EnvSupport: A Framework for Developing Virtual Environments

Kai Jander, Lars Braubach, Alexander Pokahr
University of Hamburg
Distributed Systems and Information Systems
{jander,braubach,pokahr}@informatik.uni-hamburg.de

Abstract

Multi-agent virtual environments combine the flexibility of agents with rich areas of practical application such as simulation experiments and gaming. However, the development of virtual environments remains an endeavor of considerable effort. In addition, while a number of frameworks offer considerable reductions in development effort, it comes at the cost of lower flexibility in areas such as choice of agent architectures, types of agent interaction and collaboration models. This paper demonstrates an application framework called "EnvSupport" for reducing development effort for multi-agent virtual environments. It offers agent-environment interaction, configurable graphical output, statistical monitoring features and dynamic environment capabilities for agents with application in simulation and beyond.

1 Introduction

Agents acting in virtual environments represent an important building block of many interesting classes of agent applications, such as AI testbeds, simulation experiments, or computer games and movies [Weiss *et al.*, 2010]. Examples range from simple grid environments such as the well-known Wumpus world from [Russell and Norvig, 1995] over more complex 2d continuous spaces like e.g. Cleaner world [Braubach *et al.*, 2005] to very complex and reality like 3d game environments in the spirit of Unreal Tournament.

Building agent applications of this type requires huge efforts on the agent as well as on the environment side. To simplify the creation of agent applications that include an environment, the developer has the option of reusing an existing environment or using some kind of rapid prototyping tool for developing a new one. Although sometimes environments are developed with reusability in mind (e.g. some computer games offer proprietary programming interfaces), up to now no general and systematic means for reusing environments are available. Only recently the idea of a generic environment interface standard [Behrens *et al.*, 2009] has been introduced, which aims at easy interfacing of existing environments with different agent platforms. The second option, rapid prototyping tools for agent environments, is commonly found in agent simulation toolkits. Support environment-based agent application design in a similar fashion is the primary inspiration for the approach presented in this paper.

The basic idea is to provide a specialized framework for quickly developing spatial agent environments. The proposed framework "EnvSupport" allows for describing an environment in a declarative manner. It includes support for the environment domain model, the agent-environment interaction, the graphical presentation and also the evaluation of environmental data.

In contrast to specialized agent simulation toolkits, which also do a good job on rapid prototyping simulation models, our approach is intentionally broader and decouples the agent and environment aspects of development. Due to this separation, the non-environment parts of an application can be developed as in any other (non-environment-based) application. This approach has several advantages. Sophisticated agent architectures (e.g. BDI and task-based agents) as implemented in existing general purpose agent platforms can be used in simulation experiments as well. Simulated environments can be attached to real agent applications allowing to benchmark these under different conditions and using the virtual environment as an integral application part (e.g. as a coordination layer in a logistics application).

The remainder of the text is structured as follows. In the next Section the main concepts of EnvSupport are explained and in Section 3 its usage is illustrated by an example application. In Section 4 the approach is related to existing work and finally in Section 5 a conclusion and outlook on possible future enhancements is given.

2 EnvSupport Concepts

EnvSupport ("Environment Support") is an integrated support component of Jadex [Pokahr and Braubach, 2009a]. The goal of EnvSupport is facilitating the development of applications using agent environments, for example spatial multi-agent simulation applications. EnvSupports builds upon the applica-

tion descriptor and space concept support described in [Pokahr and Braubach, 2009b]. The application descriptor is an XML file which specifies the structure of an application like agent types and its runtime configuration. Environments can be declared in the application descriptor as *spaces*. Spaces are considered to be a generalized concept of environments and can represent any conceivable environment or agent relationship. The concept of a space is intentionally abstract, allowing concrete application to define the space concept required for their specific application.

This extension point is used by EnvSupport to supply a specific space implementation including concepts, which are geared towards the application domain of spatial environments. This means that applications using EnvSupport can use the application descriptor to quickly configure the application by configuring an EnvSupport-provided space and declaring the agent involved in the application. In addition, extended concepts which are useful for environments such as a graphical observer tool and an evaluation tool for data processing, can be declared and configured using the application descriptor.

EnvSupport provides a number of important default infrastructure components which are already sufficient for a large class of environment applications by configuring them to suit the requirements of the application. In addition, almost every aspect of EnvSupport is optional and extensible if the application's requirements are not met by the default components.

2.1 Environment

EnvSupport currently supports two specialized spaces for environment development which both represent a configurable, two-dimensional area. A two-dimensional grid space provides the features used in classical grid-based environments where the environment area is divided into rectangles of equal area. The second environment provided by EnvSupport is a continuous space which does not explicitly quantize the environment area but instead represents an approximation of a continuous space environment using floating-point arithmetic.

Each environment consists of a space which contains components providing the functionality of the environment (cf. Figure 1). Central components are the *space objects*, which represent entities existing in the environment. Space objects are typed structs which can be nested arbitrarily and include customizable properties.

Space objects can be ownerless or they can be owned by one or more agents. A space object which is owned and controlled by an agent is considered to be an *avatar* of the agent. Avatar relationships are defined by associating a specific kind of agent with a specific space object type using the *agent/avatar mapping*. This mapping allows also defining the characteristics of the bi-directional relationship between both via specific boolean flags (create_avatar, create_agent, kill_avatar, kill_agent). It is thus possible to tie the creation and termination of each association end together as needed. This means that if a new agent is added (and create_avatar is true), a space object of the associated type is automatically created and ownership of the object is transferred to the new agent. The same applies for the removal of an agent or avatar.

In order to express dynamic behavior of the environment, EnvSupport provides for two different dynamic components, *processes* and *tasks*. Processes are used to represent global processes such as heat dissipation and generation of space objects. They have full access to all components within the environment, including all environment objects.

As a finer-grained approach to dynamic behavior, EnvSupport offers a second dynamic component called tasks. Tasks are part of a specific space object and are restricted in their access of environment state to that particular space object. Tasks can modify the properties of the space object and provide for simple dynamic behavior such as updating the position of the object according to a velocity vector or decreasing the battery charge of the object.

All of the dynamic objects need to be executed and the correct time for execution needs to be controlled. This is accompished by the *space executor* which represents the active component of the environment.

EnvSupport provides two ready-to-use implementations of space executors. The first implementation is a time-based executor which runs the dynamic components of the environment whenever the environment clock updates. This executor is particularly useful when implementing applications that are based on an approximately continuous view of time. The second space executor is a round-based executor which will perform rounds at regular intervals. This executor is suited for environments that involve round-based rules.

The use of a space executor is not mandatory. If the environment is not supposed to be dynamic and solely influenced by agents, the space executor can be omitted. However, omitting the space executor means that no processes or tasks are executed and therefore do not provide any function in the environment.

In order to interact with the environment, the agents need both sensors and effectors. In EnvSupport, sensors are represented by the percept system. An EnvSupport environment has a number of predefined *percepts* that can occur during execution. Percepts are triggered by an internal event system with the environment. Dynamic components can create an internal event, which prompts the *percept generator* to instantiate a percept. The percept is passed on to agent-specific *percept processors*. Percept processors are able to pass the information to an agent either by directly modifying the agent's state or by using non-invasive means such as message passing.

Agent effectors are represented by *agent actions*. Agent actions, like percepts, are pre-defined in the environment. They can be invoked by the agents in order to modify the environment state. EnvSupport provides two kinds of agent actions. Immediate agent actions immediately and directly modify the environment state. This type of agent action can be used if the agents are allowed to affect the environment at

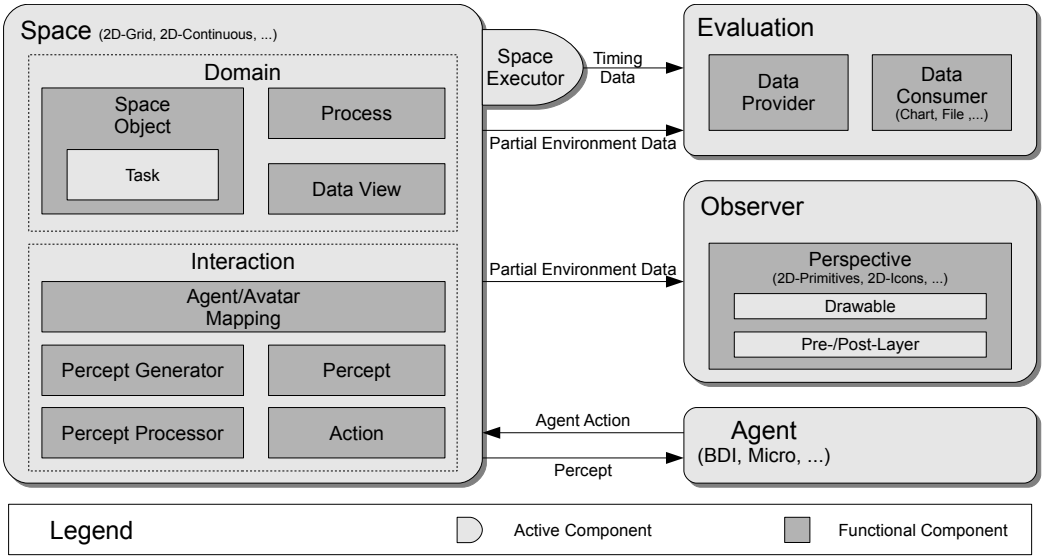

Figure 1: The EnvSupport Architecture

any time, for example, if the application is solely time-based and does not have a concepts of rounds.

The second type of agent action is the ordered agent action. Unlike the immediate agent action, it is not performed immediately but merely added to a queue. This queue is then processed the next time the space executor is invoked. The space executor can reorder the actions and enforce limits on the number of actions per agent. This approach is particularly useful for applications involving rounds, where an agent may only be allowed a limited number of actions or there is a specific order in which they are to be performed.

Agent Actions can also have return values, providing an additional way for the agent to acquire data from the environment. This path of data acquisition differs from percepts since the data request is initiated by the agent, while percepts are initiated by a dynamic component of the environment.

2.2 Observer

The observer allows inspection of the environment while the application is running and allows graphical output of the current environment state. However, displaying the full environment state is not always desirable, for example, if the intent is to display only the known world state of a single agent, the environment data has to be restricted.

This is accomplished using *data views*. Data views are part of the environment and act as a filter between the environment and the observer. The observer uses data views to acquire the environment state that is supposed to be displayed, while the data view only passes data about objects that are relevant to the current view. The exact criteria which specify the relevance of data in a particular view has to be implemented by the application developer. If a global view of the environment is sufficient, an EnvSupport-provided data view that passes all data of the environment to the observer can be used.

The representation of space objects in the observer is specified for each space object type, such that space objects of the same type have the same graphical representation. The specification of the representation is accomplished through the use of a simplified, single-level scene graph. Scene graphs are graphs or trees where the parent nodes influence the appearance of their child nodes [Rohlf and Helman, 1994]. EnvSupport uses a simplified approach, which specifies a container of primitives each space object type. This container is refered to as *drawable* and has a number of attributes associated with it, which define their general appearance and position, such as scale in both dimensions, rotation with regard to three axes, and position.

In order to specify the exact appearance, the application developer has a choice of a number of graphical primitives, such as colored triangles, rectangles, circles, regular polygons, text outputs and texturized rectangles which can be added to drawables. Each primitive has the same attributes as a drawable and includes some additional attributes specific to it, such as color, image of the texture and the number of vertices in the case of regular polygons. Attributes which primitives share with drawables are interpreted like a scene graph to be relative to the specification in the drawable.

Both the attributes of primitives and drawables can either be specified as a fixed value or they can be bound to properties of the space object they are representing. The bound value can also refer to an expression that performs a calculation on space object properties before returning the resulting value to the observer by introducing a local property of the drawable which defines the expression. Using property bindings, the representation can dynamically accomodate environment state, which is critical in cases of object position but is also useful for implementing graphical features, such as the representation of objects growing

in size if a space object property increases.

Each of the primitives can have a layer associated with them. If no layer is specified, the default layer 0 is chosen. The layer specifies when an object is drawn with lower values indicating that the primitive is drawn earlier than primitives with a higher layer value indicating which primitives are allowed to obscure others.

In addition to drawables for space objects, *pre-layers* and *post-layers* are provided by the observer. Pre-layers and post-layers are environment area-wide graphical features that are drawn before or after the space objects are drawn. Therefore pre-layers can be used to draw backgrounds, such as background images or grids, while post-layers can be used to draw foreground features such as a cloud layer.

A set including drawables for the space object types and a number of pre- and post-layers is called a *perspective*, defining a certain graphical appearance of the environment. The application developer can specify multiple perspectives for the observer, which allows chosing between them while the application is running. Taken together with multiple data views, this gives a wide choice of graphical appearances and restricted views on the environment, which can be used in arbitrary combination while using the observer.

2.3 Evaluation

The evaluation facilities of EnvSupport allow performing statistical evaluation on user defined environment data. Conceptually, the evaluation is subdivided into *data providers* and *data consumers*. Data providers have the purpose to collect data from the space, whereas data consumers can be used to process the collected data in arbitrary ways.

Data providers allow an application developer to specify in a declarative way the interesting properties of the environment that should be gathered during runtime. A data provider is responsible for executing a query on demand and deliver the results of this query (it does not store the values of earlier queries). In order to simplify the specifcation a table based format is used, whereby each data provider owns one table. The query is formulated using data sources (typically space objects) and column properties. As in relational data base queries a join operation is performed against all data sources so that the result set consists of the cartesian product of values from the sources.

Data consumers employ data providers for fetching their data. A data consumer is activated via the space executor and then uses its data consumer for getting the current data table. It then can process that data further, e.g. present it visually as chart or histogram or store it in a csv file. The XML specification of data consumers depends heavily on their concrete type and is supported in a generic way via properties.

The conceptual differentiation between data providers and consumers has several benefits. Primarily, it introduces a separation of concerns between collecting and processing data, which reduces the complexity of the components. This facilitates usability and fosters future extensions of evaluation with new components.

3 Example Application

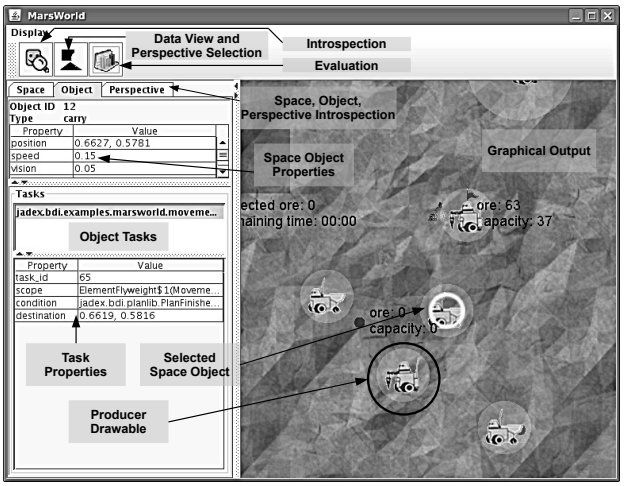

Figure 2: MarsWorld application in observer window

Jadex includes several example applications which use EnvSupport. The "MarsWorld" application (cf. Figure 2) involves the automated search for and mining of ore on mars. This is accomplished by three different robots. The job of the sentry robot is to search for new mining locations and make them known to the other robots. The producer robot proceeds to locations discovered by the sentry and mine the ore until the location is exhausted. Lastly, the carrier robot will transport the ore mined by the producer to the homebase, where it is stored.

The MarsWorld application uses the application descriptor to define the application. The application descriptor is split in sections which describe the type or structure of an application components such as an agent or space and a section declaring an application instance. The instance defines which components are involved in the application and the number of instances required. Since the MarsWorld environment is supposed to be continuous, the two-dimensional, continuous space provided by EnvSupport is chosen to represent the environment. In addition, the environment requires a number of space object types, some of which represent avatars of agents, others, like the ore, are ownerless. The space type and its space object types with their properties are defined in the application descriptor (cf. Figure 3).

3.1 Domain Specification

The space used in MarsWorld is defined to have a 1-by-1 area. For each of the space object properties, an initial value is defined. In addition to immediate values, expressions can be added used in properties wich will be evaluated dynamically during runtime.

MarsWorld does not contain any global processes but a few tasks are used to help the agents control

```
<env:envspacetype name="2dspace" class="ContinuousSpace2D"
  width="1" height="1">
  <env:objecttypes>
    <env:objecttype name="target">
      <env:property name="ore">0</env:property>
      <env:property name="capacity">0</env:property>
      <env:property name="state">"unknown"</env:property>
    </env:objecttype>
    <env:objecttype name="sentry">
      <env:property name="vision">0.1</env:property>
      <env:property name="speed">0.05</env:property>
      <env:property name="position" dynamic="true">
        $space.getSpaceObjectsByType("homebase")[0]
          .getProperty("position")
      </env:property>
    </env:objecttype>
    [...]
  </env:objecttypes>
  [...]
</env:envspacetype>
```

Figure 3: Space type definition in application descriptor

their avatars. The first task used is the "move" task. This task controls the movement of the avatar by setting the new position of the avatar after each interval. The agent specifies a target location for the avatar and initializes the task using an action. The task calculates the correct velocity vector using the avatar's current position, the target's position and the maximum speed of the avatar. Since the task is executed by the space executor after each clock update, it can correctly calculate the new position of the avatar by multiplying this vector with the time interval that has passed. After the target has been reached, the task terminates. This allows the agent to direct its avatar by issuing a new target location and waiting for the move task to finish. In addition, three specialized tasks for each robot type are declared. The "analyze" task is used by the sentry to explore a new source of ore, the producer applies the "produce" task when producing the ore and the carrier robot can use the "load" task to load ore for transport.

3.2 Observer Specification

The perspective section describes the perspectives which are used by the observer. For each of the space object types a graphical representation needs to be defined. This is accomplished by declaring a drawable and defining the primitives of that drawable (cf. Figure 4).

```
<env:drawable objecttype="sentry"
  width="0.07" height="0.07">
  <env:property name="visionsize" dynamic="true">
    new Vector2Double($object.
      getProperty("vision").doubleValue()*2)
  </env:property>
    <env:ellipse layer="1" size="visionsize"
      abssize="true" color="#FAFA1E32" />
    <env:texturedrectangle layer="2"
      imagepath="jadex/.../sentry.png" />
</env:drawable>
```

Figure 4: Drawable definition of the sentry robot

The description declares a drawable defines the object's general appearance such as overall size. The drawables contains two primitives, an ellipse to represent the vision range of the sentry and a textured rectangle which uses an image to represent the sentry robot itself. The primitives employ layers to ensure that the sentry image is not obscured by the vision ellipse.

3.3 Instance Configuration

In addition to defining the structure of application components, the application descriptor contains declarations of application instances in different configurations. The instances contain different numbers of components like agents

Finally, an observer instance declared. This starts an instance of the observer once the application is started. The observer automatically connects to the space instance and presents the user with an overview according to the definition of the graphical representation discussed earlier.

4 Related Work

The focus of EnvSupport is two-fold. On the one hand, it provides generic concepts for building many kinds of environments with clearly defined extension points for any required additional features. On the other hand, it achieves a clean separation of agent and environment design and implementation, supporting a multitude of use cases in the areas of simulation, benchmarking, environment-based coordination, gaming, etc. Thus, in the following we consider simulation toolkits, which allow building virtual environments, as well as other work that focuses on the relationship between agents and the environment.

Many specialized simulation toolkits such as NetLogo[1] and Repast Simphony[2] offer generic concepts and tools for building simulation environments in a rapid prototyping like fashion. Moreover, these systems commonly offer built-in means for statistical evaluation of simulation runs. In fact, EnvSupport has been inspired by the functionality and also some concepts of these systems, e.g. by the 'context' and 'projection' concepts found in Repast Simphony. Yet, unlike agent platforms such as JADE[3] and Jadex, these systems usually implement simple agent models without sophisticated means of specifying agent behavior or message-based agent interaction. Moreover, the implementation of agents and the environment is often highly intertwined in these toolkits, which facilitates an easy creation of simulation models but has also drawbacks with regards to a clean application design.

A clean separation between agent and environment aspects is considered in work of e.g. [Weyns *et al.*, 2007], which argues for an explicit representation of an environment in agent applications (e.g. for coordination purposes). This work is focused on the generic concept of environment and the nature of

[1] http://ccl.northwestern.edu/netlogo/
[2] http://repast.sourceforge.net/
[3] http://jade.tilab.com/

the agent/environment interaction. It is not geared towards supporting quick environment implementations based on reusable components. A specific model of agent/environment interaction is the A&A model, which is implemented by the CArtAgO middleware [Ricci *et al.*, 2007]. Yet, the A&A model proposes a specific paradigm and is thus less generic than e.g. general purpose simulation toolkits.

Reusability is also addressed by the Environment Interface Standard (EIS) initiative [Behrens *et al.*, 2009], which aims at providing a standardized interface for any kind of agent environment, such that once developed environments can be easily reused from any kind of agent platform that implements EIS. EnvSupport and EIS share many conceptual similarities (e.g. avatars, percepts, actions) and can be seen as complements to each other. EIS is an interface for sharing (existing) environments, while EnvSupport provides reusable components for building (new) environments. Therefore, EIS-compatibility is a long-term goal of EnvSupport, allowing easy reuse of EnvSupport environments for other agent platforms beyond Jadex.

In summary, EnvSupport combines concepts and approaches from different strands of existing work and tools with the aim at bringing together features from simulation toolkits (rapid prototyping of spatial environments, statistical evaluation) and agent/environment interaction (environment as first-class abstraction) with the power of a fully-fledged agent platform.

5 Conclusion

This paper has tackled the problem of constructing agent applications with spatial environments. It has been argued that spatial environment types have similar characteristics and their specification and functionalities can thus be integrated into a dedicated environment framework.

Such a framework, called "EnvSupport", was proposed in this paper. EnvSupport mainly takes care of handling the environment domain model, the agent environment interaction, the presentation and also evaluation aspects. For all those parts of the framework the core concepts have been elaborated and their specification has been made possible in a declarative manner. The usage of the approach has been verified by a reimplementation of all spatial Jadex examples, from which the MarsWorld has been described here in more detail.

EnvSupport has been designed with several design rationales in mind. The first one is the completeness of the approach, meaning that a broad range of use cases should be easily realizable. This has been achieved by offering default implementations for all important core framework aspects (e.g. a 2d continuous and grid world, percept generators for local visions, space executors for continuous and round-based processing). The second one is the extensibility of the approach. If a use case should not be implementable using the available means the framework offers clear extension points, in which new functionalties can be easily deployed (e.g. a 3d world). Finally, EnvSupport allows a broad range of applications being developed by cleanly separating the agent from the environmental aspects. Examples are beyond pure simulation scenarios and include benchmarking real applications in a simulation testbed and also real applications that make use of a virtual spatial representation, e.g. for coordination purposes.

As future work we plan extending EnvSupport in several directions. One idea is the addition of a polygon space with possibilities for collision detection. Another interesting area is the addition of fields for naturally representing environment effects or forces.

References

[Behrens *et al.*, 2009] T. M. Behrens, J. Dix, and K. V. Hindriks. Towards an environment interface standard for agent-oriented programming (a proposal for an interface implementation). Technical report, Clausthal University, 2009.

[Braubach *et al.*, 2005] L. Braubach, A. Pokahr, D. Moldt, and W. Lamersdorf. Goal Representation for BDI Agent Systems. In *Proceedings of the 2nd International Workshop on Programming Multiagent Systems (ProMAS 2004)*, pages 44–65. Springer, 2005.

[Pokahr and Braubach, 2009a] A. Pokahr and L. Braubach. From a research to an industrial-strength agent platform: Jadex V2. In *9. Internationale Tagung Wirtschaftsinformatik (WI 2009)*, pages 769–778. Oesterreichische Computer Gesellschaft, 2 2009.

[Pokahr and Braubach, 2009b] A. Pokahr and L. Braubach. The notions of application, spaces and agents — new concepts for constructing agent applications. In *Multikonferenz Wirtschaftsinformatik 2009*, 2009.

[Ricci *et al.*, 2007] A. Ricci, M. Viroli, and A. Omicini. The A&A programming model and technology for developing agent environments in MAS. In *Programming Multi-Agent Systems, 5th International Workshop (ProMAS 2007)*, pages 89–106. Springer Berlin / Heidelberg, 2007.

[Rohlf and Helman, 1994] J. Rohlf and J. Helman. IRIS performer: a high performance multiprocessing toolkit for real-time 3d graphics. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 381–394, New York, NY, USA, 1994. ACM.

[Russell and Norvig, 1995] S. Russell and P. Norvig. *Artifical Intelligence: A Modern Approach.* Prentice-Hall, 1995.

[Weiss *et al.*, 2010] G. Weiss, L. Braubach, and P. Giogini. Intelligent agents. In *Handbook of Technology Management*, chapter 22. Wiley, 2010.

[Weyns *et al.*, 2007] D. Weyns, A. Omicini, and J. Odell. Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, 2007.