

Crisis response simulation combining discrete-event and agent-based modeling

Rafael A. Gonzalez

Delft University of Technology

r.a.gonzalez@tudelft.nl

ABSTRACT

This paper presents a crisis response simulation model architecture combining a discrete-event simulation (DES) environment for a crisis scenario with an agent-based model of the response organization. In multi-agent systems (MAS) as a computational organization, agents are modeled and implemented separately from the environmental model. We follow this perspective and submit an architecture in which the environment is modeled as a discrete-event simulation, and the crisis response agents are modeled as a multi-agent system. The simultaneous integration and separation of both models allows for independent modifications of the response organization and the scenario, resulting in a testbed that allows testing different organizations to respond to the same scenario or different emergencies for the same organization. It also provides a high-level architecture suggesting the way in which DES and MAS can be combined into a single simulation in a simple way.

Keywords

Multi-agent systems, discrete-event simulation, coordination, crisis response.

INTRODUCTION

In the domain of crisis response, the two main approaches to recreate crisis scenarios are simulations and drills (Massaguer, Balasubramanian, Mehrotra, & Venkatasubramanian, 2006). Both simulations and drills are used not only for research, but also for training and planning. Two commonly used simulation approaches are discrete-event simulation (DES) and agent-based simulation, which can be modeled as a multi-agent system (MAS). This paper presents an architecture in which both approaches are combined to allow for different combinations of crisis response organizations (MAS-based) against different crisis scenarios (DES-based).

Agent-based simulation is one of the most promising for crisis response, in part because of the ability to quickly refine agent-based models, which also implies the ability to quickly start with rough models (Robinson & Brown, 2005). Such extensibility allows for addition of new behaviors (Comfort, Ko, & Zagorecki, 2004) and new roles which can be integrated to and compared with old ones (Massaguer et al., 2006). Agent-based simulations, given their capacity to produce aggregate behavior, can be used to produce complex patterns of interactions which emerge for the analyst or researcher to observe (Comfort et al., 2004). However, MAS models typically focus on response agents or organizations, e.g. responders (Takahashi, 2007), but are not necessarily appropriate for modeling the crisis situation itself. Other types of models may be required for representing the objects, events and dynamics of the environment in which those agents interact. Discrete-event simulation is process-oriented and events are arranged discretely, which is in line to how crisis simulations (computer-based or not) are typically arranged. For example, a description of an emergency starts with an incident and, as the situation evolves or escalates, further events are added to the description in a historical fashion. DES is well-suited to modeling entities and relationships that are subject to these events or in turn affect the outcome of the events. For example, environmental entities such as vehicles or infrastructure can affect the dynamics of an evolving fire incident. These entities differ from the response agents in that they do not necessarily require a high degree of autonomy or complexity. Accordingly, we suggest combining the benefits of both approaches in an architecture that combines MAS and DES, integrating them but maintaining enough separation so as to preserve the qualities of each approach and enabling independent changes in both the crisis situation model (DES-based) and the response organization (MAS-based).

Previous research has already tested the plausibility of combining DES and agent-based simulation, although to our knowledge not in the domain of crisis response. In addition, most efforts are aimed at supporting the features of one

the approaches inside the other or vice versa, while in our case we keep each approach and tools separated but provide a common architecture for combining them.

It has been shown that classic problems of discrete-event simulation (e.g. the generic job shop) can be simulated as a multi-agent system (Zhou, Lee, & Nee, 2008). Other approaches aim at combining the process-oriented approach of DES and the autonomous characteristics of MAS by adding a simulator with artificial time mechanisms to study different design choices enacted by the agents (Janssen & Verbraeck, 2005). On the one hand, it is possible to bring DES concepts into MAS (Gianni, 2008). This option extends agent behaviors to support DES behaviors, such as event handling and notification. On the other hand, agent-based systems can also be built in a DES environment (Kádár, Pfeiffer, & Monostori, 2005). In this case, communication between the agents is added as an extension to the DES model, using specifically developed agent interaction protocols. Similarly, it is possible to use the DEVS (Discrete Event Systems Specification) formalism to emphasize state over events or processes, with the aim of providing compatibility with the automaton-based view of agents (Uhrmacher, 2001). This approach is aimed at merging discrete-event simulation with agents into a single testbed that is an extension of the DEVS formalism.

The architecture presented in this paper uses the process-oriented, discrete-event characteristics of DES in order to model the crisis environment, that is, the entities and events related to the emergency. First, the underlying DES simulator handles the events and the random number generation so that the same numbers can be generated for different simulation runs. Second, we use MAS as an adequate representation of crisis response agents (e.g. firemen) because we want to support autonomy and be able to model different coordination mechanisms using agent-based communication protocols. However, we also allow for the agents themselves to be represented inside the DES component of the architecture as entities: animated proxies which handle the physical interaction with the environment (after being prompted to do so by the “owner” agent). This separation is not strictly necessary, but allows the MAS and DES components of the architecture to evolve separately – this is where this approach differs from those mentioned earlier. Thus, it is possible to produce new crisis scenarios (dynamic descriptions of an incident) relying on the DES, while at the same time making it possible to add new agents or new agent behaviors to the MAS to alter the way in which the response is carried out. There is however, one caveat: in order to support the interaction between the DES and MAS components of the architecture, we chose to have a *Model* entity in the DES component as a container for all the environmental entities. This *Model* implements a DES model interface, as defined by the underlying DES simulation suite, while at the same time being an extension of an agent, as defined by the underlying MAS environment. This extension allows the response agents and the DES entities to exchange messages. The *Model* sends messages to the agents notifying them of environmental changes; the agents send messages to the *Model* to prompt events that produce changes in the entities (e.g. moving a proxy fireman entity to fight a fire entity). This solution, however, does not alter the independence of the crisis scenario and does not require any additional layers on top of the MAS and DES frameworks chosen.

The rest of this paper is structured as follows. In the next section we briefly present the crisis scenario chosen for our simulation as a starting point for describing the architecture, keeping in mind that other scenarios are equally amenable to be modeled with our architectural approach. Then we present the high-level architecture as a structural representation of the *packages* that contain the DES and MAS components of the architecture, using a UML-based representation. We then zoom in to the MAS and DES components of the architecture in a subsequent section. In our last section we provide some discussion and conclusions.

THE CRISIS SCENARIO

A fictitious scenario of an emergency will be used for experimenting with coordination issues of crisis response. By “crisis scenario” we understand the description of a particular crisis incident and how it is expected to evolve in time in order to be the basis for experiments. In this paper it serves as starting point for presenting the architecture. To contribute to external validity of the simulation, the scenario is taken from an existing description of the way in which an emergency response should be carried out in the context of the Dutch GRIP (or crisis response coordination procedure) levels. The choice is then to adopt a scenario provided in the Dutch Ministry of Interior Website for training purposes (<http://www.minbzk.nl/contents/pages/3383/opschaling1.swf>). The scenario starts with a crane doing work on a road in the jurisdiction of a given municipality. The incident starts when a truck, carrying flammable liquid, crashes onto the crane. This prompts the response of fire, police and ambulance services in what is initially a routine situation. Escalation of the incident occurs when the truck catches fire. The incident becomes larger than originally assessed, more response units are needed and a coordinated response is required from multiple disciplines which will setup a CoPI (Commando Place Incident) operational team. Further escalation does

occur but will not be included in this version of the simulation model. A diagram representing the emergency scenario at this point is shown in Figure 1.

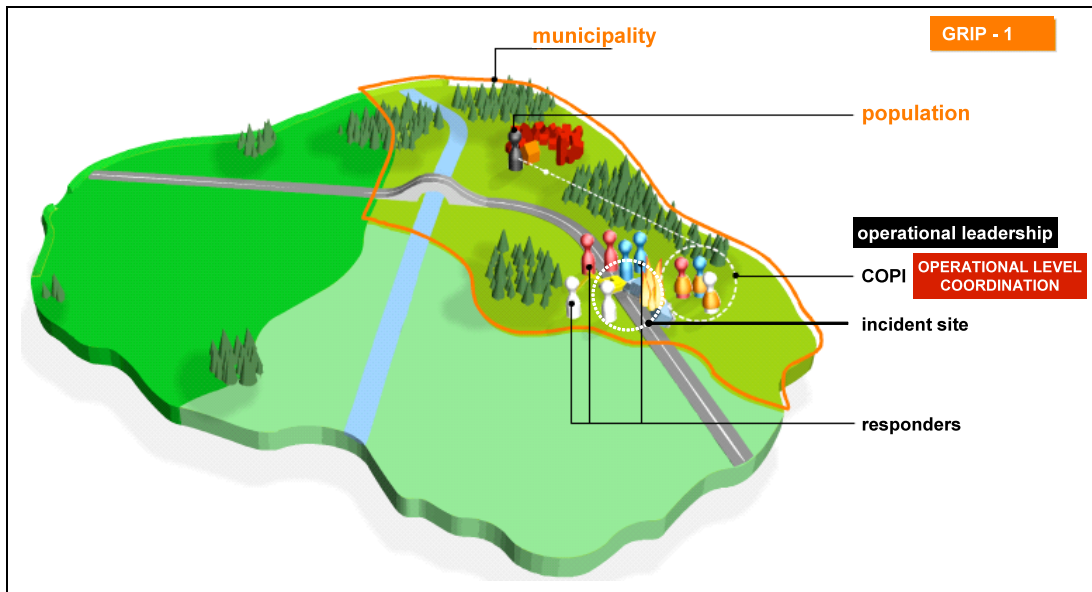


Figure 1. Crisis scenario, adapted from <http://www.minbzk.nl/contents/pages/3383/opschaling1.swf>

HIGH-LEVEL ARCHITECTURE

The architecture of the simulation model presented in this paper follows the view that a multi-agent system is a computational organization that interacts with an environment. In this case, such environment is defined and implemented separately from the MAS. The logic of the design of the architecture is dominated by a MAS development approach, because the focus of study is different configurations of the agents, their behavior and their interactions. The simulation itself, however, is led by the DES, because the time and events are deployed in the environment, which is modeled in the DES component of the architecture. Additionally, we use the DES-based environment also as the unique source of random numbers for the simulation, so that different MAS configurations can be tried for the same crisis scenario under the same conditions. The agents can interact with this DES environment, but do not require any additional simulation, since they exchange messages with the *Model* in the DES component of the architecture that acts as container for all environmental entities. The visual representation of this initial separation between the MAS organization and the underlying environment can be seen in Figure 2.

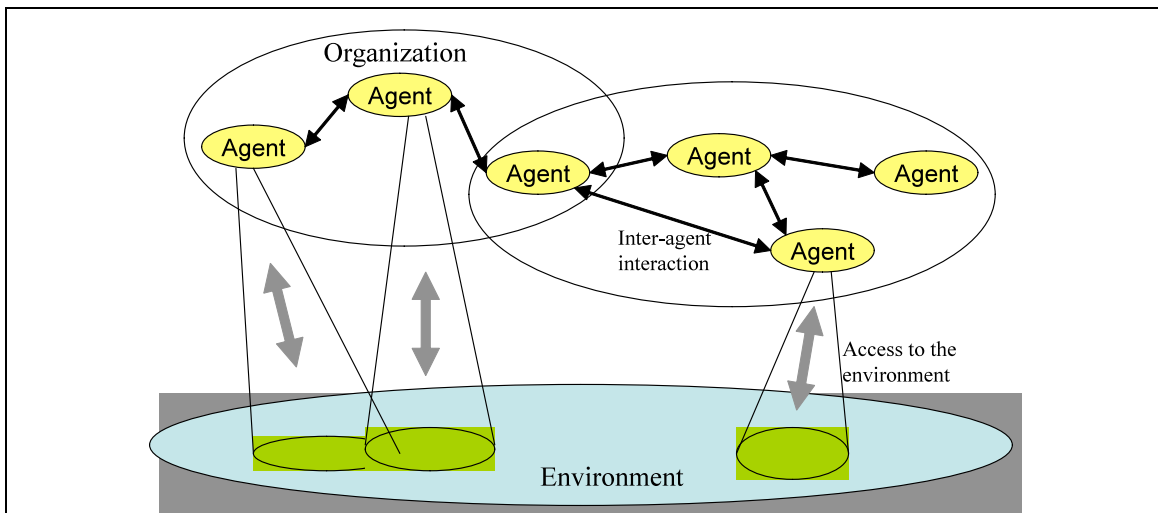
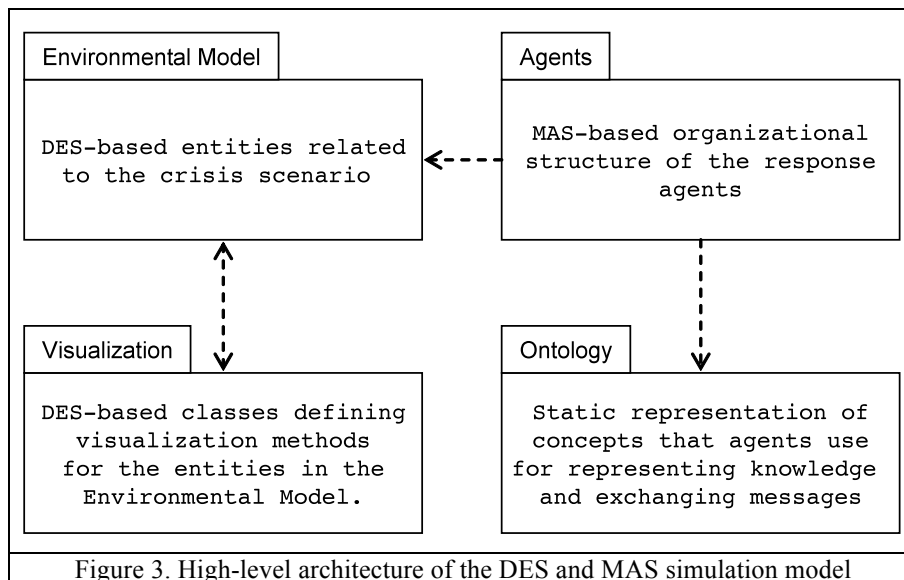


Figure 2. MAS computational organization, adapted from (Zambonelli, Jennings, & Wooldridge, 2003)

The high-level structural architecture of the simulation model is presented in this section as a set of interconnected UML packages (represented as folders) that contain the classes which implement DES entities (*Environmental Model*) and MAS agents (*Agents*) respectively, in addition to a *Visualization* and an *Ontology* package. The reason for the *Ontology* package is that although the architecture is implementation-independent, it does reflect the philosophy behind the FIPA agent specification (<http://www.fipa.org>), requiring an ontology to be defined for the multi-agent system. The *Visualization* package is also used for support, in this case for providing visualization functionality to the *Environmental Model*. All four packages can be seen in Figure 3.



The *Environmental Model* is the package containing the DES entities for the crisis scenario. In our case, the crisis scenario is the one described in the previous section, but again the idea is that this should be the container for any given crisis scenario that is to be simulated. Essentially, this package is where the discrete-event model resides. All environmental entities are modeled as classes (in the object-oriented sense), which can generate or react to discrete events and which are synchronized by the same discrete-event simulator. The *Environmental Model* contains the logic of the environmental entities, but interacts with a *Visualization* package that contains presentation methods for the simulation.

The *Visualization* package contains the classes that define the visualization behavior for the entities in the *Environmental Model*. This package is added to enable further flexibility in the architecture. While the *Environmental Model* contains the entities related to the crisis scenario (e.g. housing, vehicles, victims), the *Visualization* package defines abstract animated or static objects that define the methods for visualization and movement when running the simulation. This allows the *Environmental Model* to be changed or extended without being attached to particular visualization design decisions. For example, we have chosen visualization in simple 2D, but the same *Environmental Model* classes could extend 3D classes (when provided in the *Visualization* package) without having to do any major changes to the actual definition of the crisis scenario. Here, an important decision on the architecture must be noted. The response agents also need to be visualized in the simulation, but there is no direct connection between the *Agents* package and the *Visualization* package. The reason for this is that we chose to model response agents **both** as entities inside the *Environmental Model* **and** as agents in the *Agents* package. It follows that when a response agent (e.g. a fireman) exists as an entity inside the *Environmental Model* it can then extend its behavior by being associated with an animated object inside the *Visualization* package. At the same time, this responder entity will also be connected to its corresponding agent inside the *Agents* package, but the latter is independent of the underlying visualization.

The *Agents* package contains the organizational structure of the MAS of response agents. In our case, those related to the three main disciplines during a typical emergency, such as the one selected as scenario (fire, medical and police services). Again, the agents here are connected to the *Environmental Model* by having a “proxy” entity

representing the physical responder inside the environment. Although it is equally possible to extend the agents so that they, once instantiated, have an actual presence inside the *Environmental Model*, we chose to separate the MAS behavioral descriptions inside the *Agents* package, with the discrete-event responders inside the *Environmental Model*. For each agent inside the *Agents* package, there will be one responder inside the *Environmental Model*. In a way, this constitutes a separation of body and mind of the response agent that enables extensions of the behavior of the agent regardless of whether it will be deployed in a discrete-event simulation environment or not. Equally, the responder entity inside the *Environmental Model* exists independently of whether it is controlled by an agent or not. In our case, the responder entity only acts when prompted by the corresponding agent in the *Agents* package, but other non-agent mechanisms (i.e. the own methods of the responder entity) could also make it move or interact with the other entities in the environment. The response agents in the *Agents* package interact with the *Environmental Model* by reading (observing) the entities in an area around the physical representation of them and can only effect changes in the environment when their physical proxy is able to do so.

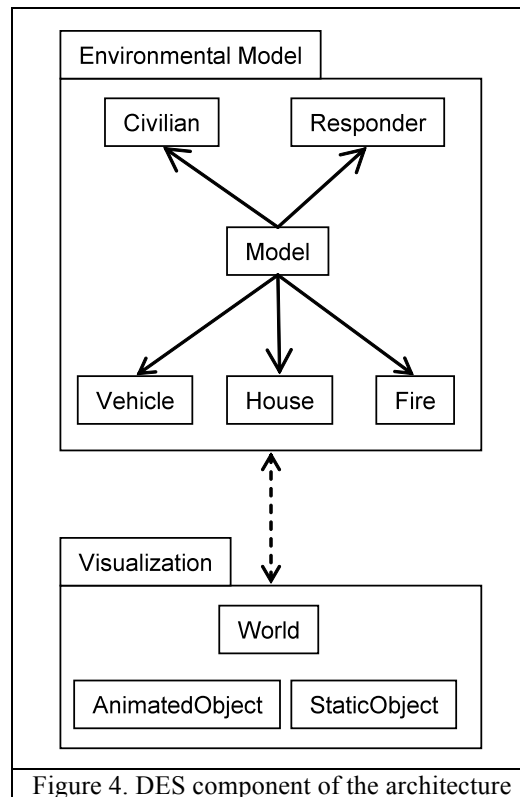
The *Ontology* package is provided so that the agents comply with the FIPA specifications and can use communicative acts that use the ontology as the language for exchanging, coding and decoding messages. The ontology can also be seen as a static (and empty) representation of the knowledge of the agents, i.e. their mental model. Hence the arrow connecting the *Agents* package and the *Ontology* package flows from the agents to the ontology, which remains static. The *Ontology* package contains the ontology objects that should map to the *Environmental Model* entities, following an agent's observation. For example, there will be an ontology object for the fire as well as a fire entity in the *Environmental Model*. However, in the *Environmental Model* the fire will have a state corresponding to the true state of the fire (including for example its size and growth rate), while the fire ontology object, once instantiated inside an agent, will contain information on the fire as observed by the agent. This means that there could be additional information inside the ontological fire, such as assigning a risk value; but there can also be inconsistency between the ontological fire object that the agent *knows* and the real fire entity that the agent *observes* (for example, a misperception of size). In any case, the ontology object of fire inside the *Ontology* package does not change at all, because it is a static description of a concept and not an instantiated piece of knowledge. The *Ontology* package is also useful for coding the communication between the agents, allowing for analysis of coordination, which is the purpose of our simulation. Communication is modeled as interaction patterns between the agents, defining the message templates as composition of ontological objects (although this is outside the scope of the present paper).

DES AND MAS COMPONENTS OF THE ARCHITECTURE

For the DES we have chosen to use the D-SOL discrete-event JAVA-based simulation suite (Jacobs, Lang, & Verbraeck, 2002). For the MAS we have chosen JADE as the JAVA-based framework (Bellifemine, Caire, Trucco, & Rimassa, 2008). The facts that the DES and MAS frameworks are both JAVA-based allows for an almost seamless integration of the two without having to alter or extend the original libraries or indeed for creating additional integration layers. This does not mean that the architecture is implementation-dependent at this stage, but suggests that using compatible MAS and DES frameworks contributes to its implementation. In this section we “zoom in” on the packages presented on the previous section presenting the class diagrams in each package and how they are related to the underlying MAS and DES frameworks.

DES-based packages

As noted before, the *Environmental Model* is designed as a DES simulation model, where entities are subject to events and dependent on the same time-advancement mechanism. There are entities related to the emergency situation (including physical representations of the response agents); and there are generic classes defining the behavior of visualization aspects. Both are contained in the packages shown in Figure 4.

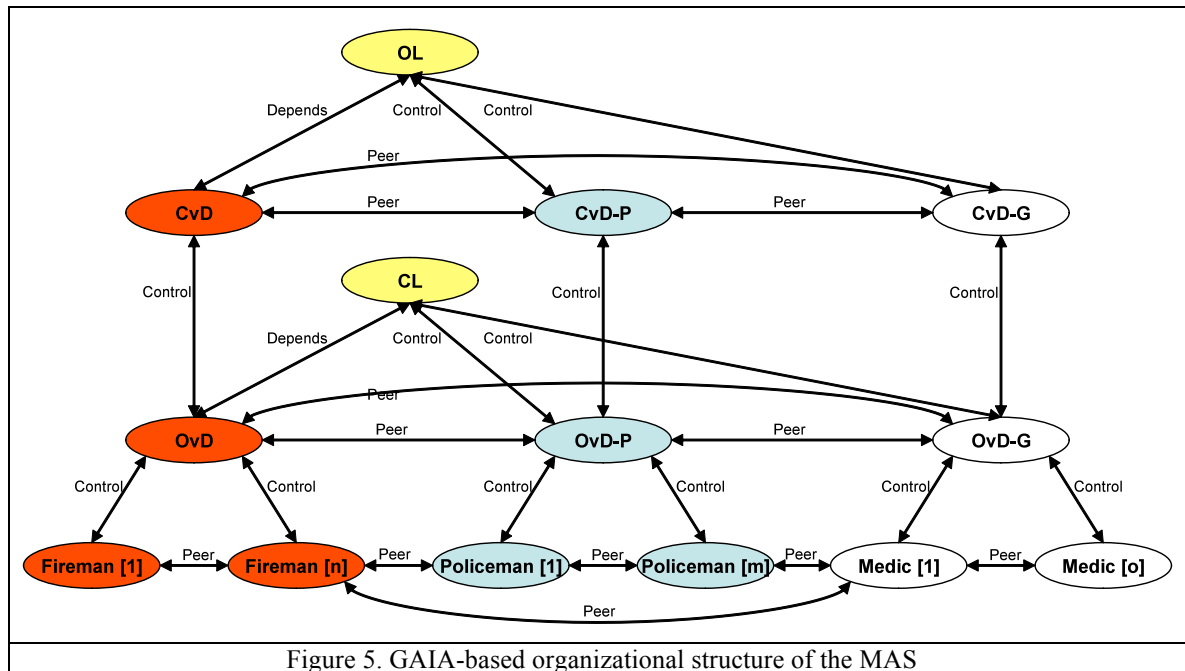


The *Environmental Model* package contains a *Model* class where all the entities are instantiated. This *Model* class implements a *DES model interface* as defined by the DSOL simulation suite. This basically provides the means for the model to be constructed with a reference to a DES simulator that is instantiated at the application level – the application itself can be managed through a utility class as in our case (not shown in the architecture) or using the interface provided by DSOL. Once the *Model* is constructed, all environmental entities are instantiated in its context with a reference to the same simulator. In this case, those entities are implemented as the following classes: *Civilian*, *Responder* (physical proxy for the response agent), *Vehicle*, *House* and *Fire*. These classes can be instantiated or deleted from the container *Model* as the crisis evolves in run-time, but can also be added or modified in design-time to define a different emergency scenario. Given that we need the response agents in the *Agents* package to communicate with the *Environmental Model*, we have the *Model* class extend a JADE agent. Effectively this means that the *Model* class is both a DES model and an agent. However, this has no influence on the DES aspects of the *Model* itself, since the agent-based behaviors are defined separately and will not be active until an agent container (in this case provided by a JADE plug-in for Eclipse, called EJADE: <http://dit.unitn.it/~dnguyen/ejade/>) has been deployed. Without an agent container and an agent setup, the *Model* simply behaves as a DES simulation model. Because the *Model* is not really an agent inside the *Agents* package – in the sense that it exhibits no autonomous behavior – it only implements the standard *setup* and *take down* methods of a JADE agent. This allows messages to be exchanged between the *Model* and the agents. The *Model* then acts as a centralized gateway between the agents and the entities of the crisis environment without having to change either the implementation of the *Environmental Model* or the implementation of the *Agents*. The fact that both JADE and DSOL are JAVA-based and that the *Model* class implements a DSOL model and extends a JADE agent, allows for this simultaneous integration and separation of both components of the architecture.

MAS-based packages

The MAS component of the architecture is built in line with the GAIA methodology (Zambonelli et al., 2003). In this approach to analysis and design of multi-agent systems, the architecture is equivalent to the organizational structure of the system, itself a combination of the topology and control regime of the agent organization. The topology in this case follows the basic organizational response structure as defined by the Coordinated Response Procedure of The Netherlands for the Rotterdam-Rijnmond Region in which we base our studies (Trijselaar, 2006).

An initial diagram of the topology including the control regime (peer, dependence and control relationships) is shown in Figure 5.



At the bottom level are the first responders of the three basic disciplines: firemen, policemen and medics. The operational leadership of these levels is represented respectively by the OvD (Fire Chief Officer), the OvD-P (Police Chief Officer) and the OvD-G (Medical Chief Officer). When an emergency reaches the point in which multidisciplinary response is required, these disciplines must act together, requiring the setup of a multidisciplinary response team called CoPI Team, which requires a leader: the CL (CoPI Leader). In case the emergency escalates to the regional level, the commanders of each of the services need to be involved, establishing a level of control and coordination at the regional level, on top of the already existing CoPI. The commanders from the different disciplines are shown in the figure as CvD (Fire Regional Commander), CvD-P (Police Regional Commander) and CvD-G (Medical Regional Commander). A regional multidisciplinary response team for coordination needs to be setup in an equivalent but higher level than the CoPI: the RegOT (or Regional Operational Team), which is led by the OL (Operational Leader) shown as the uppermost agent in the organizational structure.

Each agent is defined as a Finite State Machine (FSM) by extending the FSMBehavior provided by JADE. This follows the GAIA2JADE (Moraitis & Spanoudakis, 2006) process which allows us to continue the analysis and design based on GAIA into an implementation-dependent representation of the agents in JADE. Each of the states defined for an agent is obtained from standards crisis response procedures and modeled in UML as state transition diagrams (outside the scope of this paper) that are subsequently implemented using the FSM constructs of JADE (FSMBehaviour, FSMChildBehaviour, and transitions).

Having defined the MAS organizational structure, we follow the design by defining a FIPA compliant ontology for the agents to use for exchanging messages. The domain ontology in JADE describes the elements that agent use to create the content of messages, specifically concepts, predicates and actions (Bellifemine et al., 2008). Each of these three elements must be represented through a schema and a JADE class implemented for each schema (from JADE interfaces). The empty description of the ontology is defined in the *Ontology* package. This package, along with the *Agents* package that defines the agent organizational structure in an object-oriented fashion can be seen in Figure 6.

The organizational structure of the MAS can be simplified by defining more than one role for certain agents. In this case, the CL will by default be the Fire Chief and the OL will be the Fire Regional Commander. In addition, because our study focuses on an emergency that does not scale up to the regional level, we need only consider the CoPI level. Accordingly, the *Agents* package shows the MAS-based component of the architecture where the agent classes have been limited according to the above design decisions.

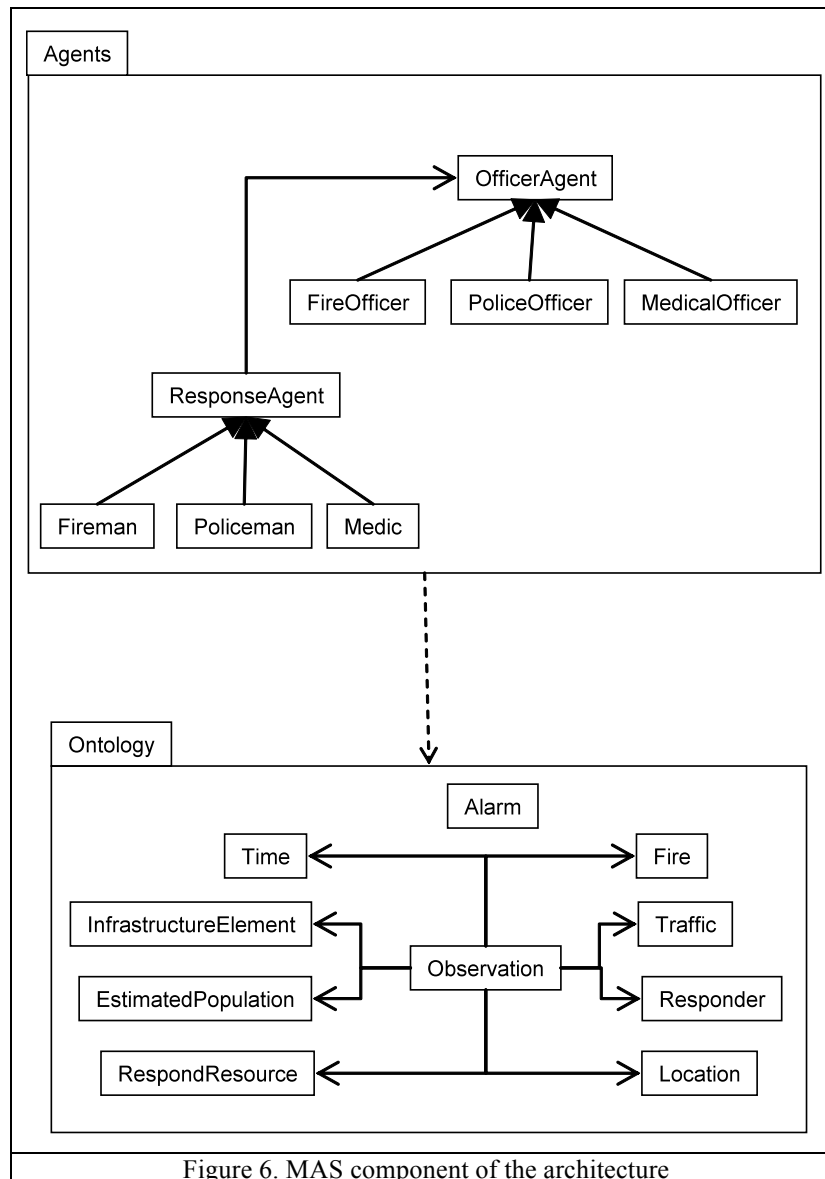


Figure 6. MAS component of the architecture

The Concepts inside the *Ontology* package are the semantic elements of the vocabulary that the agents use and for this scenario have been defined as: *Estimated Population*, *Fire*, *Infrastructure Element* (such as a house), *Location* (generic concept used for describing the location of entities in the scenario), *Material Response Resource* (such as fire truck or traffic control elements), *Responder* (the physical representation of the response agents present in the environment), and *Traffic*. Predicates are the structural elements that link concepts together. In our case, we use *Observation* as the predicate that asserts a given observation about the environment by a given agent at a given time. This *Observation* is used as the main container for the mental model of each agent. Actions are special concepts that denote agent actions. In our case *Alarm* is used to request an action from the agents (getting to the incident location).

DISCUSSION AND CONCLUSION

The architecture presented above is being used to experiment with coordination mechanisms of crisis response organizations. On the one hand, the crisis scenario has been defined using discrete-event simulation and runs independently of the agent-based model. On the other hand, the response agents are defined in the MAS and are only deployed after receiving an alarm from a dispatch agent (which is not shown in the architecture for simplification). Changes in the DES and MAS models can be made without affecting each other, which for our experiments allows

changing the coordination between the agents as defined through conversations (implemented as FIPA interaction protocols). In general terms, this strategy can be used to try out different configurations in the agent organization, to add new behaviors in the individual agents and to test the same configurations and behaviors in different crisis scenarios, as long as the agents have a physical representation they can control and also the ability to exchange messages with the DES model.

The relationship between the ontology and the DES environmental model can be seen clearly in the architecture and suggests a mapping between the two, although not on a one to one basis necessarily, since agents can reason about the environment through additional concepts. This soft coupling between the DES environment and the ontology maintains separation of MAS and DES but does require a link at the architectural level between the two. In our case, however, such a link is used for experimental purposes because it allows us to analyze the inconsistencies between the different mental models of the agents and also between these and the actual state of the environment. Such inconsistencies are used to study the coordination required to come to shared situational awareness, which is represented by merging the instantiated observations of several agents. In addition, using an ontology in the FIPA spirit, permits modeling agent interactions that follow FIPA interaction protocols directly allowing for a simple definition of different coordination mechanisms as implemented by different interaction protocols (such as Request-Reply or Contract Net). New coordination mechanisms can then be defined in the same style, resulting in the definition of new interaction protocols that can be defined in this standard way not only to reflect coordination mechanisms of real crisis organizations but also to be applied in MAS that support crisis response organizations.

REFERENCES

1. Bellifemine, F., Caire, G., Trucco, T., & Rimassa, G. (2008, August 18, 2008). Jade Programmer's Guide: JADE 3.6. from <http://jade.tilab.com/doc/programmersguide.pdf>
2. Comfort, L. K., Ko, K., & Zagorecki, A. (2004). Coordination in rapidly evolving disaster response systems: The role of information. *American Behavioral Scientist*, 48(3), 295-313.
3. Gianni, D. (2008). *Bringing Discrete Event Simulation Concepts into Multi-agent Systems*. Paper presented at the Tenth International Conference on Computer Modeling and Simulation, 2008. UKSIM 2008. .
4. Jacobs, P. H. M., Lang, N. A., & Verbraeck, A. (2002). *D-SOL: A Distributed Java Based Discrete Event Simulation Architecture*. Paper presented at the 2002 Winter Simulation Conference.
5. Janssen, M., & Verbraeck, A. (2005). An agent-based simulation testbed for evaluating internet-based matching mechanisms. *Simulation Modelling Practice and Theory*, 13(5), 371-388.
6. Kádár, B., Pfeiffer, A., & Monostori, L. (2005). Building Agent-Based Systems in a Discrete-Event Simulation Environment. In M. Pechoucek, P. Petta & L. Z. Varga (Eds.), *Multi-Agent Systems and Applications IV* (pp. 595-599). Berlin: Springer.
7. Massaguer, D., Balasubramanian, V., Mehrotra, S., & Venkatasubramanian, N. (2006). *Multi-Agent Simulation of Disaster Response*. Paper presented at the First International AAMAS Workshop on Agent Technology for Disaster Management.
8. Moraitis, P., & Spanoudakis, N. (2006). The GAIA2JADE Process for Multi-Agent Systems Development. *Applied Artificial Intelligence*, 20(2), 251 - 273.
9. Robinson, C. D., & Brown, D. E. (2005). *First responder information flow simulation: a tool for technology assessment*. Paper presented at the 37th Conference on Winter Simulation.
10. Takahashi, T. (2007). *Agent-Based Disaster Simulation Evaluation and its Probability Model Interpretation*. Paper presented at the 4th International Conference on Information Systems for Crisis Response (ISCRAM2007).
11. Trijselaar, J. (2006). *Gecoördineerde Regionale Incidentenbestrijdingsprocedure Rotterdam-Rijnmond. Versie 4.2*. Retrieved Apr. 8, 2008. from <http://www.rhrr.nl/eCache/DEF/1/956.pdf>.
12. Uhrmacher, A. M. (2001). A System Theoretic Approach to Constructing Test Beds for Multi-Agent Systems In H. S. Sarjoughian & F. E. Cellier (Eds.), *Discrete Event Modeling and Simulation Technologies: A tapestry of systems and AI-based theories and methodologies* (pp. 315-333). New York: Springer.
13. Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2003). Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3), 317-370.
14. Zhou, R., Lee, H. P., & Nee, A. Y. C. (2008). Simulating the generic job shop as a multi-agent system. *International Journal of Intelligent Systems Technologies and Applications*, 4(1/2), 5-33.