Combining Object-oriented and Ontology-based Approaches in Human Behaviour Modelling

M. Kvassay*, L. Hluchý*, B. Kryza**, J. Kitowski**, M. Šeleng*, Š. Dlugolinský * and M. Laclavík *
* Institute of Informatics, Slovak Academy of Sciences, Bratislava, Slovakia
{marcel.kvassay, hluchy.ui, martin.seleng, stefan.dlugolinsky, laclavik.ui }@savba.sk
** Academic Computer Centre CYFRONET, University of Science and Technology in Cracow, Poland
{bkryza, kito}@agh.edu.pl

***Abstract*—This article proposes a combination of object-oriented and ontology-based approaches for real-time interworking of human behaviour models in the context of agent-based simulation systems. We present a conceptual design of a semantic intermediation framework, including the split of the responsibilities between the intermediation ontology and software code. We illustrate our design in the context of the EDA project A-0938-RT-GC EUSAS, where it will be used for integrating various behaviour models and for virtual trainings running in real time. We also report the results of preliminary performance tests related to ontological queries, and conclude with our future plans concerning the intermediation infrastructure.**

I. INTRODUCTION

Human Behaviour Modelling is an important area of computational science with implications not only for social sciences, but also for economics, epidemiology and other fields. Scientific literature abounds in specialized models of human cognition, emotions and other socio-psychological functions and processes. One of the challenges in human behaviour modelling is how to make all these models interoperate given the fact that they are created by experts from diverse fields, and often for divergent purposes. The task gets even more difficult when the models have to interoperate in real time as happened in one of our recent research projects, the EDA project A-0938-RT-GC EUSAS. One of the goals of this project is to model crowd-phenomena and crowd behaviour, but also to use the system for virtual training of security forces. This means the simulated characters are expected to interact with real people (security forces) in a highly realistic 3-D cyber environment, which links our efforts with the areas of man-machine interaction and computational intelligence. A natural choice for simulations of this kind is a system with intelligent agents, although in human behaviour modelling we tackle the problem of intelligence from a different angle than typical Artificial Intelligence applications. Drawing on an apt characterization of this difference given in [1], we aim at agents that act "like humans" rather than rationally. In fact, capturing human irrationality in our models may be the key to successful reproduction of crowd-phenomena and crowd behaviour in our simulations.

We present here a conceptual design of a semantic solution to this problem, and illustrate it in the context of the EUSAS project where it will be used for integrating various types of human behaviour models.

A. Agent Models of Human Behaviour

There are several agent types and architectures that we considered. The simplest type – reactive – represents an automaton that reacts to the input signal and transforms it into output. More advanced types (behavioural and deliberative) have an internal representation of the external environment, plan their actions in advance and act purposively [2]. Intelligent agents may be capable of autonomous learning [3].

FSMs (Finite State Machines) represent a universal technique and standard model for simple automatons. But using FSMs for more advanced agent types leads to complicated design structures. A more effective solution was provided by newer agent architectures, such as BDI (Belief, Desire, Intention) [4]. BDI is ideally suited for the design of simple rational agents (e.g. robots), but it does not capture the full complexity of human nature. In BDI, it is difficult to represent complex human emotions and cognitive processes, or the influence on them of physical factors, such as fatigue, hunger, thirst and stress. Several recent agent architectures, e.g. EBDI [5], PMFserv [6] or PECS [7] try to overcome these limitations.

In this article we shall use the PECS reference model, since it was the model of choice for the EUSAS project, but our semantic approach could be used with other agent architectures as well. In the following sections we briefly characterize the PECS model and the modelling requirements of the EUSAS project.

B. The PECS Reference Model

The four letters of the acronym "PECS" stand for *Physical conditions, Emotional state, Cognitive capabilities* and *Social status*. According to [7], "PECS is a multi-purpose reference model for the simulation of human behaviour in a social environment," with emphasis on "emergent behaviour… typical of the formation of groups and societies in social systems." The context for this model is provided by the so-called "agent world," which comprises three kinds of entities: the environment, the connector and the agents. Put briefly, the agents communicate through the connector and interact in (also with) the external environment. For each agent, the other agents are a part of the environment.

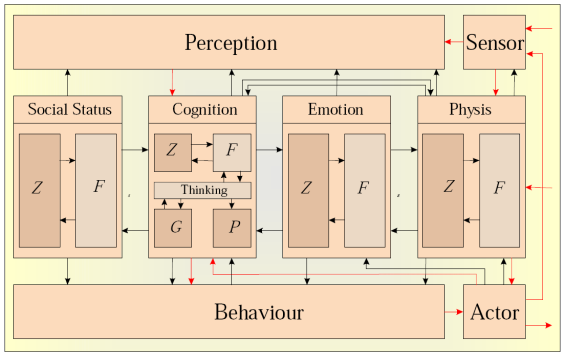

Figure 1. Structure of a PECS Agent (reproduced from [7])

The internal structure of PECS agents is shown in Figure 1. Black arrows represent causal dependencies, red arrows the actual flow of information. Following the general system theory, PECS agents are structured into input, internal state and output. Sensor and Perception components receive and pre-process the information from the environment, so they represent input. The middle four components (Social Status, Cognition, Emotion and Physis) together represent the internal state of the agent. Each of them has its own internal state variables (Z) along with their state transition function (F). Cognition component additionally includes goals (G), action plans (P) and thinking functions, so it is the most complicated component of the four. Behaviour and Actor components represent output: the first determines the actions to be executed, the second executes them.

In simple agents the state variables can directly determine the behaviour. In general, however, the situation is more complex. In the PECS model, behaviour is driven by the so-called "motives" that causally depend on (but are not identical with) state variables. A good example is the level of physical energy (state variable), and the perceived intensity of hunger (corresponding motive). There is a causal dependency, but the relationship is nonlinear. The PECS model introduces the function H that converts state variables into motives. This function needs to be defined so that various motives (hunger, fear, anger, etc.) are numerically comparable. During the simulation, the values of all the motives are evaluated at each time step, and the motive with the highest intensity becomes the action guiding motive.

The PECS reference model is universal; it serves as a blueprint for a whole class of real systems with a common deep structure, differing only in superficial qualities. The adaptation to individual conditions occurs by filling in the slots provided by the architecture (e.g. by defining state variables and their state transition functions) and by leaving out the slots that are not needed.

C. *Modelling Requirements of the EUSAS project*

The EUSAS project focuses on asymmetric security threats in urban terrain. Its goal is to improve the mission analysis capabilities as well as virtual training of real human beings (security forces) in a highly realistic 3-D cyber environment. This will be achieved by a detailed modelling of the behaviour of individuals and crowds on the basis of latest findings deriving from psychology, using the PECS reference model.

The project comprises an experimentation phase, development of models and formulation of conclusions, as well as recommendations for future mission planning. During simulations, various behaviour models are coordinated through an intermediation infrastructure and learning functionality. The main idea of the project is to use these common components (behaviour models, intermediation infrastructure and learning functionality) for both virtual training and mission analysis.

The mission analysis includes intensive experimentation based on Data Farming methodology, which will help military analysts evaluate and optimize military tactics and procedures, with the added advantage of using the analysis results also for improving the agent behaviour models. Similarly, the virtual training sessions will be logged and analyzed, and the analysis of tactics employed by the experienced security personnel (as opposed to novices) may help improve the rules of engagement as well as behaviour models.

The modelling tasks comprise environment modelling and behaviour modelling. Based on a detailed analysis and highly realistic scenario descriptions, environmental factors will be adequately modelled. Human beings in the scenario will be perceived as psychosomatic units with cognitive capacities embedded in a social environment. The modelled aspects include social, political and cultural influences, as well as group processes leading to the formation of demonstrating and rioting crowds.

The project's analysis of the state of the art in human behaviour modelling highlighted the missing components (models) of human characteristics and processes that would guarantee valid simulated behaviours in a broad range of applications. Also missing was the capability to interconnect models on different levels of abstraction, or models describing different, but mutually dependent, phenomena (such as social pressure and individual decision making) based on a semantic understanding of the concepts involved. Ontology-based technologies embedded in a semantic intermediation framework would be able to link such different types of models.

The EUSAS project will also explore the possibility to create and adjust behaviour models by behaviour cloning based on the multi-agent strategy discovering algorithm MASDA [10]. If successful, it would provide automatic or semi-automatic improvement of the behaviour models.

II. Design Considerations

A. *Semantic Intermediation Framework*

Right from the start, the EUSAS project envisaged a common ontology at the core of the intermediation infrastructure. This would guarantee a unified semantic description of all the behaviour models. The intention was to use the state of the art semantic web technologies based on RDF/OWL standards, which were successful in various areas but never used for behaviour models description and intermediation. In this respect, we could build on our earlier work, which included enhancing software agents with OWL-based knowledge models [8, 9].

Initially, we considered the ontology an obvious candidate for storing all the behaviour-related knowledge, but our subsequent analysis revealed potential drawbacks. First and most important, generic ontologies and reasoners tend to be slow, and our system was expected to perform

in real time. Second, the dynamics of human motives requires extensive numerical calculations, and storing numerical formulas in ontology and retrieving them through a reasoner would be inefficient. Third, we expected our behaviour models to require adjustments and fine-tuning, and for this we wanted to have a regular programming language (e.g. JAVA) at our disposal.

These reasons led us to the decision to split the behaviour-related knowledge in two parts. One part will be captured in the ontologies, the other will be captured directly in the software code of the objects (classes) representing the ontological entities in our system.

We also realized that the reasons favouring the use of ontologies were typically user-centred, while the reasons against them were simulation-centred and related to the real-time performance of the system. This gave us the opportunity to structure the system so as to derive the maximum benefit from each of the two approaches at places where it was most appropriate.

As a general rule, we store in ontologies all the static knowledge (facts that do not change during simulation) that is easily and naturally represented by ontological formalism. For instance, software agents are endowed with a certain repertoire of behaviour patterns and motives that trigger them. These links – which pattern is triggered by what motive – are typical examples of static facts. Because they do not change during the simulation, the software objects can often pre-load them and reason on them in advance so as to minimize direct access to ontologies during simulation.

Dynamic knowledge (facts that change during the simulation, such as objects positions or agent states) and knowledge that does not lend itself easily to ontological formalism will be captured directly by the software code. Since JAVA is currently our top candidate for implementation, we shall presume that the software objects representing ontological entities are implemented as JAVA classes.

Sections that follow give an overview of how the system as a whole is filled with information and then used for simulation. At the beginning, the developers of the system provide a set of pre-defined JAVA classes for the ontological entities that represent the elementary building blocks from which complete agents can be built. We refer to these building blocks as agent sub-components. The developers also fill in the ontologies with the information describing these sub-components and how they can be combined in agent definitions. With these two pre-conditions fulfilled, new agent types and scenarios can be created and used in simulations.

B. Role and Structure of Intermediation Ontology

As mentioned in the previous section, the ontologies provide for flexible configuration of the simulation scenarios. Although some attempts at supporting the environment and behaviour modelling with ontologies in multi-agent crowd simulation already exist [11][12][13], they do not provide a concise and universal way for application in simulation frameworks requiring both 2D and 3D simulations and complex behaviour models. Additionally, none of them covers the concepts necessary for military contexts, where specific types of weapons, actions and behaviour patterns must be modelled. The main goals of applying ontologies in our project include

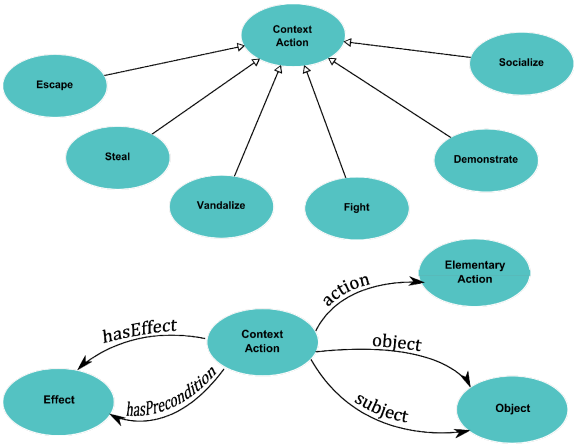

Figure 2. Excerpts from the intermediation ontology

such aspects as enhancing matchmaking capabilities for models, supporting agent-type creation from the existing behaviour elements, supporting agent context-specific object identification and supporting discovery and composition of action plans by agents based on their current motive and location in the environment. The ontology consists of generic parts, which are common for the entire system and for all scenarios, but these elements can be extended (by subsumption) with more specialized concepts and relations for particular scenarios (e.g. different objects can be found in a European city and different objects can be found in an Afghanistan camp).

The intermediation ontology is divided into two major parts: environmental and behavioural. The environment ontology provides the means for annotating the elements of the terrain and urban area, the dynamic objects which can be found and used by agents, weapons and control force facilities, as well as possible actions which can be performed on the objects in a given context. The main use of this part of the intermediation ontology will be during the environment modelling.

The behaviour ontology on the other hand provides means for annotating the different elements of agents in order to allow users to configure/build different agent types according to the requirements of a particular scenario. This entails, among others, concepts representing inputs and outputs of different elements of the behaviour model, or verification whether two parts can be combined in one agent type. Another important aspect of this ontology will be the modelling of typical behaviour patterns, both civilian and military. This will enable complex descriptions of possible behaviours of agents, support selection of behaviour patterns based on internal and external stimuli and modelling of agent motives. The behaviour ontology will also support the agent to select the most probable/appropriate action plan from the currently available possibilities.

An excerpt from the environment ontology presenting the core concept ContextAction and exemplary parts of object and action hierarchies is shown in Figure 2. Such structure allows to model actions and objects separately and to define custom action types which connect objects and actions including effects and preconditions. For instance, definition of an action of throwing a rigid object into a store window would be defined as follows:

$$\mathbf{ThrowRigidObjectAtStoreWindow} \sqsubseteq \mathbf{Vandalize} \sqcap \exists \mathbf{action.Throw} \sqcap \exists \mathbf{object.(RigidObject} \sqcup \mathbf{Dynamic)} \sqcap \exists \mathbf{subject.StoreWindow} \sqcap \exists \mathbf{hasPrecondition.AgentHasObject} \sqcap \exists \mathbf{hasEffect.\{IsOpened, IsDamaged\}}$$

This statement allows the ontology engine to return to the Agent-based Simulation component all the possible combinations of actions that can be performed on objects and subjects existing in the environment of the agent. Of course, even in the simplest scenario, the total number of combinations of all agents, objects and possible actions would be too large to handle in reasonable time. Thus the queries take into account, for each agent, the current action guiding motive (e.g. Vandalize, Escape, Steal), and can also filter out the objects invisible to the agent.

In order to get the initial performance results to see what can be the limitations of using ontologies in our use case, a sample test case based on our ontologies was performed for various combinations of number of objects and number of agents in the environment. The ontology itself consisted of 150 concepts describing various object and action types.

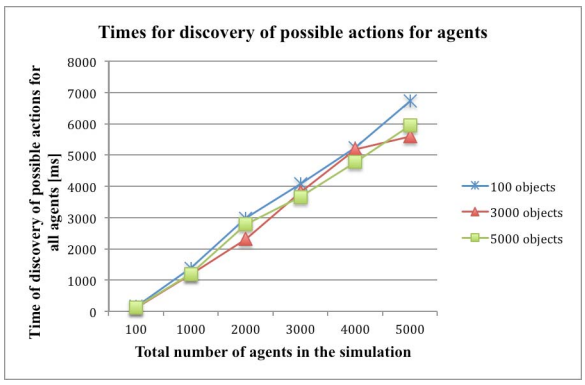

Figure 3. Total times of discovery of possible action plans for each agent for 100, 3000 and 5000 objects in the environment

The results show that the discovery of possible actions for agents in the simulation does not depend on the total number of objects in the environment, only on the number of agents, as the number of agents is proportional to the number of queries that have to be executed. Essentially, the results show that the performance impact is about 1s for every 1000 agents, which is acceptable for the purpose of data farming. The measured times do not include the reasoning time, as this is only performed once during the initialization of a particular simulation.

The tests were performed on Intel Dual Core 1.6 GHz machine, using Java 1.6, Jena [14] version 2.6.3 and Pellet [15] version 2.2.2 libraries. The Java heap space was set to 512 MB. All values were averaged over five independent runs of the same test. The queries assumed in any moment each agent can see a maximum of 30 objects.

C. *Defining New Agent Types and Scenarios*

Once the ontologies are in place, the users can define new agent types that suit their modelling needs and scenarios. Figure 4 illustrates the process on a very simple civilian agent (protester) endowed with just two motives (anger and fear) and two corresponding behaviour patterns

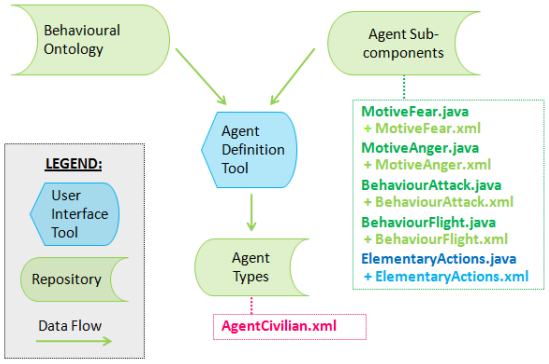

Figure 4. Definition of New Agent Types

("attack" and "flight to safety"). When anger predominates, the agent attacks the security forces (e.g. by throwing stones). This relieves the agent's anger, so fear takes over and triggers the "flight to safety" behaviour so as to avoid arrest.

The figure shows these agent sub-components (motives, behaviour patterns and elementary actions) as the required input for the agent definition tool. The elementary actions represent the building blocks from which behaviour pattern objects compose detailed action plans at run-time. The implementation of each sub-component consists of a JAVA file with its dynamic model, and a corresponding XML file with the default values of its parameters. One of the key parameters is the Unified Resource Identifier referring to the ontological entity (concept) that the sub-component represents.

The agent definition task is guided and supported by the behavioural ontology, mainly to guarantee the consistency of the new agent type, which is then saved (serialized) to the Agent Types repository for future use.

New simulation scenarios are created analogously: the user chooses one from the library of predefined physical environments and places in it the desired number of agents of appropriate types. The user can then modify the parameter values of each agent instance in the scenario, and save the new scenario to the Simulation Scenarios repository for future use.

D. *Agent-based Simulation Component (ABS)*

The advantage of de-coupling the definition of agent types and scenarios from the actual simulation is that in the next step, the Agent-based Simulation component (ABS) can be invoked just with one parameter: the ID of the scenario that it should execute. ABS will load the scenario definition as well as the other required resources, e.g. the agent types and the JAVA classes corresponding to their sub-components. This will facilitate the use of the ABS component especially in Data Farming.

The overall structure of the ABS component mimics the "PECS agent world" mentioned earlier and comprises the same sub-components: (1) a collection of agents; (2) Environment, which is responsible for a collection of environmental objects, and (3) Connector, which is responsible for messages and events, and their logging.

Their interactions are shown in Figure 5, which depicts one iteration cycle of the overall simulation process. In step 1, each agent reads from the connector the messages

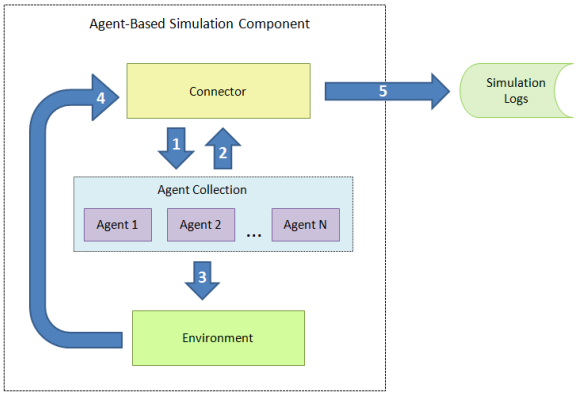

Figure 5. Structure of the Agent-based Simulation Component

and events addressed to itself, and updates its internal state accordingly. In step 2, each agent sends its significant internal changes to the connector for logging. Step 3 comprises each agent's active changes to the environment (e.g. by picking or throwing objects) as well as sending messages to other agents. The messages are routed through the environment because the communication is, in general, environmentally modified (for example, verbal messages may be missed or misheard due to surrounding noise). In step 4, the environment passes on the messages and the newly generated events to the targeted or impacted agents through the connector. In step 5, the connector automatically logs all the events and messages as well as the internal agent changes (passed on to it in step 2).

E. Generic PECS Agent

The PECS reference model gives the modellers the freedom to adjust the internal structure of the agents to suit their modelling needs. Figure 6 shows an initial high-level (conceptual) design of the agent that we plan to use in the EUSAS project. We believe it can accommodate new agent sub-components that are unknown at present, so long as they are compatible with the general PECS reference frame and conform to the software interfaces that we plan to define in the project.

The agent is conceived as an interlinked collection of agent sub-components. Each sub-component is implemented as a class (executable object) exposing its members and methods (typically motives and internal state variables) to other agent sub-components which, in turn, may use them as inputs. For the whole agent definition to be consistent, the inputs of each sub-component in its internal collection must be defined – either by linking them to the exposed members of the other sub-components, or by setting them to appropriate constant values.

At this stage of the design, we found it unnecessary to differentiate among the types of agent sub-components (i.e. whether they are related to Physis, Emotion, Cognition or Social status) and we have grouped them in one common collection, "PECS Motives and States". We have also separated cognitive motives and states from the Behavioural Cognition, whose task, as defined in Figure 6, is to choose one action plan from the set of plans generated by the behaviour patterns applicable in a given situation. The exact criteria used by the Behavioural Cognition will be defined later during the project.

The advantages of this architecture can be demonstrated on the process of the agent's internal update schematically depicted in Figure 6. The whole update procedure happens during one iteration cycle and most of it fits in between steps 1 and 3 shown earlier in Figure 5.

In step 1, the agent reads from the connector the events and messages addressed to itself. Sensory Input and Perception subsystem evaluates them and passes on the results to other subsystems. In step 2, the collection of PECS Motives and States receives the evaluated events, and each member of the collection updates itself by calculating new values of its internal motives and state variables. In step 3, Behavioural Cognition receives the inputs and stores them for later use (in step 8).

The key to agent behaviour is step 4, where the Behaviour subsystem receives and scans the inputs. If they contain a deactivating event (e.g. agent is killed or so severely injured that it falls unconscious), it puts the agent into an "inactive" state, and the agent must be explicitly re-activated to start responding to external events again. In the absence of deactivating events, the Behaviour subsystem searches for priority events, i.e. the external impacts, such as injuries, that forcibly terminate the agent's current action plan and impose a new agent state (often with decreased health level). We plan to implement the transition to the new state through a kind of internal action plan, so the processing would skip to step 6.

If there are no incoming deactivating or priority events, the agent behaviour is "normal". This means the Behaviour subsystem obtains the new action-guiding motive and health level from the collection of PECS Motives and States (step 5). If there is an action plan under execution whose motive is the same as the new action-guiding motive, the action plan is continued. If the new action-guiding motive is different (or if some elementary action of the current plan fails), the Behaviour subsystem terminates the current plan and asks for a new one in step 6. The request for a new action plan may also be triggered by a priority event from step 4.

In step 7, Behavioural Cognition invokes the applicable Behaviour Patterns and collects from them the executable action plans. In order to create the plans, the Behaviour Pattern objects may invoke other services and reasoners, e.g. A* algorithm for paths to specified target positions. These auxiliary calls were omitted from the diagram so as to avoid cluttering it. Similarly, Behavioural Cognition may also need to query other objects, e.g. the collection of

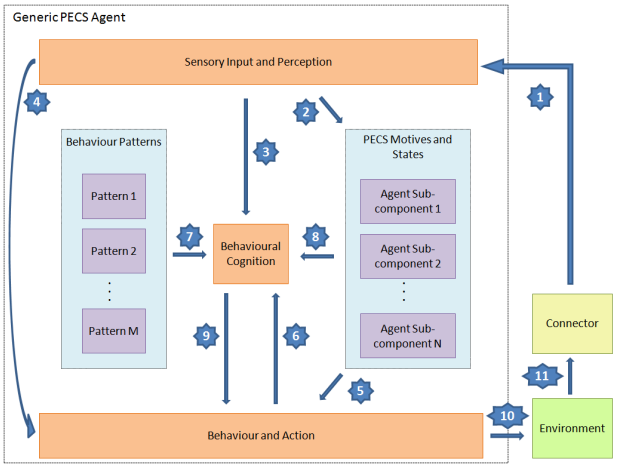

Figure 6. Generic Agent Architecture (Implementation View)

PECS Motives and States (step 8), so as to choose the "optimal" action plan and return it to the Behaviour subsystem for execution (step 9).

In step 10, all the elementary actions with external impact or environmental dependency (including messages to other agents) are passed on to the environment component for modification and execution. Their execution usually results in the generation of new events affecting other agents and objects. These are passed on to the `connector` component in step 11, where they are automatically logged. The affected agents will read them during the next iteration cycle.

F. Data Farming Considerations

The Agent-based simulation component will typically be used either for Data Farming or Virtual Training. In Data Farming, we plan to perform multiparametric studies, so we would run in parallel a number of instances, typically thousands, of the same base simulation scenario but with different values of selected parameters. For this purpose, we introduced an additional level of overriding the parameter values. The first level is the agent instantiation step in the base scenario definition, where each agent instance, after it is placed in the environment, can be given new parameter values overriding the default values inherited from its base agent type. The second level is the parameter value override for each of the parallel simulation instances in the Data Farming mode. We plan to use similar data structures for both levels, thus making it possible to fully parameterize the simulation scenario value space. The progress and the results of each parameterized simulation would be stored and later analyzed by data analysis tools.

G. Virtual Training Mode

In the EUSAS project, virtual training will be realized through the COTS (commercial off-the-shelf) battlefield simulation system VBS2 [16]. There are two principal alternatives for sharing common components between our agent-based simulation component (ABS) and VBS2:

(1) sharing agent models by generating VBS2 scripts;

(2) real-time synchronization between ABS (running in JAVA) and VBS2 (C++ code).

In general, the first option might lead to different behaviour of the models in VBS2 environment (compared to ABS), so it would be difficult to test and validate. Real-time synchronisation, with ABS controlling the agent behaviour and VBS2 simply executing and displaying the elementary actions, guarantees almost the same behaviour in Virtual Training as in Data Farming, which is crucial for validity of simulation results. However, there are a number of technical challenges that still require further investigation and experimentation.

III. CONCLUSION AND FUTURE WORK

We have presented a conceptual design of a semantic intermediation framework that combines object-oriented and ontology-based approaches in order to achieve real-time interworking of human behaviour models. In the process, we have shown how the PECS reference model could be customized for the modelling requirements of the specific scenarios in the context the EUSAS project.

The evaluation of the ontologies, although preliminary, showed the applicability of ontologies for describing static information about the simulation environment. This approach, while providing concise representation, did not impose significant performance footprint.

In future we will concentrate on the detailed design and implementation of the intermediation framework and ontologies, as well as interfaces needed for Data Farming and Virtual Training. Especially the last requires intensive investigation and experimentation.

ACKNOWLEDGMENT

This work is supported by projects A-0938-RT-GC EUSAS, RECLER ITMS: 26240220029, VEGA No. 2/0211/09 and VEGA 2/0184/10.

REFERENCES

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach, 2nd ed*. Prentice Hall, 2002.

[2] J. Ferber, *MultiAgent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, 1999.

[3] W. Brenner, R. Zarnekow and H. Wittig: *Intelligent Software Agents: Foundations and Applications*. Springer, 1998, ISBN 3-540-63411-8.

[4] L. de Silva, S. Sardina and L. Padgham, "First principles planning in BDI systems," in *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*. Vol. II, IFAAMS, 2009, pp. 1105-1112.

[5] H. Jiang, J. M. Vidal and M. N. Huhns, "EBDI: an architecture for emotional agents," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. ACM, 2007, Article No. 11.

[6] B. G. Silverman, J. Cornwell, K. O'Brien and M. Johns, "Human behavior models for agents in simulators and games: Part I – Enabling science with PMFserv," *Presence: Teleoperators and Virtual Environments*, vol. 15, pp. 139-162, April 2006.

[7] B. Schmidt, "Modelling of Human Behaviour: The PECS Reference Model," in *Proc.14th European Simulation Symposium*, A. Verbraeck, W. Krug, Eds. SCS Europe BVBA, 2002.

[8] M. Laclavík, M. Babík, Z. Balogh, E. Gatial and L. Hluchý, "Semantic knowledge model and architecture for agents in discrete environments," in *Frontiers in Artificial Intelligence and Applications*, vol. 141, *Proc. of ECAI 2006 Conference*, G.Brewka et al., Eds. IOS Press, 2006, pp. 727-728.

[9] M. Laclavík, Z. Balogh, M. Babík and L. Hluchý, "AgentOWL: Semantic knowledge model and agent architecture," *Computing and Informatics*, vol. 25, No. 5, pp. 419-439, 2006.

[10] A. Bežek, M. Gams, "From Basic Behavior to Strategic Patterns in Robotic Soccer Domain," *Informatica*, 2:461-468, 2005

[11] D. C. de Paiva, R. Vieira and S. R. Musse, "Ontology-based crowd simulation for normal life situations," in *Proceedings of the Computer Graphics International 2005.* IEEE Computer Society, Washington, DC, 2005, pp. 221-226

[12] S. Bandini, S. Manzoni and S. Redaelli, "Towards an Ontology for Crowds Description: A Proposal Based on Description Logic," in *Proceedings of the 8th international Conference on Cellular Automata for Research and Industry,* H. Umeo, S. Morishita, K. Nishinari, T. Komatsuzaki, and S. Bandini, Eds. LNCS. Springer-Verlag, Berlin, Heidelberg, 2008, pp. 538-541

[13] F. Y. Okuyama, R. Vieira, R. H. Bordini and A. C. da Rocha Costa, "An ontology for defining environments within multi-agent simulations," in *First Workshop on Ontologies and Metamodeling in Software and Data Engineering*, G. Guizzardi and C. R. G. de Farias, Eds. Florianópolis, Brazil, 16-20 October, 2006

[14] Jena project website, http://openjena.org/, Accessed on 1.12.2010

[15] Pellet website, http://clarkparsia.com/pellet/, Accessed 3.12.2010

[16] VBS2 website, http://www.bisimulations.com/, Access 3.12.2010