment, implementation of adequate situational awareness in an IVA is non-trivial. To be able to act both intelligently—or at least for its actions to be interpreted as intelligent—and believably, an IVA needs awareness both of itself and of its surrounding environment. Furthermore, both for an IVA and for a human participant, environment awareness involves not only the physical aspects of that environment, but needs to extend to the social environment.

With regard to physical aspects, an IVA needs to be capable of (i) perceiving the entities in its environment, in order to reason about how to interact with them, and (ii) interpreting the interactions (of others) taking place in its environment. The latter kind of awareness can be characterised by the identification of abstract environment states, recognizing complex domain-specific events that take place, and being aware of unfolding situations.

However, when using agent systems such as those based on the BDI (Belief-Desire-Intention) architecture to implement IVAs, making the agent aware of its changing virtual environment is not trivial for two reasons. Firstly, the sensor data interfaces provided by most virtual worlds present only a crude picture of the virtual environment, normally providing information only at the level of individual entities. Abstract information concerning relationships between entities, complex events taking place, and situational change, may be hidden in this low-level sensor data, or may be inferrable only with the addition of domain- and application-specific knowledge, making it unavailable for the agent's deliberation process by direct means. Secondly, cognitive agent systems such as those based on the BDI architecture, are not well-suited to abstracting environment information from streams of low-level virtual world sensor data.

This, in essence, is the problem of the information representation gap between agent systems and virtual worlds, as already acknowledged by many researchers [33,16,17,25]. For example, consider a virtual football game consisting of two teams of avatars and a ball object. The low-level information received from a virtual world contains individual entity states and/or primitive events corresponding to changes in the properties associated with these entities. For an intelligent agent designed to work with abstract symbolic information, the amount of reasoning that could be done with low-level sensor data, such as player A is at position (x, y) with linear and angular velocities v and θ, is minimal. Instead, the agent needs to be presented with domain-level information, such as whether a player scored a goal, successfully passed the ball to a team member, or successfully tackled an opponent. At an even higher abstraction level, the agent would want to know whether its team members are adhering to any team tactic that they have agreed upon.

Despite this information representation gap, researchers have identified many advantages of using the already established agent architectures, those based on BDI in particular, in implementing IVAs [17,72,48]. Therefore it is worth exploring ways to bridge this gap, in order to use agent systems to create IVAs with human-like capabilities. It is possible to include the logic needed to create these high-level representations in the BDI agent systems itself. However, implementing such logic in BDI systems is likely to be inefficient as well as leading to complicated and difficult to maintain plan triggers. Ziafati et al. [83] express similar concerns, noting that existing BDI-based agent programming languages do not allow for processing events concurrently with agent deliberation cycles, the use of efficient optimised data structures for event recognition, correct handling of delayed event perception, and ease of programming event-handling code.

The aim of this paper is to examine how to bridge this information representation gap so as to make an IVA aware of the dynamism that exists in the environment in which it is situated at an appropriate level of abstraction. In this context, we identify two facets to the problem of bridging the gap:

1. The sensor data interfaces exposed to agent systems by current virtual worlds are of a heterogeneous nature. Currently there exists no common terminology or ontology to describe this virtual-world-specific sensor data in a domain-specific abstract form, with respect to describing dynamism in a virtual environment.

2. From an engineering perspective, many processing steps are needed to convert the low-level virtual world sensor data into abstract environment states, complex events, and situations. Due to the heterogeneous nature of virtual worlds, it is not possible to develop a generic set of data processing steps that would work with all virtual worlds. Moreover, the type of processing steps needed depend on the application domain, as well as the type of reasoning expected to be done by an IVA. Middleware for deploying intelligent agents in virtual worlds such as Pogamut [25] or CIGA [51] currently do not have the flexibility to change the set of data processing mechanisms to accommodate these many differences (a detailed comparison is presented in Section 7).

To address the first facet, we require a way of describing virtual environment dynamism in an abstract form suitable for the deliberation process of an agent, which should be independent of any virtual-world-specific information. Our solution to this problem is to convert this low-level virtual-world-specific sensor data into semantic descriptions of the virtual environment using a dynamic virtual environment ontology. This ontology has the flexibility to incorporate the information specific to a given virtual world, but is general enough to be extended in a domain-specific manner, according to the application domain of the virtual simulation.

For the second facet, we require a framework that can provide flexibility in adding and removing different data processing components according to the selected application domain, virtual world, and the information requirements of the agent. Our solution to this is a middleware framework for deploying intelligent agents in virtual worlds, which supports the integration of loosely coupled data processing components in order to carry out the information conversion needed between agent systems and virtual worlds. The interfaces between these components are based on our dynamic virtual environment ontology, which makes them easily configurable to use new components suited to the needs of the virtual world, application domain, and the type of reasoning expected by the agent.

The next section provides an informal description of the different elements that contribute to dynamism in a virtual environment, and sets the scene to describe the dynamic virtual environment ontology and the data processing framework. Section 3 discusses previous work related to formal models and ontologies that describe virtual environment dynamism, as well as previous work related to inferring abstract environment information from low-level virtual world sensor data. Section 4 describes our dynamic virtual environment ontology. Section 5 describes the loosely coupled virtual world data processing framework that makes use of our ontology to interface its different components. Section 6 describes how the tight coupling of the data processing components in our previous research [61] has been relaxed thanks to this new framework [43]. Section 7 compares this new framework with some of the existing middleware and frameworks for deploying agents in virtual worlds. Section 8 provides an evaluation of the performance of the framework, and finally Section 9 concludes the paper.

2 Virtual Environment Dynamism—An Informal Description

2.1 Using States and Events to Characterise Dynamism

An environment that changes in ways beyond an agent's control can be identified as a dynamic environment [63]. Helleboogh et al. [31] define the term dynamism as the "evolution of the simulated environment over time". Dynamism in a virtual environment can be understood in several ways. One approach is to describe it using activities, behaviours, and scenarios of participants in the environment [69, 31]. Another approach is to make use of environment states and events to describe the interactions of different environment constituents. In this paper, environment states and events are used to describe the dynamism observed in a virtual environment.

The idea of defining an agent's dynamic environment using events and states is not new. Parunak [58] asserted that most agent research to date *at that time* was founded on state- and/or event-based approaches to the modelling of agent-environment interaction. In the artefact-based environment model presented by Ricci et al. [62], an agent's percepts contain both observable properties of artefacts, and the events (or signals) generated by the artefacts. Bromuri and Stathis [9] capture agent actions and object reactions as events that can be perceived by agents.

From a more general perspective, the state/event approach has been used to model systems with a complex, dynamic, and unpredictable nature. Specifically, in the areas of situation awareness [20] and data fusion [15], system dynamism has been presented in the context of states, events, and situations. In the presence of human participants, virtual environments are also of a complex, dynamic, and unpredictable nature, which makes a state/event approach suitable for describing the dynamism in a virtual environment as well.

While there could be virtual worlds that do not explicitly generate events for an external user, many popular virtual worlds such as Second Life[1] and an extended version of Unreal Tournament [1] have an explicit event representation that can be observed by an external agent. These virtual worlds have a set of low-level events defined within the game or physics engine, but it is also possible for the programmer to script new events. Moreover, many frameworks that connect agents with virtual worlds have assumed that virtual world state changes are based on events [17,24]. Therefore sensor data generated by a virtual world can be assumed to be mainly comprised of event notifications. Even if a virtual world only generates state updates for its clients, it is always possible to derive event information based on this state information, for example by using complex event processing mechanisms [61,59]. Hence, we believe it is justifiable to regard a state/event-based representation of dynamism as a reasonable assumption in a virtual environment.

2.2 Environment Constituents

To begin with, we have to identify what constitutes a virtual environment. Helleboogh et al. [31] consider a simulated environment as comprising a set of entities

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

and environment properties. This observation is valid for virtual environments too. The entities that constitute a virtual environment are avatars and objects. In some virtual environments, there could be properties that are applicable to the virtual environment as a whole, such as sunshine, temperature, and humidity. However, such environment properties are not commonly available in many of the present day virtual worlds and so we do not consider them further. Some virtual environments also provide chat channels that can be used for communication by entities.

Low-level sensor data interfaces published by many present day virtual worlds describe the state of the virtual environment using the state of objects and avatars in that environment. This state is described in the form of *entity properties*. However, it is important to observe that an IVA deployed in a virtual environment using an externally connected agent system may not receive information about *all* the objects and avatars in a virtual environment. Some virtual worlds are designed to send information only about entities that are in the perceptual range of the agent's avatar.

Different virtual worlds define different sets of properties for objects and avatars. Properties of an entity may include position (or boundary for entities such as buildings), velocity, acceleration, mass, texture, etc. Of these, position, velocity, and acceleration are the properties that typically change during the course of a simulation. Thus, such changes have a direct impact on the agent perception of dynamism in a virtual environment.

Most virtual world simulations contain object and avatar types specific to that simulation. For example, a football simulation may contain avatars of type *player*, *referee*, and *coach*. A medical simulation may contain objects of type *thermometer*, *ECG machine*, and *blood pressure monitor*. Objects and avatars may have different sets of properties based on their entity types. For example, an avatar may have a set of animations that it is currently performing, and an object may have an owner. A gun object in a gaming simulation may have a property recording the amount of ammunition left, and different types of player avatars may have different properties. Of course, attributes of entities can vary from virtual world to virtual world as well. A dynamic virtual environment ontology should be able to accommodate the differences introduced by different virtual worlds and different application domains, when describing entity properties.

2.3 Entity Relationships

The states of environment entities, at a given instant in time, can be viewed as capturing the state of the virtual environment at that moment. However, this state data typically describe the agent's environment at a relatively low level of detail that is unsuited to the reasoning processes of a BDI agent. Therefore, we make use of *entity relationships* to abstract this low-level state information, inspired by human cognitive skills that permit us to pick out the different relationships between entities in our surroundings. For example, a book is said to be *on* the table, and a handle is *part of* a door.

The importance of relationships in an agent environment was identified in early work on defining agent environments. For example, Odell et al. [50] note that the

ability to define an agent's location *relative* to another environment location is an important factor in environments, especially for agent communication.

Relationships defined for a virtual simulation could have different forms, including spatial relationships, structural relationships, and functional relationships. For example, spatial relationships might include *in*, *on* and *near*. However, the set of relationships valid in a particular virtual world simulation depends on the associated domain. For example, in a football simulation, entity relationships that can be identified include players or the ball being at specific locations on the field, a player having possession of the ball, and players who are near each other. An ontology to describe dynamic virtual environments should contain both the set of generic relationships applicable to any virtual environment, and the set of relationships specific to a given application domain.

2.4 Communication Interactions

In the presence of human participants, interactions in virtual environments typically involve communication. In other words, what a participant says and what a participant does (his or her 'physical actions') are interleaved and are causally related. For example, in a football simulation, a player requesting another player to pass the ball to him (e.g. 'pass me the ball, Bob') followed by the second player making the actual pass are causally related. Thus an IVA needs to be able to be aware of both these types of interactions. Therefore we concur with Dignum et al. [17] who claim that decisions made by IVAs depend not only on the information they receive through perception, but also on the information they receive through communication.

However, an agent may not be able to make use of the chat messages exchanged in virtual world chat channels as they are. Rather, the agent needs to be able to infer the abstract meaning conveyed by each chat message. We term this abstract meaning of a chat message the *institutional act*. For example, institutional acts defined in a medical simulation may include *take history*, *prescribe medication*, *inform about tests*, and *inform about the illness* [59]. Such institutional acts are domain-specific and their occurrence may depend on the physical and social context of the interaction (such as the roles of the communicating parties). Thus their recognition involves explicit inference based on the content of the chat messages, the physical and social context, and additional domain knowledge[2].

In this work, the concept of dialogue acts is used in abstracting this low-level message information into institutional acts. Dialogue act recognition is an important sub-area of natural language processing, and is an important step in understanding spontaneous dialogue [67]. According to Stolcke et al. [67], a dialogue act represents the meaning of an utterance at the illocutionary force level, and is (approximately) equivalent to speech acts introduced by Searle [64]. Some commonly used dialogue acts are *greet*, *thank*, *goodbye*, *accept*, *state* and *question*. These dialogue acts can be used in association with information about the sender and the recipient of a chat message to identify the institutional act applicable to that chat message. For example, consider the phrase "Do you have a cough?" that

[2] It is beyond the scope of this paper to propose a specific mechanism for modelling institutions, but an overview of various approaches is presented in a recent volume [55, Part IV]

may be exchanged in a chat channel in a medical scenario. This message can be assigned a dialogue act of type *question*. If the sender of the message is a doctor and the receiver is a patient, this message can be said to be an instance of the *take history* institutional act.

To summarise, in order to abstract chat messages in the form of institutional acts, two pre-processing steps are needed to identify (i) the dialogue act applicable to the given chat message, and (ii) its intended recipient (addressee).

2.5 Events

So far we have discussed how to abstract the *state* of the environment, consisting of entity states and message information[3] at a given time instant. In contrast, *events* are associated with changes in the state of the environment. Indeed, in this state-event approach, events are regarded as the cause of dynamism in an agent's virtual environment.

In a virtual simulation, two types of events can be identified: (i) instantaneous primitive events, and (ii) complex events. Primitive events include low-level environment events such as a change of the value of an entity attribute (e.g. a change of an entity's velocity), or the change of a relationship that exists between entities (e.g. an avatar coming *near* an object). Primitive events occur at points in time. In contrast, a complex event is a composition of other primitive or complex events, and will typically have an associated time span. The set of events, complex events in particular, applicable to a simulation depends on the problem domain. For example, for a football simulation, the set of complex events includes the scoring of goals, successful passes of the ball, successful tackles and free kicks.

Some virtual worlds such as Second Life notify their clients about entity property changes in the form of events. In the absence of such primitive events from a virtual world, it is straightforward to infer such primitive events, e.g. by comparing the value of an entity property at two consecutive time instants. However, identification of complex events may not be that straightforward—many complex events depend on complicated temporal relationships between other events and associated environment state information, and require the use of an event recognition mechanism.

2.6 Situations

Earlier, we made use of the concept of entity relationships to provide a more abstract view of the environment state, when compared with the environment state information that consists only of entity state information. However, the abstraction provided by entity relationships may not be enough for an agent in complex virtual simulations. For example, consider the abstract state information required by an agent engaged in a football game. An agent might be interested in knowing whether

[3] Note that it is also possible to consider message sending to be an event. However, as explained later in this section, both events and messages applicable at a given time instant are included in a *snapshot*, and can be used for complex event recognition and situation recognition. Therefore in our approach, it is not significant whether or not a message exchange is treated as an event.

the opposition is leading, or whether an opposition player is attempting to score a goal. In contrast to entity relationship information, this type of information is agent-specific. For example, for two agents in the two teams of a football game, only the agent in the trailing team should perceive the state information that 'the opposition is leading'.

We use the term *situation* to refer to such state information as perceived by individual agents. However, it is not possible to infer this high-level state information using entity relationships alone. Nevertheless, it is possible to make use of environment events to identify the start and end of a situation faced by an agent. It should also be emphasised that situations and events possess a fundamental difference with respect to how they are perceived and how they are acted upon by an agent. Situations are considered to be persistent until terminated by some condition, while events are perceived as transient.

Similarly to events, there can be primitive and complex situations. Complex situations are identified based on the relations between other situations[4]. The constituent situations of a complex situation can be considered as invariants—a complex situation remains active as long as all its constituent situations remain active[5]. The situations applicable to a given simulation depend on the associated domain. For example, in a football simulation, an agent might be facing situations such as *opposition is leading, opposition has possession of the ball*, and *potential goal score by the opposition.*

2.7 Summary

We now review the material covered in this section and summarise the main points. We identified the lowest level of environment state information made available to an agent as consisting of object and avatar information. The state of each object and avatar is described in terms of their properties. The use of entity relationships was introduced to present this environment state at a higher level of abstraction. We have also noted the contribution of communication interactions (chat messages exchanged in the chat channels) to the dynamism in a virtual environment, and the concept of institutional acts was used to describe these communication interactions abstractly. Events were identified as the causes of change in the environment state. The use of situations was then introduced to provide a yet higher level abstraction of the environment than the use of entity relationships, and the role played by events in identifying these situations was also observed.

We use the concept of an environment *snapshot* that encapsulates all these different types of information that hold at a given time instant. In other words, a snapshot contains the state of the entities observed by the agent at the given time instant, the set of messages exchanged in the chat channels at the time instant, any entity relationships abstracted from the entity states, events that occurred, and the situations that hold at that time instant. In Section 4.2, these different

[4] In the situation categorisation put forward by Buford et al. [10], these complex situations are referred to as compound situations.

[5] Note that it is always possible to specify more complex logic here, e.g. arbitrary Boolean expressions could be used to define when complex situations hold based on the presence of other simpler situations.

types of environment information will be put together in a dynamic virtual environment ontology, thus providing a uniform way to describe the dynamism in virtual environments.

3 Related Work

As described in Section 1, the aim of this paper is to (i) provide a semantic means to describe virtual environment dynamism to be used by the deliberation process of an agent, and (ii) to provide flexible sensor data processing, according to the different needs and characteristics of the virtual world, application domain, and the requirements of the agent. In line with these two aims, this section discusses and assesses previous work in each of these research areas.

3.1 Formal Descriptions and Ontologies of Dynamic Virtual Environments

The concept of *environment* has been researched in depth and established with respect to multi-agent systems. However, many of the previous environment specifications and formalisms are focused on designing or implementing environments to facilitate agents, and not on describing an existing environment into which an agent might be deployed. In other words, they focus on specifying the internal structures of an environment, rather than specifying how the environment might be comprehended by an agent deployed in it. For example, Odell et al. [50] present seven characteristics of an agent environment and define it as a first class abstraction. Weyns et al. [80] identify six responsibilities of an environment, one of which is that the environment should be observable. However, the authors do not describe *what* should be observable. Environment models such as those proposed by Ricci et al. [62], and Weyns et al. [80] do present the concept of environment as an abstraction of the underlying simulation environment. However, the abstraction is limited to environment constituents and services offered by the environment to agents. It provides the agent with only low-level information about the environment rather than the kind of high-level and domain-specific situational analysis that we consider essential to bridge the information representation gap.

The environment model discussed by Behrens et al. [4] describes the environment as consisting of entities that are controlled by agents, and those that are active, but are not controlled by agents. This model does not go beyond an abstract description, and does not contain any information related to dynamism in the environment.

The ELMS environment description language [52] is an example of one approach to providing a mechanism for defining observable properties of objects and agents in an environment. However, it does not provide any formal characterisation of the dynamism in the environment, other than the ability to define an agent action as a sequence of changes in properties of objects and agents. Moreover, it makes the assumption that the environment is grid-based, which is in general too simplistic for effective application in complex virtual environments.

Bromuri and Stathis [9] present a logic-based framework that describes an agent's environment as a composite structure that evolves over time. The environment consists of agents and objects. Interactions between these entities are

captured as events. The authors also describe how these events are perceived by agents, and how they affect objects and processes in the environment. The computable specification for entity interactions is implemented using Object-based Event Calculus (OEC). However, the environment dynamism captured by the specification is limited to entity interactions (actions of agents and reactions of objects). Similar to the ELMS language, this logic-based framework also assumes that the environment is a grid-based system. According to Bromuri and Stathis [9], in their model, the need to translate perceptions from the environment to the agent minds does not arise, because the environment and the agent mind share the same representation language. This assumption is not valid in general, because, as we discussed earlier, the environment representations provided by those virtual worlds are typically at a much lower level of abstraction than the kinds of perceptions appropriate for an (intelligent) agent. Nevertheless, it would be possible for one to synthesise ELMS and OEC style representations from low-level virtual world sensor data, by employing a mechanism similar to the one described in this paper.

The dynamic environment formalism presented by Helleboogh et al. [31] contains many useful constructs to define the dynamism observed in simulated agent environments in a declarative manner. These are extensible in a domain-specific manner. In this formalism, dynamism in a simulated environment is captured by the activities of entities and properties in the environment. A scenario describes the evolution of the environment, and is expressed in terms of activities. An activity refers to an evolution of a single constituent (an entity or an environment property) in the environment, meaning that a scenario can only refer to a disjoint set of activities of individual environment constituents. Therefore this formalism is not expressive enough to describe the evolution of the environment with respect to relationships that are formed among environment constituents. Moreover, it is not clear how domain-specific events such as goal scoring in a football simulation can be described using the concept of scenarios as set out above.

Gemrot et al. [25] approach the question from the virtual environment perspective and provide a set of responsibilities and a formal description of game engines for the purpose of connecting agent systems with game engines. This formal description focuses mainly on game engine internals and agent internals. Although a categorisation of game engine facts sent to an agent is given, there is no formal model for describing what these facts are—whether they refer to entity properties, event information, etc. Moreover, little emphasis is placed on describing the dynamism observed by an agent deployed in a virtual world.

Work on the use of ontologies to describe virtual environments semantically can be found, especially in the case of the design process for virtual environments [35, 37, 30]. This is due to the fact that ontologies facilitate re-use of entities across different application domains. If such an ontology is already available, this makes it much easier to define an ontology to describe dynamism in the virtual environment.

However, most present day virtual worlds do not contain any semantic information. Otto [56] describes some of the many difficulties in creating semantic information for an existing virtual environment. The author also describes semantics for multi-user virtual environments, but mainly focuses on detailed descriptions of entity relationships in the environment. Dynamism in the environment is addressed through the use of events, however no concise description is provided about the relationship between primitive and complex events, nor is the concept of situations considered.

Grimaldo et al. [28] also describe the implementation of a semantic layer on top of an existing virtual environment world model for use by IVAs. Their focus is on using ontological descriptions in facilitating agent-object interactions, and as a consequence, the focus on semantic description of the dynamism observed in the environment is limited. The authors also make use of ontologies to describe social relationships among participants in a virtual community. In respect of this last, Kao et al. [38] present an ontology to describe social relationships in a virtual environment for use by IVAs.

Chang et al. [12] identify the importance of having a formal model for virtual environments. They present a three-layer cognitive architecture as a unified model that includes both agents and the environment. However, this model only describes how an agent perceives its environment with respect to what can be done with the objects in the environment. Dynamism in the environment is not addressed.

Aranda et al. [3] present the MMOG Ontology, part of a software layer in which the rules and norms of a Massively Multiplayer Online Game can be defined and enacted independently of an underlying agent-based "intelligent virtual environment". The ontology defines a number of classes relevant to game playing, and this can be extended with ontologies specific to particular games. However, the focus is on defining the properties of entity types, and relationships and situations are not included.

In summary, we conclude that the environment models in current use, as surveyed above, in the field of MAS are not rich enough to describe the dynamism of complex virtual environments as characterised in Section 2. Furthermore, the extant virtual environment ontologies do not adequately account for this phenomenon, from which we conclude there is a need for a specific dynamic virtual environment ontology.

3.2 Generating Abstract Virtual Environment Information for Agents

In Section 2, we identified the use of entity relationships, institutional acts, events, and situations to describe dynamism in the virtual environment in an abstract manner. This section describes the previous work related to each of these elements.

Gemrot et al. [25] present a rule-based approach to convert low-level position information of entities received from a virtual world into a suitable abstraction for use by BDI agents. This infers high-level information such as agent a is chasing agent b. Similarly, Vosinakis and Panayiotopoulos [74] present a set of rules that infer entity relationships from the low-level entity position information received from the virtual world.

Inferring abstract information from textual messages is essentially a natural language processing (NLP) problem. Implementing NLP capabilities in virtual humans has for some time been considered an important aspect of improving their believability [45]. Consequently, there exists much research on applying NLP techniques to interpret chat communications in virtual environments. Virtual humans with the ability to communicate with humans are commonly referred to as Embodied Conversational Agents (ECAs). The main purpose of many of these ECAs is to answer the queries of a human user. Their NLP capabilities have typically been implemented as question-answering systems that involve both natural language

understanding and natural language generation. Conversational virtual agent technologies make use of techniques such as chat-bot based pattern-matching [34, 40], text classification [2, 70], and hybrid approaches [29].

However, there is little research in the IVA field that combines textual communication with the other observations of the environment to be used in complex event or situation recognition. The importance of the use of chat utterances to interpret the significance of events has been identified by Small et al. [65]. However, their approach to identifying complex events in virtual environment is currently achieved by manual annotators. Orkin et al. [54] describe an algorithm that categorises physical actions and textual messages into high-level tasks, but the algorithm does not consider complex temporal relationships between these two types of actions.

Warden and Visser [76] employ rule-based pattern recognition on spatio-temporal sensor data extracted from a 3D simulation to infer complex events. However, this reasoning is only based on discrete state updates. This means that information about events taking place in between these discrete updates (e.g. an event corresponding to an avatar changing its velocity) can go undetected. This is the classic problem of missing information if the environment changes multiple times between two subsequent state updates [62]. The authors claim this framework can be used by an agent system, although no examples are cited. Moreover, it should be noted that this experiment was conducted only using a 3D agent simulation, with no human participants. In that sense, it is also arguable whether this can be called a *virtual world*.

In the work of Fielding et al. [23], individual agents are capable of remembering environment objects they have perceived in the past. They compare this information with the currently sensed environment state and interpret abstract events in the environment. However, this inference is done in a rather domain-specific manner in the context of one specific game genre in Unreal Tournament.

Although not associated with IVA research, the work of Ziafati et al. [83] is of interest, because it discusses the lack of event processing support in BDI agent systems, and presents a technique for generating complex events from low-level sensor data, to be used by a robot system that makes use of BDI agent technology. Similar to the use of the Esper event processing system in our previous work [61] (see Section 6), this work makes use of an event processing component implemented in the ETALIS language.

Bogdanovych et al. [6] present three levels of interpretation of the sensor data received from the virtual world. The first contains avatar positions (e.g. distance and orientation with respect to some static objects) and position changes (e.g. moving forward, or turning right). At the second level, they identify simple avatar interactions such as avatars approaching each other. The third level is based on the information generated by the 'virtual institution' they have employed. This includes messages such as placing bids, entering rooms, etc. that are generated by the virtual institution. Thus, these high-level events only arise if such a virtual institution layer is present.

With respect to situations, Zhang et al. [82] present a pattern-driven approach to recognising situations in a virtual battlefield. These patterns (or structures) reflect the organisational and spatial relationships of the entities in the battlefield, and each such spatial and organisational structure is represented as a k-d data structure (a special form of tree structure). In order to recognise situations within

an intelligent pilot bot, Ehlert et al. [18] provide two mechanisms: heuristic rules based on a state-transition diagram and a probabilistic approach. In the first approach, IF-THEN rules are defined for a number of different situations during the flight, and each situation can be recognised based on a number of parameters. To reduce the number of rules that should be checked in each decision cycle, these rules are grouped as a state-transition diagram. In order to remove the deterministic nature of this IF-THEN rules approach, probabilities are used to determine the start and end of situations. Cavazza and Palmer [11] make use of FSTNs (Finite State Transition Networks) to identify avatar actions, based on the events received from an event monitor situated inside the virtual world. However, this information is not provided to any IVA, and action definitions are provided only for one avatar. Feng et al. [21] present a situation model that is jointly accessed by agents in a virtual environment. These situations are based on entities and events in the environment; however, how these entity states and events are identified, or how the situation model is updated, are not described.

In a more general sense with respect to situation recognition in intelligent agents, So and Sonenberg [66] make use of the event calculus to identify situations in the BDI model based on the identified events. They identify a situation as "a narrative of observable events, which lead to a particular state of the environment when they all happened". Thangarajah et al. [71] identify active situations faced by BDI agents by detecting events that initiate and terminate them. Both these works assume agents can handle high data volumes, and have not focused on situation recognition on real-time sensor data received by an agent. Buford et al. [10] have proposed a theoretical framework that makes use of event correlation for situation recognition by BDI agents, however, no implementation is available for this framework.

From the literature that we have surveyed above, we see there is existing work that addresses inference in various ways and at varying levels of fidelity for each of the environment elements discussed in Section 2. However, none shows how *all* of these elements can be identified from the low-level virtual world sensor data, in order to provide a coherent and comprehensive view of the environment to the agent. Moreover, several appear to be bespoke solutions and it is not clear how readily these approaches can be generalised across different virtual worlds and different application domains.

4 Semantic Description of Dynamic Virtual Environments

4.1 Virtual Environments

According to Weyns et al. [80], an environment provides the agents with an abstraction level that "shields low-level details of the deployment context—as well as other resources in the system". In the context of IVAs, this deployment context is the execution of the underlying game (or simulation) engine. The game engine by itself, or with the help of other disparate components, carries out tasks such as object physics calculations, script execution, and avatar animation execution. The actions of the underlying game engine are not visible to an agent connected to the virtual world externally (i.e. agent logic is not embedded in game engine logic).

The interface provided to virtual world clients determines what information about the game engine state is disseminated to them. The presentation of this information is typically via event notifications and/or periodic state updates. Clients are then responsible for interpreting this information to create their representation of the *perceived* state of the virtual environment. Thus the clients (including agents) perceive the virtual environment at a different level of abstraction from the underlying process and state of the game engine.

In consequence of the above, this paper makes an explicit distinction between the terms 'virtual world' and 'virtual environment'. Although these two terms are often used interchangeably in the literature [19,24], 'virtual environment' is used here to refer to the abstract state space observed by an external agent. The virtual environment observed by an externally deployed agent is only an abstracted representation of the system constructed from the received event notifications and/or the state updates under some communication protocol. Moreover, virtual worlds such as Second Life constitute different inter-connected regions that do not have direct access from each other. In such situations, the environment observed by an agent is only a *part* of the virtual world that the agent currently has direct access to[6].

We do however use the terms 'virtual environment' and 'virtual simulation' synonymously. A domain-specific virtual simulation represents a subset of a virtual environment; thus any recorded information that falls outside the simulation boundaries may be ignored. For example, the SecondFootball[7] simulation in Second Life covers only about a quarter of a Second Life region, and what is happening in the rest of this region is not related to the SecondFootball simulation.

4.2 Dynamic Virtual Environment Ontology

In Section 2, we identified the elements that are essential when describing dynamic virtual environments abstractly using a state/event-based approach. These include entity property information, chat messages, entity relationships, events, and situations. In this section, we describe in detail our dynamic virtual environment ontology that encompasses these different elements and their inter-relationships, in order to meet our requirements for the virtual world scenarios in which we are operating.

As mentioned in Section 2, the sensor data interfaces provided by different virtual worlds can vary widely. Different virtual worlds may expose different sets of properties for entities, and the sensor data may well be at different granularities. Moreover, abstract information of a dynamic virtual environment depends on the domain of the simulated application. Despite these differences, when improving awareness of an agent deployed in a virtual environment, a uniform way of describing the environment dynamism to the agent is required. Moreover, such a uniform method is necessary to facilitate the communication between the different data processing components employed in abstracting virtual environment information.

[6] For example, when connecting an agent to Second Life, the region to which the agent should be logged in should be specified. The agent cannot directly access (e.g. 'walk to') another region. The only means of doing this in Second Life is by 'teleporting'.

[7] `http://www.secondfootball.com/`

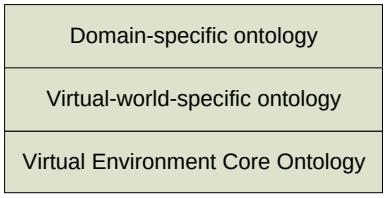

Fig. 1 Virtual environment ontology stack

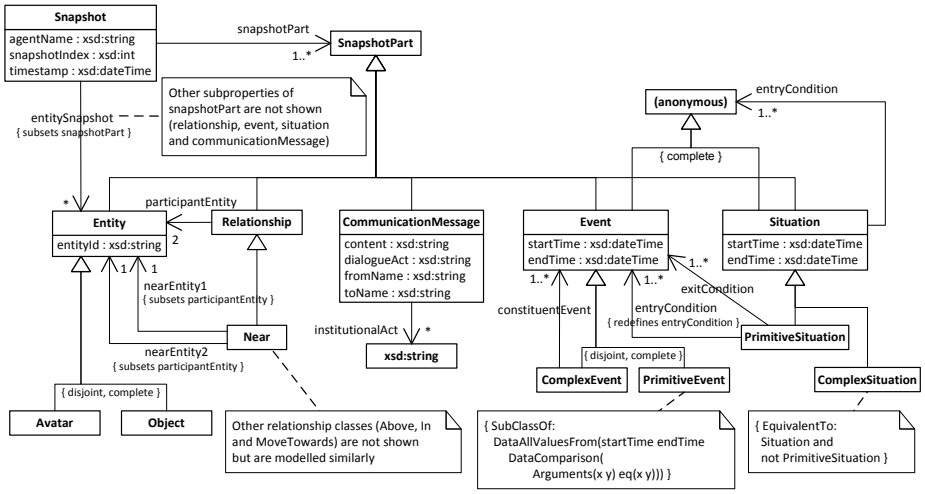

Fig. 2 The virtual environment (VE) core ontology

In order to describe the dynamism in a virtual environment, despite the heterogeneous nature of virtual world sensor data, we have designed a core virtual environment ontology that is independent of any virtual-world-specific and/or application-specific information. As shown in Fig. 1, this dynamic virtual environment core ontology can be extended with a virtual-world-specific ontology and a domain-specific ontology on top of that. The virtual-world-specific ontology captures the information specific to a particular virtual world, while the domain-specific ontology contains information related to the specific application domain. This approach is different from the work of Grimaldo et al. [28] described in Section 3.1, where the virtual environment ontology has only two layers—a core virtual environment ontology, and a domain-specific ontology. Our approach is more robust because it is not affected by the heterogeneity of the sensor data interfaces published by different virtual worlds. We make use of the OWL 2 Web Ontology Language as a foundation for the dynamic virtual environment ontology.

Fig. 2 shows our virtual environment (VE) core ontology. Fig. 3 and Fig. 4 show the ontological information related to Second Life (SL) and the application domain

of SecondFootball (SF), respectively. For concise presentation, these figures use the Object Management Group's UML profile for OWL[8][49].

In our VE ontology (Fig. 2), environment dynamism is encapsulated in a `Snapshot`, as described in Section 2.7. A snapshot describes the state of the environment at a discrete time instant, and records this information at different abstraction levels. A snapshot is a collection of `SnapshotPart`s. A snapshot also has three properties that identify the name of the agent receiving it (`agentName`), the `snapshotIndex` in the sequence of snapshots, and the `timestamp`. In our informal description of virtual environment dynamism, we noted that the state of individual entities in the virtual environment collectively represent the state of an environment. In that description, relationships were introduced as an abstracted view of this low-level entity information. In addition, any chat messages exchanged in chat channels should be captured. Dynamism in the environment is characterised through events and situations. In order to capture these different types of information in a snapshot, a `SnapshotPart` is defined to be one of an `Entity`, a `Relationship`, a `CommunicationMessage`[9], an `Event`, or a `Situation`[10]. All these belong to our core virtual environment ontology.

The `Entity` class encapsulates the state of an entity in a virtual environment. An `Entity` may be either an `Avatar` or an `Object`. Every `Entity` has an `entityId`. As described in the previous section, the set of properties of an `Avatar` or an `Object` exposed to an external agent depends on the selected virtual world. In order to have the flexibility to support different sets of entity properties, they are defined in the virtual-world-specific ontology. For example, the classes `slAvatar` and `slObject` represent the entities in a Second Life environment (see Fig. 3). Currently we have defined position, velocity, and acceleration as properties common to all Second Life entities. Since these are complex property types having a 3D vector representing their value, a `SpatialVector` class is defined in the SL ontology to record these 3D coordinate values.

The `Relationship` class in the VE ontology is used to describe the various relationships that exist among the objects and avatars included in a `Snapshot`. It is possible to define relationships that are common to any virtual environment, as well as those specific to a given application domain. For example, the `Near` relationship shown in Fig. 2 is defined as a generic relationship, while the `HasPossession` relationship in the SF ontology (Fig. 4) is specific to the SecondFootball domain, and describes the relationship between the ball and the player who has possession of it.

Section 2 identified the contribution of chat messages to the dynamism of an environment. Therefore the VE ontology (Fig. 2) has a `CommunicationMessage`

[8] Following this UML profile, UML "subsets" constraints indicate subproperty relationships, "redefines" constraints indicate subproperties with additional OWL2 property restrictions, "disjoint" constraints indicate disjoint subclasses, and "complete" constraints indicate that a superclass is the union of a set of subclasses. Other OWL2 constraints are represented using the OWL2 Manchester syntax within UML constraint boxes. In Figure 2, the constraint on class `PrimitiveEvent` uses the OWL 2 Linear Equations data range extension [57]. For clarity, the diagrams omit inverse properties and some disjoint class constraints. We also omit the "owlClass", "objectProperty" and "datatypeProperty" stereotypes defined by the profile.

[9] Note that we treat a CommunicationMessage as an entity different from an event (see Section 2.5).

[10] In the figure, only `entitySnapshot` is shown as a subproperty of `snapshotPart`. Other subproperties are omitted for clarity.

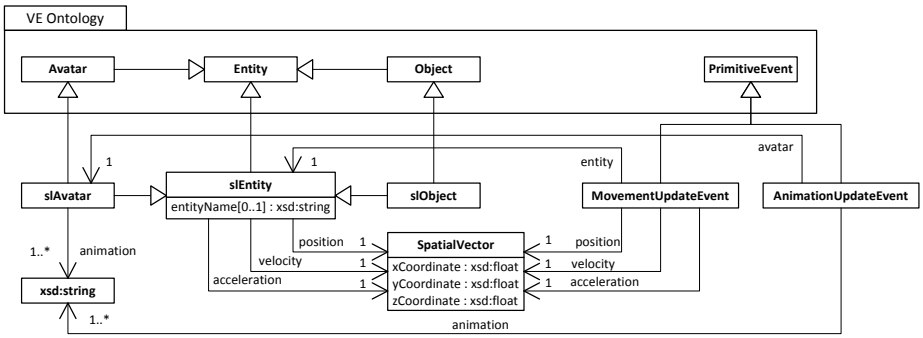

Fig. 3 The Second Life (SL) dynamic environment ontology

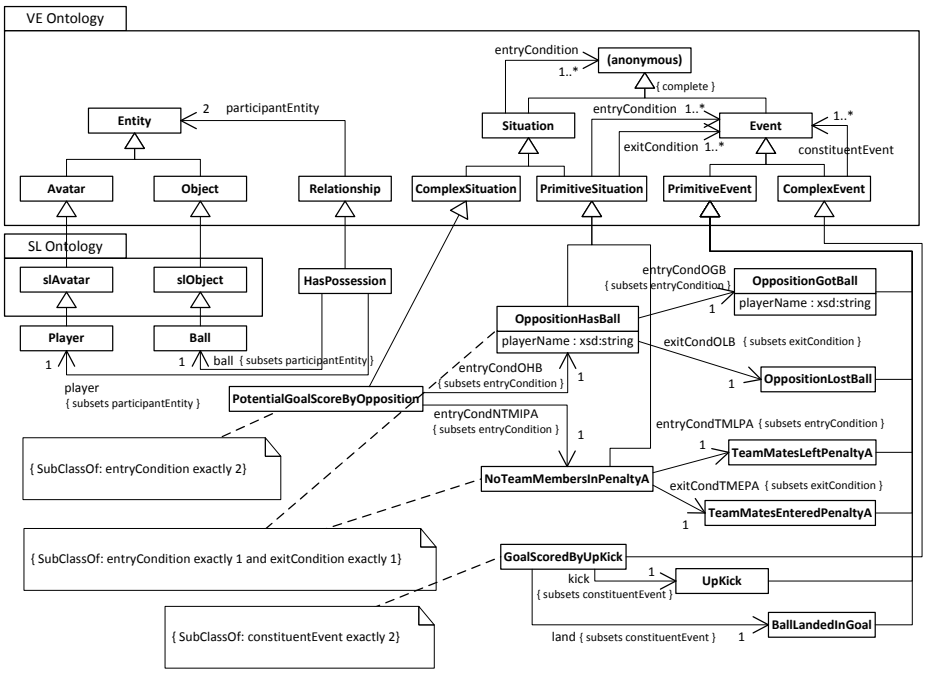

Fig. 4 The SecondFootball (SF) ontology

class to represent such chat messages sent in the virtual environment. Among its properties are `fromName` (the sender) and the message `content`, which refer to the sender of the chat message and the message content, respectively. This is the only information available about a chat message received from a virtual world. Section 2 identified the concept of institutional acts to describe these chat messages abstractly, and the `institutionalAct` property associates zero or more of these with a `CommunicationMessage`. We also identified the use of dialogue acts and the message receiver information in inferring the institutional act of a chat message, and these are included in the ontology as the `dialogueAct` property and the `fromName` properties of the `CommunicationMessage` class, respectively.

Section 2 identified two types of events in a virtual environment: primitive events and complex events. The `Event` class in the VE ontology (Fig. 2) is the base class for both primitive and complex events that take place in the virtual environment at the time instant represented by the `Snapshot`. These events could be specific to a given virtual environment or an application domain. For example, in Fig. 3, `MovementUpdateEvent` and `AnimationUpdateEvent` are specific to Second Life. These two events, respectively, correspond to the change of velocity of a Second Life entity, and a change of the currently played animations of a Second Life avatar. In addition to the generic event properties `startTime` and `endTime` defined in the virtual environment ontology, each event can have a set of properties specific to it. For example, the `AnimationUpdateEvent` class has properties specifying the avatar name and a string containing the new list of animations being "played" by the avatar (see Fig. 3). On top of these, some complex events have a set of constituent events, being the set of primitive or complex events associated with that complex event. As shown in Fig. 4, the `UpKick` and `BallLandedInGoal` events in a SecondFootball simulation serve as constituent events for the `GoalScoredByUpKick` event.

In the VE ontology, the class `Situation` represents any situation that holds at the time instant represented by the `Snapshot`. Situations can be either primitive or complex. A primitive situation has entry and exit conditions that are (implicitly) conjunctions of events. A complex situation has only a conjunction of entry conditions, of which at least some are situations—the situation is deemed to end when one or more of the entry condition situations end.

For example, in the SecondFootball ontology (Fig. 4), for the `OppositionHasBall` situation, the `OppositionGotBall` event serves as the entry condition, and the `OppositionLostBall` event serves as the exit condition. In contrast, the `PotentialGoalScoreByOpposition` situation has no exit condition; it has only two entry conditions that refer to the `OppositionHasBall` situation and the `NoTeamMembersInPenaltyA` situation.

In the ontologies presented in this paper, we only model situations specific to the SecondFootball domain. It is also possible to define domain-independent situations at the virtual environment ontology level, such as an entity maintaining a constant velocity for a period of time. However, we believe these are less likely to be useful to an agent's deliberation than domain-specific situations.

5 A Loosely Coupled Data Processing Framework

When interfacing virtual worlds and agent systems, we can make several observations:

- A virtual world may have its own data protocol to communicate with the clients. If the virtual world server-side logic is proprietary, it is not possible to adopt a new communication protocol.
- Similarly, an agent system may have a protocol to communicate with the environment to which it is connected. It may be infeasible, or at least not trivial, to change this communication protocol.
- The type and number of the specialist data processing components needed (to process sensor data, in our context) depends on the selected virtual world, agent system, and the application domain of the virtual simulation.

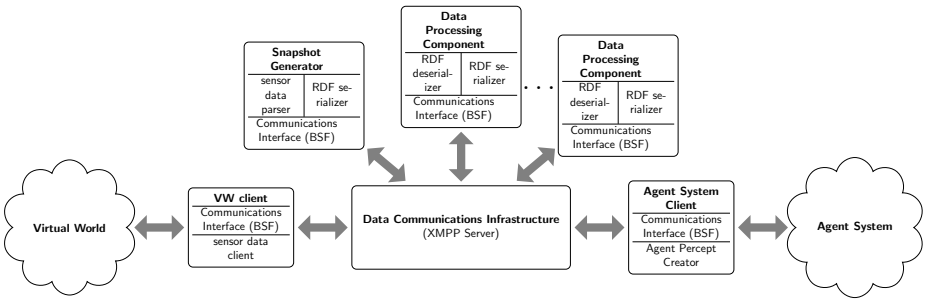

Fig. 5 Overview of the data processing framework

In order to address these different concerns, we propose a loosely coupled data processing framework (Fig. 5), where the communication across different components of the framework is achieved through a common data communication infrastructure, with each data processing component implementing a common interface to this infrastructure. In essence, we rely on the widely accepted service-oriented approach that is now commonly used to implement inter-operable enterprise applications.

In its current instantiation, this data communication infrastructure is realised by an XMPP (Extensible Messaging and Presence Protocol) server[11], while the communication interface of each component is implemented using the Bath Sensor Framework (BSF) [42], in which all components utilise a publish/subscribe façade.

XMPP consists of a set of open technologies that have been used in many different lightweight middleware architectures, in addition to their common use in instant messaging, multi-party chat, and voice (and video) calls. Based on XML, XMPP technologies are designed to be open, standardised, flexible, and extensible. As well as their commercial applications such as Google Talk[12] and Facebook chat, XMPP technologies are also used for research purposes [68,75,77,41,27].

In contrast to the TCP/IP-based approaches used in previous research to interface agent systems with virtual worlds, XMPP adds a substantial degree of flexibility. In particular, it supports one-to-one, one-to-many, many-to-one and many-to-many data transport mechanisms. From our previous experience with the use of TCP/IP for interfacing agent systems and virtual worlds [61], TCP/IP supports 1-1 connections, and tends to be fragile and cumbersome. In TCP/IP, one node has to act as a server, which is an unnecessary burden on a data processing component. Furthermore, the client always has to know the server identity, and explicit fault-tolerance logic should be in place to handle the case when the server fails. In contrast, most of the communication management in XMPP is handled by an XMPP server, and provides a degree of fault tolerance to the communication. This way, such additional logic does not have to be replicated in each participating component. Most importantly, XMPP supports the publish/subscribe mechanism, so that the communicating components do not have to know the identity of each other, and are oblivious to the existence of the others.

[11] We currently use the Openfire XMPP server (`http://www.igniterealtime.org/projects/openfire/`).

[12] In mid 2013 Google Talk switched to Google's own push-notification service

The Bath Sensor Framework (BSF) supports both the publish/subscribe and one-to-one communication patterns enabled by XMPP. Just like XMPP, BSF, to a large extent, is independent of both the operating system and the programming language. Currently it is available for both Java and C#[13]. Libraries to implement the logic in almost all of the commonly used programming languages have been provided by the XMPP community[14]. In the work of Lee et al. [42] the BSF was used to interface a virtual world with an agent system, but latterly Lee et al. [43] demonstrated the use of an institutional governance component. In the current work, we have capitalised on the framework's flexibility and have introduced a series of additional data processing components that are used to convert the low-level virtual world sensor data into a form assimilable by the agent system.

As stated in the list of observations at the beginning of this section, it is not possible to have a generic component to retrieve sensor data from a virtual world, as this depends on the communication protocol employed by the virtual world. Therefore as shown in Fig. 5, a Virtual World (VW) Client that supports this communication protocol is implemented, and it directly communicates with the virtual world server. This VW Client is specific to the selected virtual world, and is the main component in need of replacement when the agent is connected to a different virtual world.

Like the VW Client, the Snapshot Generator is dependent on the type of sensor data provided by the virtual world. Its responsibility is to convert the virtual-world-specific sensor data into an RDF model, expressed in terms of the virtual world ontology described in Section 4.2. After this step, each data processing component in the route communicates using this RDF data. In fact, the Snapshot Generator can be considered as the most prominent component in this framework, as it converts the sensor data from a virtual-world-specific format to a semantic representation.

A notable aspect of this loosely coupled data processing framework is the ability to change the set of data processing components with minimal effect on the rest of the components. Therefore, it is in principle possible to plug in any data processing component according to the needs of the selected virtual world, agent system, and the simulated application domain. In the next section, we show how a set of data processing components is incorporated into the framework in order to convert the low-level sensor data received from Second Life into an abstract format suitable for the deliberation process of a BDI agent implemented using the Jason [7] platform. As described further in the next section, each of these data processing components is responsible for inferring different `SnapshotPart`s described in the dynamic virtual environment ontology.

Note that the responsibilities of the data processing components can also be designed in many different ways. For example, consider the model for active perception for situated agents (Fig. 6) presented by Weyns et al. [81]. In this model, active perception of situated agents (which includes virtual agents) is divided into three functional modules: sensing, interpreting and filtering. One could develop data processing components to represent each of these modules. It is also possible to distribute the functionality of each of these three modules across more than one data processing component, in order to accomplish the different types of pro-

[13] `https://code.google.com/p/bsf/`

[14] `http://xmpp.org/xmpp-software/libraries/`

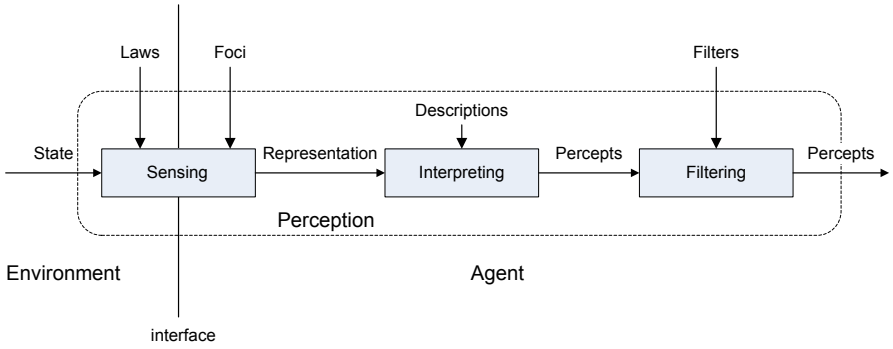

Fig. 6 Model for active perception in situated agents (redrawn from Weyns et al. [81])

cessing that has to be done at each level. For example, the set of data processing components described in Section 6 can all be categorised under the interpreting step of this active perception model. The framework makes no assumption about the type of processing conducted inside a component, as long as the communicated data adheres to the virtual environment ontology.

Similar to the virtual world interface, an agent system might have its own interface to communicate with its environment. Therefore we introduce an Agent System Client at the agent system side, which is responsible for doing the necessary data conversion from the communicated RDF data into a representation accepted by the agent system.

XMPP is flexible and extensible with respect to the type of information that can be communicated across the participating clients. Therefore, communicating RDF data over XMPP is a fairly straightforward process—each component needs an RDF serialiser that converts the data from the component's internal representation to an RDF representation and sends it out through the BSF Communication Interface. Similarly, if a component receives RDF data, it requires an RDF deserialiser to convert the received data into the internal data representation of the component.

6 Example Implementation for Second Life and Jason

In the previous section, we discussed the general architecture of the sensor data processing framework. In this section, we show how this framework can be used to interface Second Life and Jason, together with a set of data processing components that convert the low-level sensor data received from Second Life into the `Snapshot` components discussed in the virtual environment ontology.

In previous work, we described the development of a framework to convert the low-level Second Life sensor data into an abstract virtual environment representation to be used by Jason agents [61]. A limitation of this framework was that it was tightly connected with the data representation specific to Second Life. Moreover, all the data processing components were tightly connected to each other. Because of the latter, it was difficult to add new components that were implemented in programming languages incompatible with the original implementation. In this prior

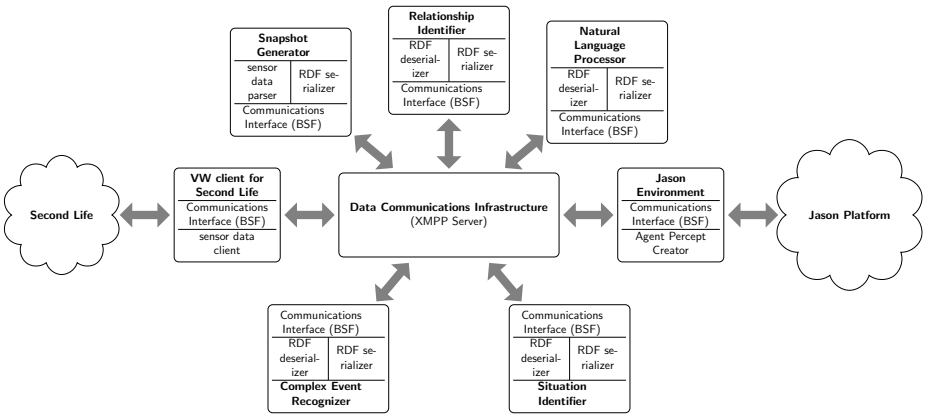

Fig. 7 Example implementation of the framework for Second Life and Jason

work, communication between the data processing framework and Jason was implemented using TCP/IP sockets, which converted the data processing framework into a server and led to a rigid connection between the two systems.

Here, we show how to overcome these limitations. With respect to data representation, we make use of the virtual environment ontology discussed earlier, to provide a virtual-world-neutral ontological data feed that abstracts away from the concrete, syntactic structures used previously. We also utilise the loosely coupled sensor data processing framework to build a less rigid, component-oriented, distributed processing system, in which the previously tightly coupled components communicate through a BSF interface. Fig. 7 shows how the different components in the framework are interfaced with each other. Below, we briefly describe the functionality of each of the data processing components implemented first by Ranathunga et al. [61], and further improved in later work [59]. Note that the Natural Language Processor and the Situation Identifier were not implemented in the former work [61], but are new additions in the latter [59]. Therefore for a full detail of the implementation, we refer the reader to this most recent work [59].

The VW Client for Second Life makes use of the LibOpenMetaverse (LIBOMV)[15] open source library to establish and maintain connections with the Second Life server. With appropriate programming techniques, the LIBOMV library can be used to create a Second Life avatar that looks exactly like and has behavioural abilities similar to avatars controlled by humans. An agent connected to a LIBOMV-based client can take advantage of two main types of functionality provided by the LIBOMV library. The first ability is to perform actions inside Second Life. The second is sensing the surrounding environment and receiving messages that are exchanged in the chat channels (of Second Life) or those that are sent directly to the avatar as instant messages. The LIBOMV Client receives events generated by the Second Life game engine upon the change of entity properties and at the exchange of a message in a public chat channel, as well as the periodic state information generated by an in-world sensor data extraction script that we have developed in previous work [60].

[15] `http://lib.openmetaverse.org/wiki/Main_Page`

The VW Client for Second Life acts as a container for LIBOMV clients, and communicates the low-level Second Life sensor data received by each LIBOMV client through the BSF communication interface. The VW Client creates and maintains LIBOMV clients upon the receipt of a request by a remote agent. A sample of textual data produced by this component on behalf of one such agent is shown in Fig. 8. This piece of sensor data corresponds to a periodic state update generated by our in-world script. It has been received by the LIBOMV client corresponding to agent Ras Ruby, and has been recorded at the time instant 10:11:12. It contains position and velocity information of two avatars, Ras Ruby and Su Monday.

```
Ras Ruby script_periodic_update 10:11:12
f336924b-b880-448f-ad35-55cd5dffb3e4:Ras Ruby:1:
<199.0084, 191.9911, 1001.5500><0.0000, 0.0000, 0.0000>|
f556924b-b880-444f-ad45-00cd5dffb7e4:Su Monday:2:
<100.0084, 191.9911, 1001.5500><3.5200, 0.0000, 1.0055>|
```

Fig. 8 Example output from the VW Client

The Snapshot Generator receives low-level Second Life sensor data in textual form as shown above, parses it and constructs an equivalent RDF representation using the dynamic virtual environment ontology. Note that our dynamic virtual environment ontology can be easily extended to contain information specific to a given virtual world. We make use of this feature to communicate (ontological) data that also contains Second Life specific information. Each RDF data structure generated by the Snapshot Generator represents a snapshot of the virtual environment (as defined by the ontology) derived from the sensor data received by a LIBOMV client at a particular time instant. As explained above, the sensor data received from Second Life contains both event information generated by the Second Life game engine, as well as the periodic state information generated by an in-world sensor data extraction script that we have developed. The Snapshot Generator produces a snapshot upon the receipt of every sensor data item received from Second Life, and therefore does not lose any information sent from Second Life.

After production by the Snapshot Generator, the RDF graph contains entity state information, as well as information related to any event notifications sent by Second Life, and any messages that were exchanged in chat channels. Each data processing component discussed below processes this snapshot information and adds the newly inferred information to the RDF graph. Fig. 9 depicts the RDF representation of a snapshot as a UML object diagram. Here, the `In` relationship between slAvatar 'Ras Ruby' and slObject 'midfieldA1' is information inferred by the relationship identifier, while the remaining information has been produced by the Snapshot Generator based on the sensor data received from Second Life.

The Relationship Identifier makes use of this RDF data, and infers the predefined relationships that exist between the entities included in that snapshot.

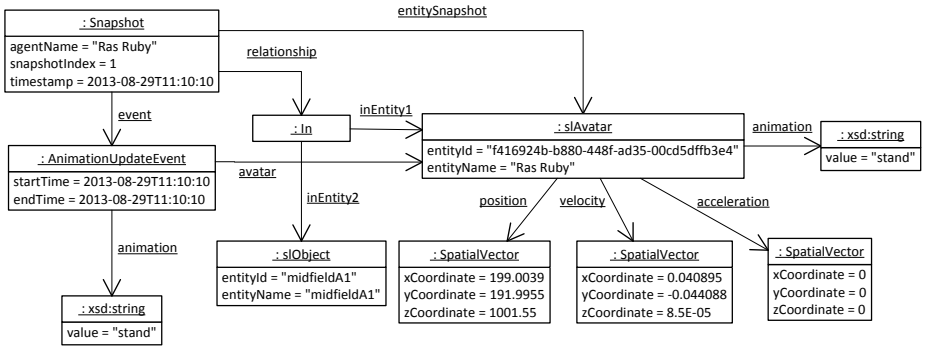

Fig. 9 UML representation of a snapshot

The approach for dynamic relationship identification presented in this paper is similar to that of Vosinakis and Panayiotopoulos [74]. Functions are provided to derive the set of relationships that hold for entities included in each snapshot. All the functions valid for a given simulation (as specified by a configuration file) are executed on the set of entities stored in each snapshot. Functions are executed in such a manner that the relationship predicates identified in earlier functions can be used in functions that are executed afterwards. We do not provide a generic approach to define these relationship functions, and these are left as an implementation detail.

For example, consider the *near* relationship defined between two entities. For each pair of entities e_1, e_2 included in the snapshot, we execute the function *get_near_entities* to check whether the distance d between e_1 and e_2 is less than a pre-defined distance threshold dt.

The Natural Language Processor is the next in the processing line. Its purpose is to deal with any communication messages in a snapshot[16]. As discussed in Section 2.4, a communication message received from a virtual world does not, in general, include any abstract information pertaining to that message. This is true in the case of Second Life chat messages as well. Therefore the responsibility of the Natural Language Processor is to assert the dialogue act and the set of institutional acts applicable to a chat message. In addition, it is required to identify the receiver (addressee) of a particular message. In order to accomplish this, the Natural Language Processor consists of three parts: (i) a dialogue act recogniser based on a statistical classifier, (ii) a rule-based addressee recognition algorithm, and (iii) a rule-based institutional act recogniser. These components are used in a sequential process, where the inferred dialogue act and addressee information is used in inferring the set of institutional acts applicable to a given chat message.

The Complex Event Recogniser receives the snapshot next. As the name implies, the task of the complex event recogniser is to infer any complex events that occurred at the time instant represented by the snapshot. In other words, event detection involves making use of the different elements included in the

[16] Note that the Snapshot Generator creates a new snapshot whenever it receives an event representing a chat message exchanged in the Second Life public chat channel.

incoming snapshot stream and identifying pre-defined temporal patterns corresponding to primitive, as well as complex events.
Event detection in our work has two parts: first, occurrence conditions corresponding to events that should be recognised in a given virtual environment are defined. Then these event occurrence conditions are applied on the incoming snapshot stream to detect corresponding events. Here, defining event conditions is done at design time, while actual event recognition is carried out in real time. For real-time event recognition, we make use of the Esper event stream processing engine[17]. Esper has both event detection capabilities and event stream processing capabilities. Therefore, using Esper on the incoming snapshot stream for event recognition is a straightforward process.

The Situation Identifier is responsible for making use of the event information included in a snapshot and identifying any active situations that are faced by an agent at the time instant represented by the snapshot.
The Situation Identifier makes use of a set of situation templates that describe the entry and exit conditions for each situation, following our virtual environment ontology. A rule-based algorithm is designed to identify active situations in a virtual environment using these templates. Events identified in the previous step are used as the input to the situation identification process. After a primitive situation is identified, a check is carried out to determine if any complex situations having this situation as a triggering condition should be started. Upon identification of the start of a situation, it is added to the list of active situations. If an event is received that is equal to the exit condition of an active situation[18], that situation is removed from the set of active situations. All complex situations having this situation as a constituent entry condition are also terminated.

The Jason Environment class is used as the interface between the Jason agent and the environment in which it is deployed. In contrast to the Agent System Client that was shown as an external component to the agent system in our general architecture (see Fig. 5), in Jason, the Environment class is tightly coupled with the Jason Agent Infrastructure. Therefore, the percepts that the Environment class generates from the received ontological data are communicated to the corresponding Jason agents through direct method invocations.

The VW Client, Snapshot Generator, Relationship Identifier, Natural Language Processor, Complex Event Recogniser, and Situation Identifier are all currently implemented in C#. Therefore it is straightforward to include the C# version of the BSF interface in each of these components. We implemented an RDF serialiser/deserialiser library in C#, which is used by all these components, except the VW Client.

An RDF serialiser/deserialiser library was required because none of these data processing components have the capability to directly deal with RDF data. The data structure of the `Snapshot` internally used in these components is similar to the our core ontology class structure. The RDF deserialiser logic converts the received ontological data into this data structure. Once the data structure is created, it is used for all the internal processing of a component. This snapshot data structure

[17] `http://esper.codehaus.org/`

[18] The current implementation is based on an earlier version of the ontology that specified that a primitive situation should have a single event as an exit condition.

creation process is designed in such a way that it retains the corresponding ontological data (as a graph structure) inside the data structure itself. Therefore it is simple to serialise the `Snapshot` data structure back into RDF, once the component is done with its processing.

This serialisation/deserialisation logic inside each data processing component introduces additional processing time into the data processing pipeline. Therefore future work should explore how to directly use the RDF data in the data processing logic of each of the component. Having said that, this work will be non-trivial, particularly if the data processing logic (e.g. the Esper logic used in this work) makes use of other existing technology.

The Jason Environment class uses the Java implementation of the BSF interface. The Jason Environment class contains the logic to convert the received ontological data into a Java object that describes the snapshot[19]. This object representation is similar to the data structure mentioned above for C#, where objects are created for each communication message, event, entity snapshot, relationship, and situation included in a snapshot. The classes corresponding to these objects contain the logic to generate Jason percepts.

The communication between these components can be summarised as follows. When a Jason agent wants to connect to a Second Life region, it executes the `connect_to_vw` action[20], which is then interpreted by the Jason Environment class and sent to the VW Client component. Upon the receipt of this request, the VW Client spawns a new LIBOMV client and associates it with the corresponding agent information. Any action request issued by the Jason agent is also received by the Jason Environment class, and is communicated to the LIBOMV client, which contains the logic for executing actions inside Second Life. Sensor data received by this LIBOMV client is communicated to the Snapshot Generator. The RDF version of this sensor data then travels through the Relation Identifier, Natural Language Processor, Complex Event Recogniser, and the Situation Identifier. Finally, the Situation Identifier sends the processed RDF data to the Jason Environment class. This RDF data is converted into an object representation, and is then converted to Jason percepts[21].

6.1 Monitoring for Durative Agent Actions

In our previous work [61, 59], we demonstrated how the aforementioned data processing components are used to process low-level Second Life sensor data into abstract environment information. In this section we demonstrate how this loosely coupled sensor data processing framework can be easily used by an agent to monitor for its durative actions inside Second Life.

When implementing IVAs, as well as maintaining awareness of the agent's environment, it is equally important to implement the behaviour execution of the IVA. Due to the nature of virtual environments, unlike in a grid-based system,

[19] We make use of the Apache Jena (`http://jena.apache.org/`) for this conversion.

[20] Please refer to Appendix H in [59] for full details of how this action is implemented in Jason.

[21] Please refer to Section 8.1.2 in [59] to see how a Jason agent reacts to these different perceptual information received from the data processing framework.

most of the agent actions executed in a virtual environment are durative. For example, consider an agent moving from one location to another. In a grid-based system, this movement means moving the agent from one grid location to another, which can be accomplished within one execution cycle. In contrast, in a virtual environment, this movement may take more than one execution cycle of the agent, depending on the distance between the two locations. Therefore, when executing such durative actions, it is important that the agent is aware of the continuing progress and completion of these actions. Dignum et al. [17] highlight the importance of notifying an agent about the result of the execution of its actions, because "actions in the game world are not always executed right away, do not always succeed, and sometimes have unexpected results". They also note that simply waiting for an action to complete might not be practical, because some agent actions might never be completed inside the virtual world. Therefore some mechanism is needed to notify the agent about the progress of its actions in an efficient manner. To address this issue, the CIGA middleware [51] defines an *action percept* that contains feedback about the on-going actions of the agent.

Although providing a full account of IVA behaviour execution is out of the scope of this paper, we briefly discuss how our data processing framework is capable of monitoring and notifying the agent about its ongoing actions, as this provides further evidence of the benefits of our framework, and demonstrates that communication between different components can be set up easily.

For monitoring the progress of agent actions, we make use of the complex event recogniser we have already implemented. However, it is equally possible and straightforward to plug in a new component (e.g. the Python-based model checking tool presented by Cranefield and Winikoff [14]) to take responsibility for this.

According to the communication flow described above, when an agent issues an action command, the Jason Environment class notifies the VW Client to request the agent's corresponding LIBOMV client to start executing this action. To handle durative actions, in parallel with this, the Environment class uses XMPP messaging to send one or two Esper patterns to the Complex Event Recogniser to monitor for the *completion* of the action, and (optionally) also for its *continuation*. In the current implementation, each durative action we define in Jason has these one or two corresponding Esper patterns defined in the Jason Environment class, with place holders that are filled in using the parameters in the action request.

Each Esper pattern sent by the Environment class is dynamically added to the list of patterns of the Esper engine. The on-demand query and pattern processing feature in Esper is used to start and stop these Esper patterns. When the corresponding event is recognised, the Complex Event Recogniser pushes this event notification to the Environment class through the XMPP server. When the agent gets the notification of the result of its action execution, it notifies the Complex Event Recogniser to stop monitoring for this action. Establishing this new communication route between the Environment class and the Complex Event Recogniser was straightforward, due to the flexible communication mechanism supported by the XMPP server and BSF.

As an example, below we show the two Esper patterns implemented to monitor for the continuation and the completion of the durative action that corresponds to the IVA moving from one location to another.

The first pattern (Listing 1) checks if the action is continuing—whether the distance between the avatar and the target location keeps on reducing. The second

Listing 1 Pattern to check for continuation of an action

```
SELECT * FROM PATTERN
[
EVERY
a = EntitySnapshot(EU.GetContextVal(Entity.PropVals, "EntityName") = agent-name)
->
b = EntitySnapshot(EU.GetContextVal(Entity.PropVals, "EntityName") = agent-name,
                   EU.GetDistance(EU.GetContextVal(Entity.PropVals, "Position"),
                                  target-position) <
                     EU.GetDistance(EU.GetContextVal(a.Entity.PropVals, "Position"),
                                    target-position),
                   SnapshotIndex = a.SnapshotIndex+1)
]
```

Listing 2 Esper pattern to check for completion of an action

```
SELECT * FROM PATTERN
[
EVERY
 a = EntitySnapshot(EU.GetContextVal(Entity.PropVals, "EntityName") = agent-name,
                    EU.GetDistance(EU.GetContextVal(Entity.PropVals, "Position"),
                                   target-position)
                    > 0)
 ->
 b = EntitySnapshot(EU.GetContextVal(Entity.PropVals, "EntityName") = agent-name,
                    EU.GetDistance(EU.GetContextVal(Entity.PropVals, "Position"),
                                   target-position)
                    = 0,
                    SnapshotIndex = a.SnapshotIndex+1)
]
```

pattern (Listing 2) checks if the action has completed—whether the avatar has reached the target location. The variables `a` and `b` are aliases for data items of type `EntitySnapshot`, and '`->`' indicates that `a` is 'followed by' `b`. The `EVERY` keyword applies this pattern to each pair of `EntitySnapshot`s. The `SnapshotIndex` property is used to identify two consecutive snapshots. `EU` stands for our `EsperUtility` C# class that contains complex logic that cannot be expressed in Esper logic. For example, `EU.GetContextVal` method returns the value of the given entity property.

7 Comparison with Existing Systems

In this section, we compare our loosely coupled data processing framework with two existing systems for interfacing agent systems and virtual worlds: Pogamut and CIGA.

7.1 Pogamut

The Pogamut framework was initially presented as "an open source platform for rapid development of behaviour for virtual agents embodied in a 3D environment of the Unreal Tournament 2004 video game" [26]. As the description implies, Pogamut has been designed specifically to make use of the Unreal Tournament 2004 videogame for research and educational purposes, and it has become popular as

an IVA research and teaching tool. However, later versions of the framework indicate generalisation of this work into an "interface bi-directionally bridging the representational gap between a game world and an external DMS" [25], where the acronym DMS denotes an agent decision-making system.

Due to the fact that Pogamut was first developed in order to interface agents with one specific game engine, it has not addressed the idea of a common ontology for the environments provided by different virtual worlds. It makes use of the Java class hierarchy to allow different levels of abstraction, which the authors present as an analogue to semantic ontologies [26]. Although it might be adaptable, how this class hierarchy might accommodate the heterogeneous information of different virtual worlds is not clear. Pogamut researchers have later introduced a formal description of game engines [25], however as explained in Section 3.1, this formal description focuses more on game engine and agent internals. As such, we believe this formalisation targets a different problem to that which we address, and their description is unlikely to be applicable for processing the sensor data received from a virtual world, in contrast to how our ontology is used by the data processing components.

As with our framework, Pogamut has identified the need for different data processing components to process the low-level sensor data received from a virtual world. With this in mind, Pogamut has components referred to as the working memory, inference engine and reactive layer. However, in contrast to our loosely coupled framework, these components are tightly connected to each other. Currently Pogamut has no option to incorporate a new data processing component that has been implemented in a programming language that is not compatible with Java. Even the agent system (currently based on AgentSpeak(L)) is tightly integrated with the data processing components. It would be interesting to see how the loosely coupled data processing framework presented in this paper can be accommodated to remove this rigid component coupling of Pogamut.

Similar to our VW Client, Pogamut also has a separate component that interfaces with the remote virtual world. This is currently fully implemented for Unreal Tournament 2004. If the need arises to connect to another virtual world, this component would have to be replaced with the logic corresponding to the new virtual world. However, it is not clear how the heterogeneous nature of sensor data received from different virtual worlds are processed by this component before providing them to the data processing components. Finally, Pogamut focuses more on agent behaviour execution inside a virtual environment, while the emphasis in our work is on the problem of processing virtual world sensor data for consumption by an (external) agent system.

7.2 CIGA

CIGA is a more recent system than Pogamut and is still undergoing in-house testing [51]. It has been presented as "a middleware to facilitate the coupling between agent technology and game technology, tackling the inherent design issues in a structured way" [51]. In other words, CIGA is motivated by the same principal factors as we are.

CIGA also emphasises the need for ontologies as a means to provide a design contract for the whole middleware. However, its approach to the design of the on-

tology is similar to the work of Chang et al. [12], where most of the focus has been on describing how an agent could interact with the environment objects. As a result, this ontology (the *social world model*) does not contain enough information to describe the dynamism in the virtual environment. Events have been identified as a part of the environment dynamism. However, the connection between one event and another, events and entities, or events and situations has not been explored, unlike the ontology we present here. Nevertheless, both aspects—their social world model and our dynamic virtual environment ontology—are equally important to describing a virtual environment, in that they complement one another. Therefore, future work could focus on evolving these two separate ontologies towards making them compatible, even to the extent of constructing one comprehensive virtual environment ontology.

A shortcoming of CIGA's social world model is that it does not contain any reference to a virtual world-specific ontology, again unlike the ontology we present. Therefore it is not clear how this social world model would accommodate the different sensor data interfaces provided by different virtual worlds.

The CIGA middleware addresses the problem of sensor data processing, as well as behaviour execution, whereas we focus only on the former. A rather noticeable limitation of CIGA is that it assumes that it is possible to modify the game engine internals to associate the CIGA interface with it. It also relies on direct access to game engine queries to retrieve and generate periodic sensor data updates to the agent, and expects the game engine logic to be aware of the social world model. However, modifying game engine internals may well not be possible, particularly in the context of proprietary virtual worlds. In contrast, our framework is decoupled in that it relies only on the sensor data interface and the communication protocol provided by the virtual world. If the virtual world allows modifications to its internal logic, our VW Client could be directly attached to it. However, this would have no impact on the rest of the framework, which is loosely coupled to the VW Client.

The CIGA interface connected to the virtual world is responsible for generating periodic state updates of the virtual environment, thus controlling the rate of communication of sensor data to an agent. However, this may not be always appropriate, especially when the virtual world makes use of an event-based approach to communicate sensor data to the clients. In contrast, our Snapshot Generator is flexible in generating state updates whenever an event notification and/or a periodic state update is received from the virtual world. Both these approaches are used in the context of Second Life in order to minimise the loss of information (see Section 6) when generating snapshots from Second Life sensor data. The CIGA middleware also assumes that the inference logic to generate abstract environment information can be done by one single data processing component. In contrast, we have identified the need for the integration of several different types of data processing components in order to bridge the information representation gap between virtual worlds and agent systems. Finally, CIGA makes use of TCP/IP for its communication model, thus facing most, if not all the difficulties described in Section 5.

8 Performance Evaluation

Our middleware framework enables virtual world sensor data to be abstracted for consumption by agents using a loosely coupled and dynamically configurable set of data processing components. We have also chosen to use a semantic representation of the data to provide a generic solution that is independent of any particular virtual world and agent platform. However, as always, providing a flexible and generic solution may come at a performance cost. In particular, communicating via an XMPP server requires (in the worst case, when the server is on a different host from the clients) two message exchanges across a network, and RDF encoding of data generates larger messages than when using platform- or application-specific encodings.

It is therefore important to evaluate the performance of our framework to gain assurance of its usability for practical applications. However, any specific application of our framework will involve multiple communicating components, with their individual capabilities and performance profiles, and the application-specific interactions between them, having a significant impact on the overall system performance. Furthermore, the ultimate measure of an intelligent virtual agent is its performance from the viewpoint of human participants in a deployed virtual world scenario. Evaluating our framework from this perspective is a significant undertaking that is beyond the scope of this paper. In this section we focus on measuring the communication time for sending snapshots between the data processing components[22] connected via the XMPP server and BSF. The performance of individual components is reported elsewhere [59]. At the end of this section we discuss some possible future developments that may enable optimisation of our approach when required.

In our performance test, the XMPP server (OpenFire 3.8.1) was running under Java 1.6.0_18 on a virtual CentOS 5.9 Linux server on a Citrix Xen 5 host[23]. The data processing components, written in Visual C# 2010, were set up on a desktop PC[24] on the same local area network as the server. A fixed sensor data record (snapshot) recorded previously from the VW client (see Fig. 8 for an example) was generated every 100ms (equivalent to 10 frames/second) and sent through the data processing components shown in Fig. 7. Each component processed the data in turn and forwarded the result via XMPP to the next component according to the following processing chain: Snapshot Generator, Relationship Generator, Natural Language Processor, Complex Event Detector, and Situation Identifier. For each snapshot, the average time to travel between each pair of consecutive components was recorded[25]. Fig. 10 shows box plots showing the spread of these average snapshot communication times for 10 runs of 30 minutes duration. In each box plot, the rectangle represents the central 50% of values, and the line within the rectangle represents the median value. The "whiskers" protruding from the

[22] This evaluation was done using an earlier version of the ontologies.

[23] The Java virtual machine configuration options were: -Xms32m -Xmx512m -Xss128k -Xoss128k -XX:ThreadStackSize=128. The server had a 1Gb dedicated NIC, 4Gb of RAM, and two virtual CPUs, each a core of a Xeon X5355@2.66Ghz.

[24] The PC was running Microsoft Windows 7 Enterprise and had a 3.4GHz four core Intel i5-3570 CPU.

[25] Since all the components were running in the same machine, it was possible to record this time using the system time.

top and bottom of each box indicate the most extreme values that were no further from the box than 1.5 times the interquartile range (the height of the box). As can be seen, the median of the average snapshot times to travel from one component to the other through the XMPP server is about 10ms.

To put this value into context, tests with the Jason BDI agent platform showed that, for one simple agent programme at least, percepts were retrieved from the agent's environment approximately every 500ms. If we consider the experience of human participants in a virtual world, one study on first-person shooter video games showed that a frame rate of 30 frames per second (i.e. an update every 33ms) is needed to provide visually smooth animation [13]. We would argue that the reactivity of a virtual agent to situational information would not be expected to match this update frequency, and that reactions an order of magnitude slower could be acceptable. Our experiments indicate such performance is achievable, even in the context of a proof-of-concept system.

We do note that there are a considerable number of outliers in our performance data, shown as circles above the upper whiskers in Fig. 10. This suggests that further investigation into the performance and optimisation of the selected XMPP server is needed, although this is unlikely to be the only significant factor.

In our approach, XMPP-based middleware is chosen to provide run-time flexibility in the deployment of sensory data between processing components and the flow of messages between them. However, this flexibility will not be needed in all applications. One desirable extension of this work would therefore be to provide the option of configuring a predefined set of data processing components and a fixed flow of data. This would then allow the XMPP server to be bypassed, with the processing components sending RDF-encoded snapshots directly to each other via some appropriate protocol. In this way, the framework would provide a user-configurable trade-off between run-time adaptiveness and efficiency.

A possible approach to reducing the cost of sending RDF data across the network is to investigate a more compact encoding for RDF, such as the binary HDT format for RDF, which compresses an RDF graph while also supporting indexed access to triples [22]. However, an additional overhead would then be incurred by the base64 encoding that is needed to include binary data within XMPP messages.

9 Conclusion

The aim of this paper has been to explore solutions to the information representation gap that exists between virtual worlds and agent systems such as those based on the BDI architecture, to improve IVAs' awareness of their dynamic environment.

Currently there exists no common terminology or ontology to describe abstractly this virtual-world-specific sensor data, which is of a heterogeneous nature. The solution to this problem is to develop a way of abstractly describing virtual environment dynamism in a form suitable for the deliberation process of an agent. Moreover, in order to convert this low-level sensor data into an abstract form, many processing steps are needed. However, it is not possible to come up with a generic set of data processing steps, due to the heterogeneous nature of virtual world sensor data, the differences in simulated application domains, and the

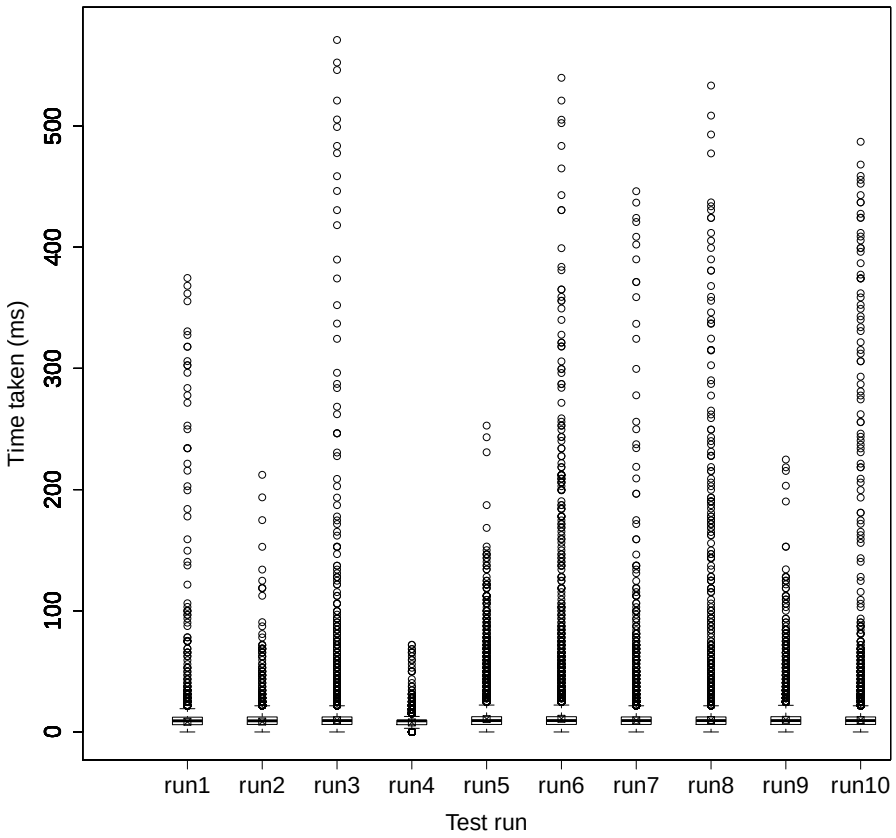

Fig. 10 Box plots for the average communication time between two components for 10 runs

types of reasoning needed by agents. Solving this problem requires a framework that can provide the flexibility in adding and removing different data processing components according to the selected application domain, virtual world, and the information requirements of the agent.

In line with these two requirements, this paper presented a dynamic virtual environment ontology, and middleware that supports the integration of loosely coupled data processing components in order to carry out the information conversion needed between agent systems and virtual worlds. Use of this middleware in interfacing Jason agent development platform with Second Life was described, along with a description of an implementation for monitoring durative Jason agent actions inside Second Life. Our semantic approach and the loosely coupled data processing framework were compared to two existing approaches for interfacing agent systems with virtual worlds. Finally, a performance evaluation of this implemented framework was also presented, which suggests that our approach will be usable for practical applications.

Acknowledgements Surangika Ranathunga was supported by a Postgraduate Publishing Bursary from the University of Otago, New Zealand.

References

1. Adobbati, R., Marshall, A.N., Scholer, A., Tejada, S., Kaminka, G., Schaffer, S., Sollitto, C.: Gamebots: A 3D virtual world test-bed for multi-agent research. In: Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS, pp. 47–52 (2001)
2. Agostaro, F., Augello, A., Pilato, G., Vassallo, G., Gaglio, S.: A conversational agent based on a conceptual interpretation of a data driven semantic space. In: S. Bandini, S. Manzoni (eds.) AI*IA 2005: Advances in Artificial Intelligence, *Lecture Notes in Computer Science*, vol. 3673, pp. 381–392. Springer (2005)
3. Aranda, G., Carrascosa, C., Botti, V.: The MMOG layer: MMOG based on MAS. In: Agents for Games and Simulations, *Lecture Notes in Computer Science*, vol. 5920, pp. 63–78. Springer (2009)
4. Behrens, T., Hindriks, K.V., Dix, J.: Towards an environment interface standard for agent platforms. Annals of Mathematics and Artificial Intelligence **61**, 4 (2010)
5. Bogdanovych, A., Rodriguez-Aguilar, J.A., Simoff, S., Cohen, A.: Authentic interactive reenactment of cultural heritage with 3D virtual worlds and artificial intelligence. Applied Artificial Intelligence **24**(6), 617–647 (2010)
6. Bogdanovych, A., Simoff, S.J., Esteva, M.: Training believable agents in 3D electronic business environments using recursive-arc graphs. In: Proceedings of the Third International Conference on Software and Data Technologies, pp. 339–346. INSTICC Press (2008)
7. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason. John Wiley & Sons Ltd, England (2007)
8. Bosse, T., Stam, S.: A normative agent system to prevent cyberbullying. In: O. Boissier, J. Bradshaw, L. Cao, K. Fischer (eds.) Proceedings of the Eleventh IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, vol. 2, pp. 425 –430. IEEE Computer Society Press (2011)
9. Bromuri, S., Stathis, K.: Situating cognitive agents in GOLEM. In: D. Weyns, S. Brueckner, Y. Demazeau (eds.) Engineering Environment-Mediated Multi-Agent Systems, *Lecture Notes in Computer Science*, vol. 5049, pp. 115–134. Springer (2008)
10. Buford, J., Jakobson, G., Lewis, L.: Multi-agent situation management for supporting large-scale disaster relief operations. International Journal of Intelligent Control and Systems **11**(4), 284–295 (2006)
11. Cavazza, M., Palmer, I.: High-level interpretation in virtual environments. Applied Artificial Intelligence **14**(1), 125–144 (2000)
12. Chang, P.H.M., Chen, K.T., Chien, Y.H., Kao, E., Soo, V.W.: From reality to mind: A cognitive middle layer of environment concepts for believable agents. In: D. Weyns, H. Van Dyke Parunak, F. Michel (eds.) Environments for Multi-Agent Systems, *Lecture Notes in Computer Science*, vol. 3374, pp. 57–73. Springer (2005)
13. Claypool, K.T., Claypool, M.: On frame rate and player performance in first person shooter games. Multimedia Systems **13**(1), 3–17 (2007)
14. Cranefield, S., Winikoff, M.: Verifying social expectations by model checking truncated paths. Journal of Logic and Computation **21**(6), 1217–1256 (2011)
15. Das, S.: High-Level data fusion. ARTECH HOUSE, INC. (2008)
16. Dignum, F.: Agents for games and simulations. Autonomous Agents and Multi-Agent Systems **24**(2), 217–220 (2012)
17. Dignum, F., Westra, J., van Doesburg, W.A., Harbers, M.: Games and agents: Designing intelligent gameplay. International Journal of Computer Games Technology **2009**, 1–18 (2009)
18. Ehlert, P.A.M., Mouthaan, Q.M., Rothkrantz, L.: A rule-based and a probabilistic system for situation recognition in a flight simulator. In: Q.H. Mehdi, N. Gough, S. Natkin (eds.) Proceedings of the 4th International Conference on Intelligent Games and Simulation, pp. 201–207. Eurosis (2003)
19. Ellis, S.R.: What are virtual environments? IEEE Computer Graphics and Applications **14**(1), 17–22 (1994)

20. Endsley, M.R.: Toward a theory of situation awareness in dynamic systems. Human Factors: The Journal of the Human Factors and Ergonomics Society **37**(1), 32–64 (1995)
21. Feng, Y.H., Teng, T.H., Tan, A.H.: Modelling situation awareness for context-aware decision support. Expert Systems with Applications **36**(1), 455 – 463 (2009)
22. Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary RDF representation for publication and exchange (HDT). Web Semantics **19**, 22–41 (2013)
23. Fielding, D., Fraser, M., Logan, B., Benford, S.: Extending game participation with embodied reporting agents. In: Proceedings of the 2004 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology, pp. 100–108. ACM (2004)
24. Gemrot, J., Brom, C., Kadlec, R., Bída, M., Burkert, O., Zemčák, M., Píbil, R., Plch, T.: Pogamut 3 – Virtual humans made simple. In: J. Gray (ed.) Advances in Cognitive Science, pp. 211–243. The Institution of Engineering and Technology (2010)
25. Gemrot, J., Brom, C., Plch, T.: A periphery of Pogamut: From bots to agents and back again. In: F. Dignum (ed.) Agents for Games and Simulations II, *Lecture Notes in Computer Science*, vol. 6525, pp. 19–37. Springer (2011)
26. Gemrot, J., Kadlec, R., Bída, M., Burkert, O., Píbil, R., Havlíč ek, J., Zemčák, L., Šimlovič, J., Vansa, R., Štolba, M., Plch, T., Brom, C.: Pogamut 3 can assist developers in building AI (not only) for their videogame agents. In: F. Dignum, J. Bradshaw, B. Silverman, W. van Doesburg (eds.) Agents for Games and Simulations, *Lecture Notes in Computer Science*, vol. 5920, pp. 1–15. Springer (2009)
27. Gregori, M.E., Cámara, J.P., Bada, G.A.: A Jabber-based multi-agent system platform. In: Proceedings of the Fifth International Joint Conference on Autonomous agents and Multiagent systems, pp. 1282–1284. ACM (2006)
28. Grimaldo, F., Lozano, M., Barber, F., Vigueras, G.: Simulating socially intelligent agents in semantic virtual environments. Knowledge Engineering Review **23**(4), 369–388 (2008)
29. Guinn, C.I., Montoya, R.J.: Natural language processing in virtual reality training environments. In: Proceedings of the Interservice/Industry Training Systems and Education Conference, pp. 44–55. National Training Systems Association (1998)
30. Gutierrez, M., Vexo, F., Thalmann, D.: Semantics-based representation of virtual environments. International Journal of Computer Applications in Technology **23**(2), 229–238 (2005)
31. Helleboogh, A., Vizzari, G., Uhrmacher, A., Michel, F.: Modeling dynamic environments in multi-agent simulation. Autonomous Agents and Multi-Agent Systems **14**(1), 87–116 (2007)
32. Hill, R.W., Gratch, J., Marsella, S., Rickel, J., Swartout, W.R., Traum, D.R.: Virtual humans in the mission rehearsal exercise system. Künstliche Intelligenz **4**(3), 5–10 (2003)
33. Hindriks, K.V., van Riemsdijk, B., Behrens, T., Korstanje, R., Kraayenbrink, N., Pasman, W., de Rijk, L.: UnREAL Goal bots: Conceptual design of a reusable interface. In: F. Dignum (ed.) Agents for Games and Simulations II, *Lecture Notes in Computer Science*, vol. 6525, pp. 1–18. Springer (2011)
34. Ijaz, K., Bogdanovych, A., Simoff, S.: Enhancing the believability of embodied conversational agents through environment-, self- and interaction-awareness. In: Proceedings of the Thirty-Fourth Australasian Computer Science Conference, vol. 113, pp. 107–116. Australian Computer Society (2011)
35. Irawati, S., Calderón, D., Ko, H.: Spatial ontology for semantic integration in 3D multimodal interaction framework. In: Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and its Applications, pp. 129–135. ACM (2006)
36. Jan, D., Roque, A., Leuski, A., Morie, J., Traum, D.: A virtual tour guide for virtual worlds. In: Z. Ruttkay, M. Kipp, A. Nijholt, H. Vilhjálmsson (eds.) Intelligent Virtual Agents, *Lecture Notes in Computer Science*, vol. 5773, pp. 372–378. Springer (2009)
37. Kalogerakis, E., Christodoulakis, S., Moumoutzis, N.: Coupling ontologies with graphics content for knowledge driven visualization. In: Proceedings of the IEEE Conference on Virtual Reality, pp. 43–50. IEEE Computer Society Press (2006)
38. Kao, E.C.C., Chang, P.H.M., Chien, Y.H., Soo, V.W.: Using ontology to establish social context and support social reasoning. In: T. Panayiotopoulos, J. Gratch, R. Aylett, D. Ballin, P. Olivier, T. Rist (eds.) Intelligent Virtual Agents, *Lecture Notes in Computer Science*, vol. 3661, pp. 344–357. Springer (2005)
39. Kenny, P., Parsons, T., Gratch, J., Leuski, A., Rizzo, A.: Virtual patients for clinical therapist skills training. In: C. Pelachaud, J.C. Martin, E. André, G. Chollet, K. Karpouzis, D. Pelé (eds.) Intelligent Virtual Agents, *Lecture Notes in Computer Science*, vol. 4722, pp. 197–210. Springer (2007)

40. Kopp, S., Gesellensetter, L., Krämer, N., Wachsmuth, I.: A conversational agent as museum guide – design and evaluation of a real-world application. In: T. Panayiotopoulos, J. Gratch, R. Aylett, D. Ballin, P. Olivier, T. Rist (eds.) Intelligent Virtual Agents, *Lecture Notes in Computer Science*, vol. 3661, pp. 329–343. Springer (2005)
41. Labidi, W., Susini, J.F., Paradinas, P., Setton, M.: XMPP based health care integrated ambient systems middleware. In: Developing Ambient Intelligence, pp. 92–102. Springer (2008)
42. Lee, J., Baines, V., Padget, J.: Decoupling cognitive agents and virtual environments. In: F. Dignum, C. Brom, K. Hindriks, M. Beer, D. Richards (eds.) Cognitive Agents for Virtual Environments, *Lecture Notes in Computer Science*, vol. 7764, pp. 17–36. Springer Berlin Heidelberg (2013)
43. Lee, J., Li, T., Padget, J.: Towards polite virtual agents using social reasoning techniques. Computer Animation and Virtual Worlds **24**(3-4), 335–343 (2013). DOI 10.1002/cav.1517. URL `http://dx.doi.org/10.1002/cav.1517`
44. Lim, C.U., Baumgarten, R., Colton, S.: Evolving behaviour trees for the commercial game DEFCON. In: C. Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. Esparcia-Alcazar, C.K. Goh, J. Merelo, F. Neri, M. Preuß, J. Togelius, G. Yannakakis (eds.) Applications of Evolutionary Computation, *Lecture Notes in Computer Science*, vol. 6024, pp. 100–110. Springer (2010)
45. Loyall, A.B.: Believable agents: Building interactive personalities. Ph.D. thesis, Carnegie Mellon University (1997)
46. Mumme, C., Pinkwart, N., Loll, F.: Design and implementation of a virtual salesclerk. In: Z. Ruttkay, M. Kipp, A. Nijholt, H. Vilhjálmsson (eds.) Intelligent Virtual Agents, *Lecture Notes in Computer Science*, vol. 5773, pp. 379–385. Springer (2009)
47. Norling, E.: Capturing the Quake player: Using a BDI agent to model human behaviour. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1080–1081. ACM (2003)
48. Norling, E., Sonenberg, L.: Creating interactive characters with BDI agents. In: Proceedings of the Autralian Workshop on Interactive Entertainment, pp. 69–76 (2004)
49. Object Management Group: Ontology Definition Metamodel version 1.0. OMG document formal/2009-05-01, `http://www.omg.org/spec/ODM/1.0/` (2009)
50. Odell, J.J., Parunak, H.V.D., Fleischer, M., Brueckner, S.: Modeling agents and their environment. In: F. Giunchiglia, J. Odell, G. Weiß(eds.) Agent-Oriented Software Engineering III, *Lecture Notes in Computer Science*, vol. 2585, pp. 16–31. Springer (2003)
51. van Oijen, J., Vanhée, L., Dignum, F.: CIGA: A middleware for intelligent agents in virtual environments. In: M. Beer, C. Brom, F. Dignum, V.W. Soo (eds.) Agents for Educational Games and Simulations, *Lecture Notes in Computer Science*, vol. 7471, pp. 22–37. Springer (2012)
52. Okuyama, F.Y., Bordini, R.H., da Rocha Costa, A.C.: ELMS: An environment description language for multi-agent simulation. In: D. Weyns, H.V.D. Parunak, F. Michel (eds.) Environments for Multi-Agent Systems, *Lecture Notes in Computer Science*, vol. 3374, pp. 91–108. Springer (2005)
53. Orkin, J.: Three states and a plan: The AI of F.E.A.R. In: Proceedings of the Game Developers Conference (2006). `http://web.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf`
54. Orkin, J., Smith, T., Reckman, H., Roy, D.: Semi-automatic task recognition for interactive narratives with EAT & RUN. In: Proceedings of the Intelligent Narrative Technologies III Workshop. ACM (2010)
55. Ossowski, S. (ed.): Agreement Technologies, *Law, Governance and Technology Series*, vol. 8. Springer (2013)
56. Otto, K.A.: The semantics of multi-user virtual environments. In: Proceedings of the Workshop towards Semantic Virtual Environments, pp. 35–39 (2005)
57. Parsia, B., Sattler, U.: OWL 2 web ontology language data range extension: Linear equations. W3C Working Group Note, `http://www.w3.org/TR/2012/NOTE-owl2-dr-linear-20121018/` (2012)
58. Parunak, H.V.D.: Go to the ant: Engineering principles from natural agent systems. Annals of Operations Research **75**(1), 69–101 (1997)
59. Ranathunga, S.: Improving awareness of intelligent virtual agents. Ph.D. thesis, University of Otago (2013)
60. Ranathunga, S., Cranefield, S., Purvis, M.: Extracting data from Second Life. Discussion Paper 2011/07, Department of Information Science, University of Otago (2011). `http://otago.ourarchive.ac.nz/handle/10523/1802`

61. Ranathunga, S., Cranefield, S., Purvis, M.: Identifying events taking place in Second Life virtual environments. Applied Artificial Intelligence **26**(1–2), 137–181 (2012)
62. Ricci, A., Piunti, M., Viroli, M.: Environment programming in multi-agent systems: An artifact-based perspective. Autonomous Agents and Multi-Agent Systems **23**(2), 158–192 (2011)
63. Russell, S.J., Norvig, P.: Artificial Intelligence: A modern approach. Prentice Hall (2010)
64. Searle, J.R.: Speech Acts: An essay in the philosophy of language. Cambridge University Press (1969)
65. Small, S.G., Stromer-Galley, J., Strzalkowski, T.: Multi-modal annotation of quest games in Second Life. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, vol. 1, pp. 171–179. Association for Computational Linguistics (2011)
66. So, R., Sonenberg, L.: The roles of active perception in intelligent agent systems. In: D. Lukose, Z. Shi (eds.) Multi-Agent Systems for Society, *Lecture Notes in Computer Science*, vol. 4078, pp. 139–152. Springer (2009)
67. Stolcke, A., Ries, K., Coccaro, N., Shriberg, E., Bates, R., Jurafsky, D., Taylor, P., Martin, R., Ess-Dykema, C., Meteer, M.: Dialogue act modeling for automatic tagging and recognition of conversational speech. Computational linguistics **26**(3), 339–373 (2000)
68. Stout, L., Murphy, M.A., Goasguen, S.: Kestrel: an XMPP-based framework for many task computing applications. In: Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers. ACM (2009)
69. Sukthankar, G., Sycara, K.: A cost minimization approach to human behavior recognition. In: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1067–1074. ACM (2005)
70. Swartout, W., Traum, D., Artstein, R., Noren, D., Debevec, P., Bronnenkant, K., Williams, J., Leuski, A., Narayanan, S., Piepol, D., Lane, C., Morie, J., Aggarwal, P., Liewer, M., Chiang, J.Y., Gerten, J., Chu, S., White, K.: Ada and Grace: Toward realistic and engaging virtual museum guides. In: J. Allbeck, N. Badler, T. Bickmore, C. Pelachaud, A. Safonova (eds.) Intelligent Virtual Agents, *Lecture Notes in Computer Science*, vol. 6356, pp. 286–300. Springer (2010)
71. Thangarajah, J., Padgham, L., Sardina, S.: Modelling situations in intelligent agents. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1049–1051. ACM (2006)
72. Torres, J., Nedel, L., Bordini, R.: Using the BDI architecture to produce autonomous characters in virtual worlds. In: T. Rist, R. Aylett, D. Ballin, J. Rickel (eds.) Intelligent Virtual Agents, *Lecture Notes in Computer Science*, vol. 2792, pp. 197–201. Springer (2003)
73. van Dijk, E.M.A.G., op den Akker, H.J.A., Nijholt, A., Zwiers, J.: Navigation assistance in virtual worlds. Informing Science, Special Series on Community Informatics **6**, 115–125 (2003)
74. Vosinakis, S., Panayiotopoulos, T.: Programmable agent perception in intelligent virtual environments. In: T. Rist, R. Aylett, D. Ballin, J. Rickel (eds.) Intelligent Virtual Agents, *Lecture Notes in Computer Science*, vol. 2792, pp. 202–206. Springer (2003)
75. Wagener, J., Spjuth, O., L, E.W., Wikberg, J.E.: XMPP for cloud computing in bioinformatics supporting discovery and invocation of asynchronous web services. BMC Bioinformatics **10**(1) (2009)
76. Warden, T., Visser, U.: Real-time spatio-temporal analysis of dynamic scenes. Knowledge and Information Systems **32**(2), 243–279 (2012)
77. Weis, G., Lewis, A.: Using XMPP for ad-hoc grid computing – An application example using parallel ant colony optimisation. In: IEEE International Symposium on Parallel Distributed Processing. IEEE (2009)
78. Weiss, G. (ed.): Multiagent Systems, 2^{nd} edn. MIT Press (2013)
79. Wernert, E.A., Hanson, A.J.: A framework for assisted exploration with collaboration. In: Proceedings of the Conference on Visualization '99: Celebrating Ten Years, pp. 241–248. IEEE Computer Society Press (1999)
80. Weyns, D., Omicini, A., Odell, J.: Environment as a first class abstraction in multiagent systems. Autonomous Agents and Multi-Agent Systems **14**(1), 5–30 (2007)
81. Weyns, D., Steegmans, E., Holvoet, T.: Towards active perception in situated multi-agent systems. Applied Artificial Intelligence **18**(9-10), 867–883 (2004)
82. Zhang, W., Hill, J., W., R.: A template-based and pattern-driven approach to situation awareness and assessment in virtual humans. In: Proceedings of the Fourth International Conference on Autonomous Agents, pp. 116–123. ACM (2000)

83. Ziafati, P., Dastani, M., Meyer, J.J., van der Torre, L.: Event-processing in autonomous robot programming. In: Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems, pp. 95–102. IFAAMAS (2013)