

An Ontology-Driven Framework for Deploying JADE Agent Systems

Csongor Nyulas¹, Martin J. O'Connor¹, Samson Tu¹, David L. Buckeridge²,
Anna Akhmatovskaia², Mark A. Musen¹

¹Stanford University, Stanford Center for Biomedical Informatics Research

²McGill University, Department of Epidemiology and Biostatistics

csongor.nyulas@stanford.edu

Abstract

Multi-agent systems have proven to be a powerful technology for building complex distributed applications. However, the process of designing, configuring and deploying agent-based applications is still primarily a manual one. There is a need for mechanisms and tools to help automate the many development steps required when building these applications. Using the Semantic Web ontology language OWL and the JADE platform we have developed a number of models and associated software tools that provide an end-to-end solution for designing and deploying agent-based systems. This solution supports the construction of detailed models of agent behavior and the automatic generation and deployment of agents from those models. We illustrate its use in the construction of a complex multi-agent system called BioSTORM that supports the configuration, deployment, and evaluation of analytic methods for detecting outbreaks of infectious diseases using public-health surveillance data.

1. Introduction

Multi-agent systems have been used in many domains to construct distributed component-based applications. While there are a large variety of methodologies and tools to assist in the various stages in the building of these systems, there are few integrated end-to-end approaches. Ideally, such an approach would support the detailed modeling of all aspects of agent behavior and the generation of deployable systems from those models. At a minimum it would require the ability to (1) model the agents in a system and the interfaces of those agents; (2) describe the information consumed and generated by each agent and by the overall application; (3) outline the possible inter-relationships between agents and the communication paths between them; and (4) specify the content of information exchanged between agents along

those paths. These specifications should be computer interpretable and provide sufficient detail to drive the construction and deployment of agents in a running system. Additional desirable features include the ability to easily modify system models and rapidly generate new deployments.

In this paper, we outline an ontology-based framework that supports this approach for building agent-based systems. This framework is targeted to the JADE [1] agent environment and uses the Web Ontology Language (OWL; [2]) as its modeling language. It provides a suite of open source tools for managing the various stages in describing and deploying agent-based systems. These tools are built as plug-ins to the popular Protégé-OWL [3] ontology development environment. Our framework provides an integrated process to support the modeling and the configuration of agents in a JADE platform, the information flow and content in the system, and ultimately the deployment of agents from those models. We illustrate the use of this framework in the construction of a complex multi-agent system called BioSTORM that supports the configuration, deployment, and evaluation of analytic methods for detecting outbreaks of infectious diseases using public-health surveillance data.

2. Related Work

Several projects have used ontologies in agent-based systems. Laclavik [4] has developed an agent architecture using a semantic knowledge model to define the behavior of agents. He has also built a library of JADE agents described using the semantic model. Tian [5] presents in his work an ontology-driven multi-agent architecture with the goal of supporting the sharing and reuse among different types of knowledge acquisition agents.

Other agent systems use ontologies to describe agent behavior. Şandru [6] proposes a generic multi-

agent task-oriented architecture based on a formal model described using the Unified Problem-solving Method description Language (UPML) [7]. Gómez [8] also describes a UPML-based framework to build information agents by reusing a library of domain-independent problem solving components. Hajnal [9] developed a Protégé-OWL plug-in to generate JADE agent code from ontology-based descriptions for the K4Care system [10]. His work and the plug-in is specific to the K4Care platform and cannot be used for general agent-based system generation.

All the mentioned projects have used formal models to describe various aspects of an agent system. However, to our knowledge, no agent system uses ontologies to describe all major aspects of a system with the final goal to support automatic generation of deployable systems. Such end-to-end systems should support the definition of detailed models to describe data sources, agent behavior, deployment configurations, communication pathways, message content, and then automatically generate a running system from those models.

3. System architecture

We have developed a JADE-based architecture for deploying a network of agents that focus on collaboratively analyzing one or more streams of data. This architecture assumes that the execution of the elementary tasks can benefit from distributed and parallel execution, a certain degree of autonomy and/or mobility, and that the execution of the individual tasks is data-driven. It views a configuration of agents as an algorithm that is to be applied to process data. We have adopted approaches from the knowledge modeling community to declaratively represent the procedural structure of these algorithms [7, 11, 12]. These approaches model knowledge about systems with respect to their goal or the task that they perform, and most share a methodology referred to as *task analysis* [13]. This methodology is used to construct algorithms or *problem solving methods* (PSMs) to solve particular tasks. In this methodology, a task defines "what has to be done". Tasks are accomplished by application of a method, which defines "how to perform a task". A method can either perform a task directly, or decompose a task into subtasks with constraints on their execution order, and delegate their execution to other methods. This recursive decomposition process – referred to as *task-method decomposition* – creates a tree-like structure representing the modeled process.

The architecture presented in Figure 1 has three layers: (1) a **knowledge layer**, containing all the

models that describe the problem decomposition, (2) an **agent platform**, which contains the deployed agents in a system, and a (3) **data source layer**, which represents a semantically characterized view of the external environment of the agents.

3.1. The knowledge layer

The knowledge layer provides several ontologies and software packages to completely specify the functionality of an agent-based system described using the task-method decomposition technique. This layer is composed of two main interrelated parts: a) the problem solving methods library and, b) the ontologies that define core system components.

The **Problem Solving Methods Library** is used to define a collection of software implementations of methods. An implementation of this architecture should provide a general problem solving method API that can be implemented by the methods. This interface defines the API used by agents to integrate methods into a running system. For example, in the BioSTORM system (see Section 5), the PSM Library contains a set of surveillance methods that employ the R open-source statistical software to detect incidents of infections.

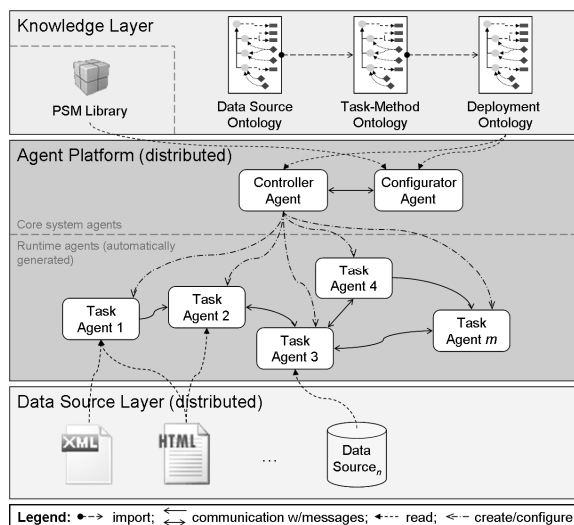


Figure 1. Overview of general system architecture

Three ontologies describe all core entities in a system.

The **Data Source Ontology** describes the data elements that system agents have to deal with at runtime. In this ontology, data elements are referenced as *variables* (e.g., the number of influenza diagnoses at an emergency room on a given day). A *tag* represents a

bundle of variables that are logically related to each other (e.g., the date, the emergency room identifier, and the number of flu diagnosis in the emergency room on that day). These bundles are used to describe the inputs and outputs of a task. In our architecture, the tag represents the basic communication unit between the agents in the deployed system. This data source ontology does not aim to tackle general data integration issues. Instead, it focuses on describing relatively simple data structures and assumes that a user of the system will perform the necessary transformations of source data. In many cases, this transformation process may be a relatively direct extraction of data from, say, a relational database; in other cases, complex transformations may be required.

The **Task-Method Ontology** specifies a configuration of a problem described using the task-method decomposition approach. It defines classes to model tasks, methods, connectors (that specify communication paths among tasks), algorithms (that consists of a collection of related tasks), together with detailed properties of those classes. The modeler of a specific system will define all the necessary domain- and problem-specific subclasses of the above classes and configure them with appropriate properties. For example, a method to compute the mean of a vector of values must define at least the properties *vector* and *mean*. The description of the problem will be realized by instantiating the user-defined subclasses and setting appropriate values to their properties. These property values together specify which task will use which method, which variables play which roles in the instantiated methods, and which tags are written and read by a task. Thus, for example, we may instantiate the method to compute the means of a vector of values for the task of computing the mean value of the baseline count of influenza infections, specifying that the *vector* property is associated with a sequence of counts values, and the *mean* property is associated with an output variable. At runtime, the values of these variables, packaged as tags, are the payloads in task-to-task communication.

The **Deployment Ontology** defines deployment configurations for the overall task defined in the task-method ontology. It specifies all the necessary information for a given system deployment, including values for the system configuration variables, initial inputs to some agents to trigger their execution, distribution policies, system output variables, variables for profiling and performance measurement, and so on.

3.2 The Agent Platform

The middle layer of our architecture is represented by a JADE agent platform that generate runtime agents based on the information encoded in the knowledge layer. This layer contains two special purpose agents: a **Controller Agent** and **Configurator Agent**. At deployment time, given a deployment configuration instance, the controller agent invokes the configuration agent to provide the necessary information to allow it to create and instantiate all the JADE task agents that are involved in a system. The configurator agent generates this information for the controller agent based on the task-method ontology and the problem solving method library. After each task agent is created and configured with its associated method they are instructed by the controller agent to execute their methods. The task agents will run independently and unsupervised until they either finish their job successfully, fail during their execution, or they are paused, stopped or destroyed by the controller agent.

In the case of data-driven systems, it is often necessary that some initial inputs are sent to some agents. The generation of the initial data is created by the configurator agent at the initiation of the controller agent based on information contained in the deployment ontology.

3.3. The Data Source Layer

The data source layer describes the environment with which the task agents are interacting during their execution. The description of this environment is based on the Data Source Ontology form the knowledge layer, and the methods associated to the tasks that interact with the environment must know how to translate/map the terms in the OWL description of the data, data types and data source to the representation in the actual physical resource.

4. BioSTORM System Requirements

New public health threats and recent advances in the capture and transmission of electronic health data are transforming public health surveillance. Public health authorities now have real-time access to data sources that tend to exhibit great variation and be of much higher volume than the data that was historically available. Automated surveillance systems play a central role in the analysis of these data. By using statistical aberration detection algorithms to automatically identify signals of public health interest, these systems allow analysts to sift through large

volumes of data efficiently. As a result, analysts can focus on decisions about public health as opposed to performing low-level data analysis.

However, there is a risk of missing or belatedly recognizing an outbreak or generating excessive false alerts if public health personnel select a suboptimal algorithm for a given surveillance context. Missing or belatedly recognizing an outbreak can result in billions of dollars of avoidable costs and frequent false alarms waste scarce public health resources and degrade confidence in a surveillance system. Our team and other groups [14] have identified important differences in the accuracy and timeliness of outbreak detection when different algorithms are applied to different types of surveillance data. Understanding these differences in detection performance is crucial to avoiding potentially serious errors and to guiding research in detection algorithms. As the adoption of automated surveillance systems increases, understanding the determinants of their performance is essential, as is developing new methods for configuring them intelligently.

One of the main goals of researchers is to develop fundamental knowledge about the performance of aberrancy detection algorithms used in public health surveillance. Our hypothesis that, through empirical experimentation, we can discover such knowledge that will allow surveillance system developers to select an appropriate algorithm, given a data stream and a surveillance goal, we have built a software infrastructure, called BioSTORM, as a computational environment for exploring how different types of aberrancy detection algorithms perform when applied to different types of surveillance data sources and outbreak signals.

In order to rapidly conduct studies and be able to integrate the evidences about the performance of aberrancy detection algorithms when applied on different data, defining a comprehensive and extensible model to represent the surveillance domain is of a key importance. This model will contain computer-interpretable description of the characteristics of statistical methods and algorithms applied in the surveillance domain and the evaluation of those algorithms.

Most evaluations of aberrancy-detection algorithms have significant processing requirements due to the need to analyze multiple data sets, often over a range of algorithm settings. These evaluations usually contain multiple tasks that can be executed in parallel. As a result, many surveillance algorithm evaluations can often benefit from task distribution. In some settings the data sets used in the analysis may be of sensitive nature or physically remote, where fetching

all the data necessary to the analysis into a centralized facility is not feasible. Agent-based systems provide the appropriate tools to support these requirements.

For simplicity we will subsequently refer to the aberrancy detection methods and algorithms as surveillance methods and surveillance algorithms.

In order to facilitate surveillance algorithm evaluations, the users of the BioSTORM system should be able to rapidly configure, reconfigure and execute deployments of surveillance algorithms. These alternative deployments can be created by specifying different values for configuration variables or applying the algorithms to different data sources. They should be also able to easily define new surveillance methods, surveillance algorithms and deployment of those algorithms, without any knowledge about agent-based systems or programming in general.

5. The BioSTORM System Implementation

The BioSTORM system (Figure 2) is based on the three-layer architecture outlined in Section 3. All layers were adapted to the task of modeling and evaluating a range of surveillance algorithms when applied to different surveillance data sources.

The steps involved in generating a BioSTORM implementation of this architecture and specifying a deployment are as follows:

- (1) Create system ontologies
- (2) Create domain methods
- (3) Create configuration knowledge bases
- (4) Generate JADE interfaces from the model
- (5) Deploy and execute

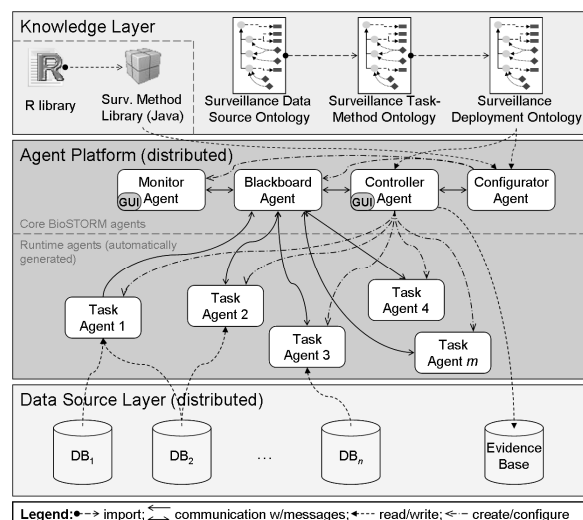


Figure 2. Surveillance-specific BioSTORM system architecture

In the following sections, we detail this process and describe the ontologies and software components that we developed to provide the functionality of each step (Figure 3).

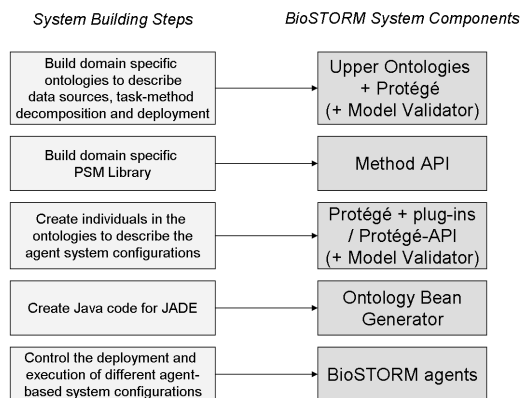


Figure 3. Steps and corresponding software components involved in generating a BioSTORM deployment

5.1. Create domain specific data-source, task-method and deployment ontologies

BioSTORM provides three ontologies that encode core system concepts (Figure 2). These three ontologies are (1) a Surveillance Data Source Ontology; (2) a Surveillance Task-Method Ontology; and (3) a Surveillance Deployment Ontology. These ontologies are specializations of the Data Source, Task-Method, and Deployment upper ontologies. These BioSTORM-specific ontologies are built to provide a formal, reusable description of the surveillance domain in a machine-interpretable form. They provide detailed definitions of a deployable, agent-based system in JADE. All BioSTORM ontologies were encoded in the Web Ontology Language using *Protégé-OWL*, an open-source ontology development platform. Protégé-OWL provides both an authoring environment for ontology editing and a software layer for building ontology-driven applications.

The **surveillance data source ontology** plays a central role in data integration. It can either provide a direct binding of SQL queries to input variables used in surveillance or provide *case definitions* of interest to surveillance. These case definitions provide an abstraction layer so that surveillance algorithms need not be concerned with low-level data formatting issues.

The **surveillance task-method ontology** defines a typology of surveillance-detection tasks and methods. It also identifies eligible methods for each task and describes salient characteristics of individual methods.

Concepts related to control and data flow to represent algorithms, iterations and input-output connections among the tasks are also described.

The **surveillance deployment ontology** provides a formal specification of deployments of surveillance algorithms. It has an important role in integrating surveillance algorithms with other operational components of the system (e.g., those supplying surveillance data for analysis, or post-processing the results of aberrancy detection). The ontology includes concepts describing properties of the surveillance data and configuration of deployments. Annotating the results of aberrancy detection and the results of performance-evaluation analysis with this information is crucial for interpreting or re-using these results.

5.2. Create a domain specific PSM library

The implementation of the BioSTORM system provides a simple general purpose *Problem Solving Method API* in Java that is used by the generated task agents to interact with the methods. This API is domain independent, and can be used in implementation of systems that employ PSMs for other domains. Using this API we have built a **Surveillance Method Library** as an extensible Java package containing a set of surveillance methods. The methods in this library use the R open-source statistical software [15] for any non-trivial statistical computations.

5.3. Describe an agent-based system, its deployment configurations and its semantically characterized environment

The users of the systems based on the proposed architecture can fully describe an agent-based system deployment by instantiating classes defined in the task-method ontology. For example, using BioSTORM, a statistician or public health specialist can fully describe a surveillance algorithm by instantiating classes defined in this ontology. Protégé-OWL provides a graphical modeling construction tool, called GraphWidget, that we use to support this authoring process (Figure 4). It allows users to describe algorithms by graphically constructing networks of tasks and the communication pathways between them.

To make such a system deployable the user also has to characterize the agents' data sources. These are defined using the data source ontology. Unique deployment configurations are created by instantiating the surveillance deployment ontology.

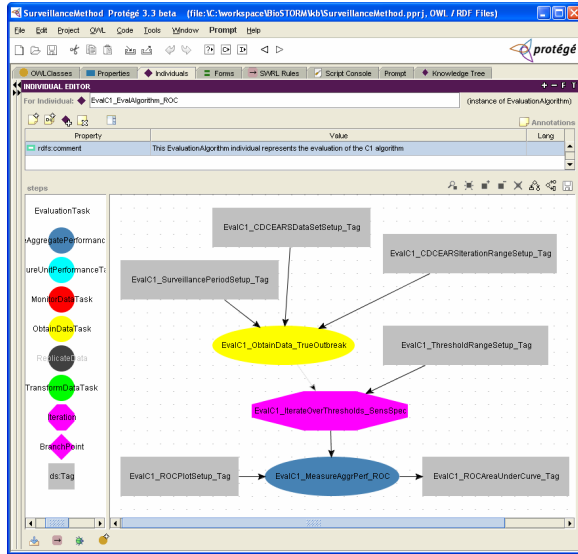


Figure 4. Screenshot of Protégé-OWL GraphWidget component showing the description of an algorithm as a network of tasks and tags

5.4 Ensure the soundness of the models by using the ModelValidator plug-in

To support the model authoring process, we have created a Protégé-OWL plug-in to validate task-method decompositions (i.e., surveillance algorithm descriptions, in case of the BioSTORM system). The plug-in also supports the automatic edit-time validation of the ontologies to ensure that editing different aspect of the model will be automatically reflected in other related aspects of the model. For example, if two tasks in a surveillance algorithm are connected through a connector, the validator plug-in will ensure the consistency of the model by automatically generating input and output configurations for both tasks. This dynamic checking helps to minimize errors and speeds up the specification process.

5.5 OntologyBeanGenerator plugin

The OntologyBeanGenerator plug-in [16] generates Java classes that represent a FIPA/JADE compliant ontology used by the JADE framework to interpret the content of messages exchanged between agents. We have extended an earlier version of this plug-in to support OWL. We have also added code generation support for dealing with OWL class hierarchies that use multiple inheritance. Most significantly, we extended this plug-in to offer an

option of generating a Java factory class to create Java beans based on instances in an OWL ontology. These beans can be used to define message content for agent to agent communication. We have also improved the usability of the plug-in by a variety of user interface enhancements.

The bean generator now offers all the necessary functionalities for fully defining the content of agent messages based on an OWL ontology, and for automatically generating Java code to read and write these messages. This plug-in dramatically simplifies the process of defining or modifying message content and significantly reduced the amount of message handling code that has to be written in an application. Our BioSTORM system ontologies, for example, have approximately 180 classes that define message content and we automatically generated over 12000 lines of code to handle these messages. Manually writing this amount of code in a bug-free and modifiable fashion would require considerable resources.

5.6. Deploying the system

Several specialized agents are used to control both the deployment of agents and run-time inter-agent communication.

Task Agent A task agent implements a particular task involved in a surveillance algorithm. Task agents are automatically generated and configured to use a surveillance method based on a deployment configuration. These agents are deployed on a single JADE platform, but they may be distributed in multiple containers across different machines.

Controller Agent A single controller agent on a platform controls the initialization, initiation and interruption of a deployment. It is responsible for the creation and initialization of all the task agents, as well as of the implementation of specific distribution policies. An associated graphical interface is provided to allow users to specify, initiate and control the execution of a deployment. A controller agent receives a deployment configuration from a configurator agent.

Configurator Agent A configurator agent translates OWL-encoded deployment configurations in the surveillance deployment ontology into agent messages, providing all the necessary information to the controller agent to create and initialize the task agents involved in the execution of a deployment. This translation uses the Java code generated by the

OntologyBeanGenerator to automatically generate message content.

Blackboard Agent Because of the data-driven nature of the surveillance algorithms and their construction from (partially) parallelizable and distributable tasks, we have found the blackboard architecture [17] suitable for inter-task communication in BioSTORM. The Linda [18] variant of this architecture can prove particularly suitable because it also provides synchronization and parallelization primitives. These primitives facilitate the coordination of all task agents involved in a deployment. The blackboard agent implements the Linda model and plays a central role in our system. All communication between task agents is through a blackboard. Results produced by a task agent are written to the blackboard, whereupon they are read by other tasks that need them as input. Task agents are unaware of the existence of other task agents, the origin of their input data, and the destination of their outputs. The use of a blackboard simplifies the development of tasks and methods considerably because individual methods need not be aware of any deployment or distribution requirements.

There is a single blackboard agent per agent container. Inter-container messaging among agents on different agent containers is automatically coordinated between the different blackboard agents in the system. The existence of multiple blackboard agents is transparent to the task agents.

Monitor Agent This agent monitors all blackboard-based communication. An associated graphical interface is provided for interactively specifying which information appearing on the blackboard is of interest to the user. It also provides both a visual and a file based logging functionality.

5.7. Evaluation Experiments

We exercised and evaluated the BioSTORM ontology-based agent architecture by instantiating the surveillance task-method ontology with a family of surveillance algorithms and by specifying and executing a number of surveillance configurations using simulated surveillance data from Center for Disease Control and Prevention (CDC).

The family of surveillance algorithm we encoded is the C-family algorithms from the CDC Early Aberration Reporting System (EARS) [19]. Three members of the family, called C1, C2 and C3, are adaptive algorithms based on a CUSUM control chart concept [20]. These algorithms are encoded in the

EARS software developed by the CDC and used widely for public health surveillance. We replicated and extended a study by Hutwagner [21], who compared C1, C2 and C3 algorithms in terms of the sensitivity, specificity, and time to detection. In our study, we used the same simulated surveillance data and followed the same evaluation steps as the original study [22].

The original simulated data included 56 distinct sets with varying baseline characteristics. We evaluated our performance against a randomly selected subset of 10 datasets. The comparative results for sensitivity, specificity, and time to detection for selected datasets closely match the original study. The absolute deviations between the values computed using BioSTORM and that study do not exceed 0.006 for sensitivity and specificity and 0.01 days for time to detection.

We have modeled a range of other surveillance algorithms using BioSTORM to ensure that the system ontologies can accurately represent typical surveillance algorithms. We have found that the automated generation of deployments from these ontologies facilitates both the rapid construction of surveillance algorithms and the rapid reconfiguration of those algorithms once they have been built.

6. Discussion

We have described a methodology and suite of tools to support the modeling and deployment of agents on the JADE platform. These models are encoded using the Semantic Web ontology language OWL and provide detailed computer-interpretable specifications of agent behavior in a JADE system. These specifications include agent configuration, agent distribution, message flow, message content, and descriptions supporting a number of data integration strategies. A suite of open source tools are provided to construct these models and produce a deployed system from them. We describe how we have used these models and tools to develop an agent-based system for rapidly generating complex configurations of agents to analyze streams of public-health surveillance data to detect outbreaks of infectious diseases.

The BioSTORM system is centered around data-driven information processing strategies and focuses on describing deployments of complex configurations of relatively simple agents to rapidly process large amounts of information. The implementation reflects this focus. For example, BioSTORM's blackboard communication paradigm proved very convenient for loosely-coupled, asynchronous and data-driven distributed systems, but it may not necessarily be

suitable for other communication strategies. As a result, while these alternative communication strategies can be modeled, the software will not directly support the fully automatic generation of deployments that use them. Additionally, the specification of agents is restricted to simple input-output descriptions and agent configurations are assumed to remain static throughout a deployment. Our system ontologies and software tools do not currently support dynamic agent behaviors.

The next evolution of BioSTORM will require much more elaborate descriptions of agents and their behavior. These behaviors will include automatic agent selection for particular data streams and strategies that dynamically redeploy agents in response to changes in those streams. We are extending our ontologies and tools to support these additional requirements. However, the current system is sufficiently rich to deal with a large variety of typical problems. The system's task-method decomposition approach is a proven AI technique for composing complex systems from smaller computational units and has been used in many domains. The agent configuration and deployment ontologies, the OntologyBeanGenerator and ModelValidator plug-ins are independent of communication pathways or distribution strategies so they are directly reusable. All ontologies and software components described in this paper are available on our web site [23].

Acknowledgements

This research was supported by a grant from the Centers for Disease Control and Prevention under the BioSense Initiative to Improve Early Event Detection (RFA-PH-05-126) and the Protégé Resource Grant from the National Institutes of Health (NLM LM007885). David Buckeridge is supported by a Canada Research Chair in Public Health Informatics.

References

- [1] Bellifemine FL, Caire G, Greenwood D. Developing multi-agent systems with JADE: John Wiley & Sons; 2007.
- [2] Web Ontology Language (OWL). Available at <http://www.w3.org/2004/OWL/>
- [3] Knublauch H, Ferguson R, Noy NF, Musen MA. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In: Third International Semantic Web Conference. Hiroshima, Japan; 2004.
- [4] Laclavik M, Balogh Z, Babik M, Hluchý L. AgentOWL: Semantic Knowledge Model and Agent Architecture. *Computers and Artificial Intelligence* 25(5): (2006)
- [5] Tian G and Cao C. An Ontology-Driven Multi-Agent Architecture for Knowledge Acquisition from Text in NKI. *CIMCA/IAWTIC* 2005: 704-709.
- [6] Şandru C, Negru V, Pop D. A Multi-Agent Problem Solving Architecture based on UPML. *Artificial Intelligence and Applications* 2005: 597-602.
- [7] Fensel D, Motta E, van Harmelen F, Benjamins VR, Crubézy M, Decker S, Gaspari M, Groenboom R, Grosso W, Musen MA, Plaza E, Schreiber G, Studer R and Wielinga R. The unified problem-solving method development language UPML. *Knowledge and Information Systems Journal (KAIS)* 2003 5(1): 83-131.
- [8] Gómez M, Abasolo C, Plaza E: Domain-Independent Ontologies for Cooperative Information Agents. *CIA* 2001: 118-129.
- [9] Hajnal Á, Pedone G, Varga LZ. Ontology-Driven Agent Code Generation for Home Care in Protégé. *Proceedings of the 10th International Protégé Conference*, pp. 91-93. Budapest, Hungary, 2007.
- [10] K4CARE: Knowledge-Based HomeCare eServices for an Ageing Europe. EU-Project. <http://www.k4care.net/>
- [11] Chandrasekaran B, Johnson TR. Generic tasks and task structures: History, critique and new directions. In: David J-M, Krivine J-P, ed. *Second generation expert systems*. Berlin: Springer-Verlag; 1993, p. 232-272.
- [12] Steels L. Components of expertise. *AI Magazine* 1990; 11: 30-49.
- [13] Chandrasekaran B, Johnson TR, Smith JW. Task-structure analysis for knowledge modeling. *Communications of the ACM* 1992; 35: 124-137.
- [14] Buckeridge DL. Outbreak detection through automated surveillance: A review of the determinants of detection. *Journal of Biomedical Informatics*, 2007 Aug; 40(4):370-379.
- [15] R Development Core Team. R: A language and environment for statistical computing. In. Vienna, Austria: R Foundation for Statistical Computing; 2006.
- [16] OntologyBeanGenerator wiki page. Available at <http://protegewiki.stanford.edu/index.php/OntologyBeanGenerator>
- [17] Englemore R and Morgan T, editors. *Blackboard Systems*. Addison-Wesley, 1988.
- [18] Carriero N, Gelernter D. *How to Write Parallel Programs: A Guide to the Perplexed*. ACM Computing Surveys. 1989, 21(3).
- [19] Centers for Disease Control and Prevention (CDC): The Early Aberration Reporting System (EARS). <http://www.bt.cdc.gov/surveillance/ears/>
- [20] Hutwagner L. The bioterrorism preparedness and response Early Aberration Reporting System (EARS). *Journal of Urban Health* 2003;80: 89-96.
- [21] Hutwagner L, Browne T, Seeman MG, Fleischauer AT. Comparing aberration detection methods with simulated data. *Emerging Infectious Diseases* 2005;11: 314-316.
- [22] Buckeridge D, Okhmatovskaia A, Tu S, O'Connor MJ, Nyulas C. Understanding Detection Performance in Public Health Surveillance: Modeling Aberrancy-Detection Algorithms. *JAMIA*, 2008, *in press*.
- [23] BioSTORM project homepage. Available at <http://biostorm.stanford.edu/>