Published in final edited form as:
Artif Intell Eng Des Anal Manuf. 2009 November ; 23(Spec Iss 4): 339–356. doi:10.1017/S0890060409990047.

Software-engineering challenges of building and deploying reusable problem solvers

MARTIN J. O'CONNOR[1], **CSONGOR NYULAS**[1], **SAMSON TU**[1], **DAVID L. BUCKERIDGE**[2], **ANNA OKHMATOVSKAIA**[2], and **MARK A. MUSEN**[1]

[1]Stanford Center for Biomedical Informatics Research, Stanford University, Stanford, California, USA

[2]Department of Epidemiology and Biostatistics, McGill University, Montreal, Canada

Abstract

Problem solving methods (PSMs) are software components that represent and encode reusable algorithms. They can be combined with representations of domain knowledge to produce intelligent application systems. A goal of research on PSMs is to provide principled methods and tools for composing and reusing algorithms in knowledge-based systems. The ultimate objective is to produce libraries of methods that can be easily adapted for use in these systems. Despite the intuitive appeal of PSMs as conceptual building blocks, in practice, these goals are largely unmet. There are no widely available tools for building applications using PSMs and no public libraries of PSMs available for reuse. This paper analyzes some of the reasons for the lack of widespread adoptions of PSM techniques and illustrate our analysis by describing our experiences developing a complex, high-throughput software system based on PSM principles. We conclude that many fundamental principles in PSM research are useful for building knowledge-based systems. In particular, the task–method decomposition process, which provides a means for structuring knowledge-based tasks, is a powerful abstraction for building systems of analytic methods. However, despite the power of PSMs in the conceptual modeling of knowledge-based systems, software engineering challenges have been seriously underestimated. The complexity of integrating control knowledge modeled by developers using PSMs with the domain knowledge that they model using ontologies creates a barrier to widespread use of PSM-based systems. Nevertheless, the surge of recent interest in ontologies has led to the production of comprehensive domain ontologies and of robust ontology-authoring tools. These developments present new opportunities to leverage the PSM approach.

Keywords

Knowledge-Based Systems; Problem Solving Methods; Reusable Problem Solvers; Software-Engineering Challenges; Task–Method Decomposition Process

1. INTRODUCTION

A major goal of research in reusable problem solving methods (PSMs) is to provide libraries of predefined, implemented algorithms that developers can use to construct knowledge-based systems (Chandrasekaran, 1986). The aim is to bring the promise of software reuse to these systems, permitting the rapid assembly of knowledge-based applications. Approaches

Reprint requests to: Martin J. O'Connor, Stanford Center for Biomedical Informatics Research, Medical School Office Building, Room X-215, 251 Campus Drive, Stanford University, Stanford, CA 94305-5479, USA. martin.oconnor@stanford.edu.

for meeting this goal include techniques to support the specification and construction of PSMs and their assembly into application systems. The knowledge-based system community has converged on a set of fairly well-understood requirements and common terms for building systems from PSMs. The two main requirements are the abilities to compose complex algorithms from simpler algorithms and to reuse these algorithms in knowledge-based systems.

To define the control knowledge for an intelligent system, a technique known as *task decomposition* allows developers to construct task models via recursive application of PSMs as building blocks (Chandrasekaran et al., 1992). Basic high-level tasks are at the root, and progressively more granular subtasks are at lower levels (Fig. 1). Developers select or create PSMs to model the control knowledge needed to solve a task; each PSM may entail subtasks, which themselves are modeled using PSMs that in turn may entail subtasks. At a minimum, each PSM provides *input* and *output relations*, which describe the task's data and knowledge requirements and its results. This modular approach to PSM assembly facilitates much more flexible PSM reuse than would be possible with monolithic PSMs. The clear specification of inputs and outputs of all subtasks in a PSM also produces a detailed picture of knowledge requirements. These specifications can also assist in knowledge acquisition (Marcus et al., 1988). A PSM is configured for deployment in a particular application domain by mapping its input–output specification to the relevant domain knowledge and available types of input data. This mapping describes transformations between the inputs of a PSM and their referents in the domain, which provides the PSM with the static knowledge and the data on which it needs to operate.

Each PSM in a library tackles a particular class of problems. In the abstract sense, a PSM is responsible for automating a class of tasks using knowledge and data from a particular domain. It describes the reasoning steps and the types of knowledge used during the problem solving process, independent of the domain. Developers build new knowledge-based systems by selecting (or creating or modifying) appropriate PSMs and configuring them to work together to solve a task. Ideally, the configuration process would be at least partially automated by using a set of overall design goals provided by the developer and self-descriptions provided by each PSM (Fensel et al., 2002).

Interest in PSM-driven systems emerged in the knowledge-based systems community in the 1980s in response to the difficulties encountered when developers attempted to scale up rule-based approaches to handle large applications (McDermott & Bachant, 1984; Clancey, 1986). As developers of knowledge-based systems sought mechanisms to make explicit the procedural knowledge required to model and implement intelligent systems, the notion of reusable PSMs became very alluring. Knowledge acquisition tools that could elicit the domain knowledge for well-understood PSMs such as *propose and revise* and *cover and differentiate* made the approach appealing (Marcus et al., 1988). The Sisyphus experiments (Rothenfluh et al., 1995) advanced at the Knowledge Acquisition for Knowledge-Based Systems Workshops offered particularly compelling examples of how PSMs could encapsulate control knowledge and enable the construction of more maintainable systems. Suddenly, PSMs allowed developers to clarify the role that every knowledge-base entry played in problem solving, and to use the knowledge requirements of PSM to guide knowledge elicitation.

Recognizing the limitations of unitary PSMs, developers sought to develop mechanisms to create assemblies of PSMs that could better accommodate the nuances of application tasks (McDermott & Bachant, 1984; Schreiber et al., 1999; Steels, 1990). Our own group's Protégé project attempted precisely this kind of approach (Eriksson et al., 1995). The use of Chandrasekaran's task–decomposition modeling approach (Chandrasekaran et al., 1992),

coupled with the mapping of domain ontologies to the knowledge requirements (Gennari et al., 1994), seemed like the obvious solution to the challenge of building component-based intelligent systems (Musen & Schreiber, 1995; Tu et al., 1995).

Despite the early enthusiasm, no widely available set of implemented PSMs or robust PSM–development environments exist, and the original challenges addressed by this avenue of research remain unsolved for practitioners who wish to build knowledge-based systems. The engineering of knowledge-based systems is still extremely difficult, and, as a practical matter, reusing control knowledge is arguably no easier now than it was when the research began. A central problem is the lack of tools to support building PSM-based systems. In addition, there is a significant gap between the current capabilities of commercial environments for the development of knowledge-based systems and the requirements for constructing PSM-based software.

This paper explores the challenge of building a PSM-based system using contemporary software-engineering tools. We illustrate these challenges by describing the development of two versions of a system called the Biological Spatio-Temporal Outbreak Reasoning Module (BioSTORM; Crubézy et al., 2005; Buckeridge et al., 2008). BioSTORM supports the configuration, deployment, and evaluation of analytic methods for detecting outbreaks of infectious diseases using public health surveillance data. We created BioSTORM as a PSM-based application, and we use PSMs as central system components. Over the past 6 years, we have produced two versions of BioSTORM, each with different goals. The first system focused on analyzing a variety of data sources, which were described with a rich domain ontology (Crubézy et al., 2003). The system was built with relatively simple PSMs. The second system used fewer data sets and instead concentrated on the configuration of more complex PSMs capable of performing more elaborate analyses (Buckeridge et al., 2008). Because of their different analysis requirements, both implementations served to emphasize two key and complementary challenges in the development of PSM-based systems: the first system's focus on analyzing a large number of data sources necessitated elaborate mappings from PSMs to domain data sources, and the large number and complex interactions between the second implementation's PSMs drove the development of a software infrastructure to control the deployment and management of these PSMs. Both implementations serve to address common problems in the development of PSM-based systems: how PSM interact with their environment and how they interact with each other in a deployed system.

The second iteration was particularly noteworthy becausewe produced a clean sheet design that could correct the shortcomings of the first implementation. The two BioSTORM implementations highlight the difficulties both of domain modeling and software engineering in the creation of PSM-based systems. Our work demonstrates that, whereas PSMs are extremely useful as conceptual building blocks that can facilitate task analysis and knowledge acquisition for intelligent systems, considerable practical problems remain in the use of PSMs as software components for implanting deployable applications.

2. BioSTORM: SYNDROMIC SURVEILLANCE USING PSMs

In recent years, public health surveillance has become a national priority, driven by concerns of possible bioterrorist attacks and disease outbreaks. Authorities argue that *syndromic surveillance*, or the monitoring of prediagnostic health-related data for early detection of nascent outbreaks, is crucial to preventing massive illness and death (Buehler et al., 2003; Lombardo et al., 2003; Tsui et al., 2003). Rapid outbreak detection is important because public health interventions generally are most effective when applied early in an outbreak. In a bioterrorist attack involving anthrax, for example, a delay of even hours in administering prophylactic antibiotics can reduce survival substantially. A syndromic

surveillance system could sound an early warning by detecting an abnormal increase in clinic visits or pharmaceutical purchases before the first definitive diagnoses are made. The desire to improve approaches to syndromic surveillance and the increasing availability of electronic data on which such systems might operate have resulted in a blossoming of projects to develop new surveillance systems. Public health analysts, however, face technical barriers to incorporating heterogeneous data sources into surveillance systems and, more important, to integrating the data sources in a way that offers semantic coherence and that improves decision making.

Syndromic surveillance systems have unique data analysis requirements. For example, the straightforward time series algorithms used to summarize traditional surveillance data are unsuitable for integrating the data used in syndromic surveillance. The data used for syndromic surveillance typically are rich in spatial and temporal measurements, which analysts must aggregate and interpret. The high dimensionality, heterogeneity, and unpredictable nature of the data and of disease outbreak patterns require that systems have a range of analytic methods that can make sense out of data in numerous contexts. Moreover, these systems must combine analytic methods in potentially complex configurations. Analyzing surveillance data is a problem solving process that necessarily involves a set of methods tailored to numerous specific situations. Rather than using *ad hoc* approaches, surveillance systems must have an infrastructure for applying different analytic strategies to incoming data streams in a principled, context-dependent manner.

To address these requirements of syndromic surveillance, we have developed BioSTORM (Crubézy et al., 2005). Bio-STORM is an experimental end-to-end computational framework. It integrates disparate data sources and uses various analytic methods to support interpretation of surveillance data and the identification of disease outbreaks. Bio-STORM's goals are to provide a run time environment that supports rapid data analysis and a modular mediation framework that supports knowledge-based method selection. As mentioned, we have implemented two versions of Bio-STORM. The first version emphasized relatively simple analyses of a large variety of data sources, whereas the second version concentrated on elaborate analyses of a small number of data source types. Broadly speaking, the focus of the first version was to integrate a large number of diverse data sources for analysis; the second version shifted the focus with a much more ambitious goal of building an evaluation infrastructure that could be used to determine the fundamental determinants of the performance of analysis algorithms.

Our approach in both cases centered on using ontologies to model syndromic surveillance data and knowledge, and using PSMs to define the analytic methods needed to analyze surveillance data. The effort in developing both systems can be broadly divided into the processes of conceptual modeling and of software development. For clarity, we use the notation BioSTORM^{A} and BioSTORM^{B} to distinguish between the first and second system versions, respectively.

3. FIRST BioSTORM IMPLEMENTATION: PSMs AND DATA INTEGRATION

We built BioSTORM^{A} around four main components: a data-source ontology for describing and integrating the surveillance data sources that may serve as input to the system, a library of statistical and knowledge-based PSMs for analyzing the syndromic surveillance data, a mediation component with a data broker and a mapping interpreter for translating the input data into a form usable by the PSMs, and a controller for deploying PSM configurations to analyze incoming data streams. The data broker integrates multiple related data sources described by the data-source ontology, and the mapping interpreter feeds the integrated data to the appropriate PSMs.

3.1. Modeling: Data-source ontology

Public health surveillance data are diverse and are usually distributed electronically in databases and files with little common semantic or syntactic structure. Such data could include spreadsheets containing over-the-counter drug sales, semiformatted text files containing records of emergency calls or school and work absenteeism in a geographical area, as well as more traditional sources such as electronic medical records. Analyzing disparate data sources requires precise specification of knowledge about how to characterize and combine them. To apply appropriate analytical methods to relevant outbreak-detection data, the data sources must be related to one another, to descriptions of reportable conditions, and to enumerations of the primitive data on which specific diagnoses can be made. We developed a *template data source ontology* that allows coherent descriptions of these entities (Pincus & Musen, 2003). The ontology makes data self-descriptive by associating a structured, multilevel *context* with each data source. Developers describe individual data elements with metadata terms adopted from the Logical Identifier Names and Codes (LOINC; McDonald et al., 2003). The LOINC approach describes a piece of data along five major semantic axes, including "kind of property," "time aspect," and "scale" axes. Clinical pathologists, for example, use LOINC to contextualize results reported by clinical laboratories. We generalized the LOINC axes from this role into a generic set of descriptors for contextualizing many different types of data involved in syndromic surveillance. Our five axes are: what is being measured? (e.g., "Robitussin sales"); how is it measured? (e.g., "Cases sold per day"); when/for how long? (e.g., "Averaged over a week"); where? and what are the possible values? The data-source ontology aims to describe data in a domain-independent way. Domain-level concepts are described in domain ontologies, which may be based on the information encoded in the data-source ontology.

Our template data sources ontology describes contexts, such as the locations in the community where individual data streams may be derived. Each context can then be related to multiple data sources (e.g., different data streams at a particular type of location). However, instead of defining the structure of the data as a global model does, our template data sources ontology defines the structure of the data sources themselves and how metadata and explicit semantics should be associated with a given data source and its data. For example, with 911 dispatch data, patient data from an electronic medical record, and data related to reportable diseases, the data source ontology captures individual-level primitive data (such as signs, symptoms, and laboratory tests) as well as observable population-level data (such as aggregated syndrome counts and school absenteeism).

A developer then describes the context of a data source by filling in a template, specifying the physical source of the data, the ways in which related data are grouped, and the specific atomic data elements (Fig. 2, left). For example, an atomic data element such as an integer can be represented as a *datum*; it may then be associated with logically related data using a *data group*. These entities are then associated with a *datum context* and a *data group context*, respectively. Our template data-source ontology, as customized for syndromic surveillance, provides a taxonomy of data-source attributes to describe such contexts (Fig. 2, right). These attributes grouped into general categories from which users choose when modeling a particular data source, data group, or individual data type. For example, the template ontology may require that an address be associated with every data source context, but it is up to the user to choose from the provided subclasses of address, such as street address or Internet address, and create an instance of the chosen address type. In this manner, describing a set of data sources is reduced to choosing attributes and values from lists, because the user is freed from the requirement of defining the structure of the descriptions when defining their content.

The template data-source ontology therefore describes generic data sources; instances of those generic data sources encode the types of data fields of particular data sources that are processed by BioSTORM[A]. The ontology allows us to represent both primitive data relating to individuals (e.g., signs, symptoms, and laboratory tests obtain from health care facilities) and population-level data (e.g., aggregated syndrome counts and measures of school absenteeism).

Our ontology's systematic, template-directed process allows developers to create a customized local model of each data source. Each description of a data-source instance shares a common structure, space of attributes, and set of possible attribute values. While capturing each data source's specific characteristics, the template data-source ontology provides a framework for representing each data source in a uniform way. More precisely, the ontology provides a hybrid approach to data integration, in that it combines the semantic rigor of a global, shared ontology with the flexibility and level of detail that comes from devising customized, local ontologies for each data source. Most important, it provides an abstract view of data sources that is unconcerned with how the primary data actually are stored. This approach supports integrating heterogeneous surveillance data at the level of semantic reconciliation, allowing uniform application of PSMs to each data source.

3.2. Software development: Deploying surveillance PSMs

BioSTORM[A] has a library of PSMs that can analyze multiple, varying data types. Our development of these PSMs was guided by the Unified Problem Solving Method Description Language (UPML) for modeling PSMs and the tasks for which they are used (Fensel et al., 2002). The UPML framework follows the task–method decomposition approach, which models each PSM as potentially entailing several subtasks. Each subtask is solved by a PSM that may, in turn, entail new subtasks. Modeling continues until one or more primitive methods can solve each subtask.

There originally was considerable excitement for UPML when a consortium of investigators (including our group at Stanford) founded an international collaborative project known as IBROW[3] (Benjamins et al., 1998), setting out in the late 1990s to define an Internet-based architecture for locating, selecting, and deploying PSMs over the World Wide Web. UPML was a major deliverable of the IBROW[3] project, and our implementation of BioSTORM[A] was the first attempt to adopt UPML for use in a large-scale intelligent system. As we now discuss in detail, our attempt to use UPML to develop BioSTORM[A] was not entirely positive. Despite our initial enthusiasm for the approach, our later decision to reengineer our system as BioSTORM[B] was a direct result of the limitations of UPML that became apparent empirically when we built BioSTORM[A].

BioSTORM[A] has a library of PSMs that address the analysis of multiple, varying types of data for detecting time-oriented data aberrancies (Buckeridge, O'Connor, et al., 2004). Its library includes both generic, disease-independent statistical methods that analyze data as single or multiple time series, and knowledge-based methods that specifically relate detected abnormalities to knowledge about reportable diseases. Relatively straightforward PSMs are implemented as simple software routines, whereas complicated PSMs are incorporated into the system by "wrapping" existing software libraries so that they conform to the requirements of our method ontology (see below).

We developed an ontology for categorizing abnormality-detection algorithms, which we used to structure the Bio-STORM[A] PSM library. This framework is based on information contexts commonly encountered in surveillance work and the functional requirements of each abnormality-detection algorithm (Buckeridge et al., 2002). We created this framework through Chandrasekaran style task decomposition by modeling the overall surveillance task

as a set of specific tasks, and by decomposing each task into subtasks. We then created a *surveillance PSM ontology*, which models and classifies the surveillance methods that may automate each monitoring subtask (Fig. 3). With this ontology, our library is available as a computer-processable repository of PSMs that BioSTORM^{A} can index, query, and invoke.

Following the UPML approach, we associate each PSM in our library with a *method ontology* that defines the classes of data and knowledge on which the given method operates. For example, statistical PSMs have specific requirements for the structure of the statistical models on which they operate, such as whether they assume population data or individual data. Many algorithms expect time-series data at varying granularities, and certain ones require spatial data at several levels of aggregation. Our method ontology makes explicit a PSM's data requirements, enabling BioSTORM^{A} to apply the method uniformly to various data sources. The method ontology helps BioSTORM^{A} map data sources to appropriate PSMs and to reconcile the semantic differences between data and methods. For example, when configuring one of our surveillance methods to aggregate different data streams, where each stream reported on different 911 dispatches, we created a set of mappings to transform the contents of the different data streams into individual events as required by the aggregation method. Furthermore, the method ontology for a PSM can facilitate interoperation with other analytic methods by identifying appropriate interactions with other PSMs in the library. Overall, our framework provides a structure for incorporating surveillance algorithms into our system and for establishing their data and knowledge requirements. By making each method's characteristics explicit, the surveillance problem-solving ontology helps BioSTORM^{A} to identify suitable methods for a specific subtask in overall task decomposition.

BioSTORM^{A}'s PSM library includes generic, domain-independent statistical methods that analyze data as single or multiple time series, and knowledge-based methods that relate abnormalities to knowledge about reportable diseases. We categorize our library's PSMs by the tasks that they perform, by the data types on which they operate, and by the types of signals that they can detect. Making such knowledge explicit facilitates system modification to enhance method portability and reuse, and helps public health professionals and other potential users to understand how the system operates.

3.3. Software development: Integrating surveillance data

The template data-source ontology provides a mechanism for describing external data and data elements in a way that allows surveillance methods to process the data. The surveillance PSMs that operate on these data may have varying input requirements. These input requirements are independent of particular data sources so that developers may reuse the PSMs to automate different analytic tasks in a flexible fashion. Each PSM in our library adheres to BioSTORM^{A}'s declarative method ontology, enabling the system to know the data types of the inputs that each PSM can process. This explicit information, combined with additional information about the external data sources available to the data-source ontology, allowed us to devise a uniform mechanism for mediating data from multiple sources to various methods at runtime (Pincus & Musen, 2003). Our approach involves the use of two specialized components called a *data broker* and a *mapping interpreter*. These components reconcile syntactic and semantic differences between the incoming data streams and the data expectations of the PSMs. Together, the data broker and the mapping interpreter provide a semantic bridge between analytic PSMs and raw data. Our approach enables BioSTORM^{A} to make meaningful computations over disparate types of data without the need for major reprogramming whenever we incorporate a new data source or develop a new PSM for the library.

3.3.1. Data broker—The data broker uses the template data-source ontology to allow PSMs to read data from many diverse external sources at run time. It queries the ontology to retrieve a data source's description and constructs a stream of uniform data objects from the raw input data. First, the data broker accesses and retrieves data in the original location based on the ontology's description of the low-level data classes. Second, the broker formats the data and groups them in a canonical structure specified by the data-source ontology. It packages the data with appropriate context annotations to create syntactically uniform and semantically unambiguous data objects. The data objects are then ready for the PSMs that operate on them. In this way each PSM receives a customized set of data objects and is insulated from idiosyncrasies in the format of the raw data.

3.3.2. Mapping interpreter—Some PSMs can operate directly on data transmitted from the data broker. However, many surveillance methods in our library expect data in a syntax, structure, or level of granularity that is different from the data broker's lower level data objects. In these cases, BioSTORM^{A} must supply data to the PSMs in the appropriate representation. The system therefore requires a means to transform the data output by the data broker and to map those transformations to the inputs of designated PSMs. We developed a *mapping interpreter* that uses an ontology of transformation types or *mapping relations* to transmute data sources and data elements to match the data requirements of different PSMs. For each data group in the data-source ontology, there are specific mapping relations that define the transformation of canonical data elements into the runtime inputs of PSMs. These transformations range from simply renaming the atomic values of data elements to terms that a PSM happens to expect to composing lexical or functional expressions of data elements to match the type of complex input data assumed by the target PSM. For example, when configuring a surveillance PSM to aggregate different data streams, we created a set of mappings to transform continuously sampled data-stream content into discrete individual events, as required by the aggregation PSM. Based on a set of data–method mapping relations that we created in the mapping ontology, BioSTORM^{A} translates incoming data elements to a set of input data instances for the particular PSM to use. The mapping interpreter performs this task by processing mapping relations for each input data group and each target PSM, and thereby generating streams of canonical data structures processable by the PSMs.

3.4. Software development: Deploying PSMs

We developed a deployment controller to invoke and manage BioSTORM^{A}'s PSMs (Buckeridge, O'Connor, et al., 2004). The deployment controller activates the PSMs to conduct surveillance and coordinates data flow from the data sources to appropriate PSMs via the data broker and mapping interpreter. It uses the data source and surveillance PSM ontologies that describe the surveillance data and the analytic methods to configure and deploy system components. The deployment controller ensures that the data broker moves the correct data at the correct time. The entire process must execute efficiently, as many data sources may be sent to various PSM configurations operating in parallel. We implemented the deployment controller using the JavaSpaces rendering (Halter, 2002) of the Linda coordination language (Carriero & Gelertner, 1989). Linda was designed to enable users to create parallel programs. It is based on logically global, associative object memory space called a *tuple space*. This tuple space is analogous to a blackboard. Linda provides interprocess communication and synchronization facilities via the insertion and removal of tuples from the tuple space. It effectively implements parallelism with a small number of simple operations on the tuple space to create and coordinate parallel processes. The tuple space looks like a single global memory space to component processes. JavaSpaces Linda implementation thus provides a distributed parallel mechanism for PSM communication (see Fig. 4).

Our approach allows BioSTORM^{A} to use the Linda model's tremendous efficiencies while enabling it to scale up to large data sets. Our solution also allows PSMs to exchange data using high-level terms provided by the data-source ontology, freeing PSMs from low-level data formatting concerns. When PSMs require data in a form that is not provided by the data-source ontology, the controller invokes the mapping interpreter, which uses the appropriate mapping ontology instance to perform the required tailoring. The controller thus provides a coherent, efficient runtime system that unifies data sources, knowledge bases, and PSMs. It is the basis of BioSTORM^{A}'s ability to support modular systems, concurrent applications, and structured evaluation of multiple knowledge-based analytic methods encoded as PSMs. Figure 5 shows a screenshot of a graphical monitoring tool displaying an example deployment of a set of temporal PSMs. This tool monitors all tuple space activity allowing it to display interactions among systems components.

3.5. BioSTORM^{A} system summary

The BioSTORM^{A} system provides a computational framework to meet performance and flexibility demands of emerging disease and bioterrorism surveillance systems. It demonstrates an end-to-end solution to many problems associated with data acquisition, integration, and analysis for public health surveillance. It leverages long-standing work in AI concerning the use of ontologies for semantic integration, deployment of reusable PSMs, and mapping of PSMs to domain. The system demonstrates how these methods can support both data integration and the rapid configuration of PSMs for analyzing large volumes of disparate, noisy data.

The main contribution of BioSTORM^{A} is a scalable, uniform method for feeding heterogenous, real-time data sources to a coherent assembly of reusable PSMs. This feature of the system also exposes its greatest weakness: the engineering overhead associated with creating the ontologies required for integrating diverse data sources. Whereas the data-source ontology provides a convenient template to support this integration process, the manual operation of creating the necessary data descriptions is cumbersome, time-consuming, and requires considerable domain expertise and familiarity with the peculiarities of each data source. Once the data are described using the data-source ontology, however, using the mapping interpreter to generate custom mappings for individual PSMs is relatively straightforward. Nevertheless, the lack of robust tools to support data integration remains a challenge. An additional limitation of this implementation is the relative simplicity of the surveillance PSM ontology. This ontology did not easily support the creation of nuanced algorithms that can use deep decompositions of PSMs, so only relatively simple algorithms could be modeled in the system. Our second implementation of BioSTORM addressed this limitation head on.

4. SECOND BioSTORM IMPLEMENTATION: EMPHASIZING INTERACTION AMONG PSMs

The last decade has seen the introduction of aberrancy-detection algorithms for screening large volumes of time-ordered data. Some of these algorithms were developed specifically for surveillance, although most were adapted from other fields, such as industrial process control and cancer epidemiology. Theoretical considerations and empirical results suggest that algorithm accuracy and timeliness in detecting disease outbreaks are quite variable. For example, differences in performance have occurred when two algorithms are applied to the same data, and when the same algorithm is applied to different data sets. Although variations in algorithm performance are acknowledged in general, specific descriptions of how algorithms differ in practice are fragmented and difficult to interpret, generally because of the absence of a conceptual framework to support their consistent description and the lack

of mechanisms to synthesize the reasons for variations. This lack of focus on critically modeling these algorithms hinders research, leads to confusion and variation in surveillance practice, and threatens the return on the considerable investment in surveillance systems and aberrancy-detection methods.

BioSTORM^{B} focused on the need to identify specific reasons for these performance variations consistently by developing an explicit model of aberrancy-detection algorithms and by developing a software infrastructure for rapidly conducting studies using elaborate PSM configurations. The BioSTORM^{B} model aimed to clarify common features and meaningful distinctions among different algorithms and provides an extensible framework for gathering evidence about relative performance. Our objective was to identify fundamental reasons for differences in algorithm performance. We also aimed to enable consistent reporting of results from studies, synthesis of results across different studies, and generation of empirical evidence to support surveillance practice and research. Accordingly, the BioSTORM^{B} system allows users to configure and evaluate alternative aberrancy-detection algorithms in a systematic manner. The PSM deployment infrastructure of BioSTORM^{A} could not support these studies. Its relatively simple analyses used few PSMs and these analyses were time-consuming to construct. This shortcoming was a result of the system's emphasis on integrating domain data with PSMs. BioSTORM^{B} required far more flexible PSM configuration and deployment software. It required software that would easily support the deployment of a large number of studies, each of which could require a variety of PSMs. The emphasis was thus shifted from PSM–data interaction to interactions between the PSMs themselves.

4.1. Model development: Using the task-analytic methodology to model surveillance algorithms

The task–analytic methodology has many desirable features for modeling surveillance algorithms. In this section, we describe how we have drawn on this methodology to extend our earlier work in modeling surveillance methods and how we have created an explicit representation of aberrancy-detection algorithms. We have focused our modeling efforts on a subset of algorithms developed for public health surveillance, specifically temporal aberrancy-detection algorithms. Our model, however, is extensible, and can include other types of algorithms as well. At the highest level of decomposition, our model identifies the subtasks involved in aberrancy detection and a number of methods described in the literature that are suitable for accomplishing these subtasks. In addition, the model identifies concepts related to data flow between subtasks and the properties of each method that may influence detection performance. Most importantly, it makes explicit the structural similarities and differences that may have implications for algorithm performance. It also clarifies the roles of different PSMs in the overall aberrancy-detection process and identifies when one PSM can be used for multiple tasks.

Our model of aberrancy-detection algorithms includes three major parts. First, it identifies the hierarchical task structure of the aberrancy-detection process and shows how it applies to individual algorithms used in public health surveillance. Second, it augments this representation with descriptors related to control and data flow, which we use to specify subtask ordering, iteration, and input–output relations among subtasks. Third, the model identifies properties that characterize the individual PSMs used to accomplish the subtasks.

The surveillance literature consistently describes temporal aberrancy-detection algorithms as procedures that evaluate the incidence of health events sequentially, in terms of differences between observed incidence and normal historical incidence. In task-analytic terms, these algorithms can be viewed as instances of a single task–decomposition method, which performs the task of detecting aberrations in surveillance data. We call this task–

decomposition method the *temporal aberrancy detection* method. Aberrancy detection entails at least three key subtasks: *compute expectation* (typically derived from historical observations), *obtain current observation* of the health event, and *compute test value* (significance of the current value's deviation from expectations). These steps are subtasks of the top-level aberrancy-detection task as performed by the *temporal aberrancy detection* method. We use one more subtask, *evaluate test value*, because many algorithms perform transformations on current observations to compute a test value or detection statistic (i.e., cumulative summation, moving average) instead of using raw data directly. Figure 6 displays methods eligible to perform each subtask.

Temporal algorithms are represented as instances of a task–decomposition method, *aberrancy detection (temporal)*, that performs the task of detecting aberrations in the surveillance data by decomposing this task into four subtasks (ellipses). Each subtask can be accomplished by different methods (rectangles), some of which perform the task directly (primitive methods, shown as dark rectangles), and some further decompose the task into subtasks (task decomposition methods, shown as light rectangles). For instance, the *compute expectation* task, which constitutes one of the steps (subtasks) of aberrancy detection, can, in turn, be decomposed into four subtasks, if an *empirical forecasting* method is used. Alternatively, this task can be accomplished directly by a primitive method: *theory-based forecasting*. Similar alternatives exist for the *evaluate test value* task.

Obtain current observation is a simple and straightforward subtask that is typically performed by querying a database. For the three other subtasks, multiple methods exist. Some are primitive, whereas the others are complex. For example, the *evaluate test value* task, which is responsible for generating outbreak alarms, can be accomplished by comparing an observed value to an expected value using a predefined threshold (via a primitive method known as Binary Alarm), or by passing the residual of such a comparison through a control chart (via a complex method known as Residual Based). The *compute expectation* subtask and the *evaluate test value* subtask can be further decomposed into constituents. To represent any particular aberrancy-detection algorithm, a single method must be selected for each subtask from the list of eligible methods, producing a task–decomposition hierarchy. The differences in the PSMs that may be used to automate particular subtasks allow developers to model the observed variety in existing aberrancy-detection algorithms.

4.1.1. Modeling individual methods—We also encoded additional knowledge about individual PSMs. One important property of a PSM is its role in the overall aberrancy-detection process, which can be inferred directly from the location of the method in the task–decomposition structure. Although some methods are relatively specialized, others may be used for several different tasks. An example of such a reusable method is the exponentially weighted moving average method, which may function either as a forecasting technique or as a means to transform raw observations into test values.

Another group of PSM properties is related to the data and objects on which the PSM operates include data processed, the results of computations performed by the method, and, sometimes, internal state information. For example, the primitive method cumulative sum takes a new observation as an incoming data object. It computes the values of upper and lower running sums using the observation and an internally stored sum from previous invocations of the PSM.

Configuration properties, or parameters, control exactly how a method should perform a task. These properties can characterize both primitive PSMs and task–decomposition PSMs, but obviously, they apply only to methods allowing variations in their internal procedures.

Finally, a number of nonfunctional properties can be useful in describing methods, such as their performance characteristics (e.g., time and space requirements) or any helpful meta-information (Gennari et al., 1994).

The task–decomposition hierarchy in Figure 6 is not a complete representation of our model of aberrancy detection, as it does not specify the ordering of subtasks or the details of data flow. Our model also includes encodings of these control flow and data flow characteristics. These encodings are applicable only to task–decomposition methods. Although primitive methods also have internal algorithmic flow, there is no need to encode these elements explicitly in our model.

4.1.2. Modeling control flow—A key element in representing flow inside task–method decompositions is an algorithm, which is a collection of tasks and control elements as a directed graph. For the purpose of modeling aberrancy detection, we distinguish between two types of nodes in an algorithm graph: subtasks and iterations. These nodes are shown in a task–decomposition tree (see Fig. 6). Iteration represents control flow when a set of subtasks in a task–method decomposition should be repeated multiple times. For example, one sequence of computations and alerting decisions is usually repeated daily in a surveillance time series. A sequence of subtasks inside an iteration generates a set of intermediate results that are aggregated with intermediate results from other repetitions of the same subtasks. Figure 7 illustrates the relationships among algorithms, tasks, iterations, and methods.

The arcs of a graph are called connectors. Connectors represent data flow through tasks and iterations. Specifically, a connector between two nodes indicates that the predecessor node's output is used by the successor node as input. Although not intended to model control flow directly, connectors represent data dependencies between tasks and iteration units. In this sense, they provide information necessary for inferring control flow knowledge from a graph.

4.2. Software development: Deploying a study

In BioSTORM^{B} we used the Web Ontology Language (OWL, 2004) to encode our model of aberrancy-detection algorithms. The ontology defines a typology of the subtasks and candidate PSMs as a hierarchy of classes, denotes potentially appropriate methods for each subtask, describes salient characteristics of individual PSMs as declarative method properties, and includes properties that encode control flow and data flow relationships among the tasks. To integrate aberrancy-detection algorithms consonant with this model with other operational components of the BioSTORM^{B} system, we extended our representation to include elements that describe properties of surveillance data and the configuration of study.

To support the significant processing requirements of running studies, we implemented an agent-based software system that can support complex deployments of methods and that can handle large quantities of data. We developed this parallelized version of BioSTORM^{B} with the Java Agent Development Framework (JADE; Bellifemine et al., 2007), an open-source platform for building distributed applications. The basic distribution unit in JADE is an *agent*, a stand-alone module that performs a particular job. Agents are assigned to a *platform*, which is a logical space that can be distributed across machines. Agents communicate with one another by passing messages.

We defined the subtasks in aberrancy-detection algorithms as JADE agents (Fig. 8). These subtask agents were configured to use a particular implementation of a PSM to perform their actions. BioSTORM^{B} provides a problem solving method application programming

interface (API) in Java that the generated task agents use to interact with their corresponding PSMs (Nyulas et al., 2008). Using this API, we have built a Surveillance Method Library as an extensible Java package containing a set of surveillance PSMs. This library uses the R statistical software package (R, 2009; rJava, 2009). R is a powerful open-source environment for statistical computing that provides a variety of analytical techniques and includes over a thousand contributed packages. R features implementations of most of the temporal methods we wish to incorporate into BioSTORM^{B}, and is thus used for any nontrivial statistical computations performed by our methods. Simpler methods are encoded directly in Java.

Three other important components of the system are also implemented as agents. A *controller agent* deploys an aberrancy-detection algorithm by creating and configuring all the relevant subtask agents. A *configurator agent* reads the configuration of an evaluation analysis from the ontology and sends necessary initial configuration information to the controller agent. A *blackboard agent* implements a blackboard; it is the only agent known to all other agents. Subtask agents send messages containing data reading and writing requests only to the blackboard agent and are therefore insulated from having to know about other subtask agents, the origin of their input data, and the ultimate destination of their outputs.

4.3. BioSTORM^{B} system summary

BioSTORM^{B} extended BioSTORM^{A}'s computational framework to provide a more elaborate, scalable, and easily configurable deployment mechanism for performing surveillance analyses and evaluating aberrancy detection performance. The emphasis shifted from the data-integration strategies of BioSTORM^{A} and instead focused on the development of a robust software infrastructure for deploying the PSMs to execute these studies. Its architecture offers a flexible framework into which developers can drop new PSMs and new data sources, and then measure system performance. Other surveillance systems, such as RODS (Tsui et al., 2003) and ESSENCE (Lombardo et al., 2003), do not support an open architecture, and adding new methods or data sources requires custom programming, which is a significant obstacle to experimentation. The ability to experiment with multiple PSMs applied to different data sources is crucial for the next generation of surveillance systems, and was a major motivation for our development of BioSTORM^{B}.

Our goal of fully describing all the parameters of real-world surveillance studies in BioSTORM^{B} was ambitious. We believe that our implementation served to further highlight both the strengths and weaknesses of using PSMs to develop software systems. Its use of PSMs and, in particular, the task–method decomposition approach proved invaluable in managing the complexity of BioSTORM^{B}'s design and implementation. Indeed, it is difficult to imagine suitable alternate approaches to manage this process. The downside is that, as in the development of BioSTORM^{A}, we needed to encode a significant amount of novel, nontrivial software components and ontologies for managing PSM deployment. This software engineering was in addition to the development efforts required for BioSTORM^{A}, which centered primarily on integrating PSM with domain data.

5. SOFTWARE REQUIRED FOR PSM-BASED SYSTEMS

Our workon the different versions of BioSTORM was conceived as an opportunity to validate our approach to knowledge-system architecture in a complex domain where the decomposition of the surveillance task into a hierarchy of problem solving components offered a particularly cogent solution. The conceptual framework provided by the PSM approach, particularly the task–method decomposition process, proved invaluable when developing these systems, providing powerful building blocks for managing a significant amount of complexity. Both versions of BioSTORM had ambitious architectural goals,

aiming to provide end-to-end support for configuration and deployment of real-world surveillance studies. The goal was to drive the specification and execution studies solely from system models encoded in ontologies and to require little or no manual programming steps. To meet this goal, we developed a variety of specialized software components and ontologies. Although some of this development effort centered on surveillance-specific functionality, considerable resources were necessary to build tools and ontologies to support the PSM approach. There are essentially no off-the-shelf tools for developing PSM-based software, so we had to create components to support all system-development stages.

Both implementations of BioSTORM were written using a mixture of Java-based, open-source components and a suite of ontology-development tools based on the Protégé ontology-development environment (Gennari et al., 2003). In BioSTORM^{A}, data-analysis tasks were straightforward and used relatively simple PSMs that needed to interoperate with a large variety of data sources. As a consequence, data mapping was central, and our development efforts concentrated on building mapping tools to support mapping domain models to PSM inputs. BioSTORM^{B} focused on more elaborate analyses using a higher number of PSMs and required more complex interactions between PSMs, so its PSM configuration and deployment components were considerably more complex. The work to build BioSTORM served to identify a core set of required ontologies and tools that are necessary when developing a PSM-based system. These include a data-source ontology, data-integration software, PSM configuration ontologies, PSM-deployment software, and inter-PSM communication software.

5.1. Data-source ontologies

The information used by PSMs is typically described by data-source models, which model all relevant information in the inputs that PSMs use. In BioSTORM^{A} we developed a rich data-source ontology to describe the possible sources of information that could be encountered by surveillance systems. Although we created this ontology to be as generic as possible, we needed to tailor the descriptions of data streams to the surveillance domain. This process typically requires extensive familiarity of both domain knowledge and data sources. Most of our development effort centered around defining these customizations. The lack of robust tools to support this customization is a source of difficulty. Most nontrivial PSM-based systems will require significant domain- and data-specific refinements of this type.

5.2. Data-integration software

In a PSM-based system, the abstract inputs of PSMs must be mapped to actual domain data. Data-integration software is thus necessary to supply streams of data in an appropriate format to system PSMs. This mapping process requires integrating individual PSMs with those data streams. In addition to constructing detailed domain models during the development phase, this process at run time typically requires preprocessing of source data, annotating those data with metadata describing their content, and transforming the data into a form that can be reconciled with the domain model. We developed the mapping interpreter for BioSTORM^{A} to address the data transformation task (Fig. 4). The mapping interpreter takes as input the data source models and instances of those data sources and produces as output integrated data streams. It supports a large array of transformations to allow integration of a wide variety of data sources.

Mapping between the generic data and knowledge requirements of PSMs and specific domain ontologies remains a particularly challenging task. There is a pressing need for end-to-end tools that support the interactive exploration of the data to be integrated, the annotation of those data, and the detailed specification of a range of possible data

transformations. There also is a need for software libraries to invoke these real-time data transformations from PSMs. The software that we developed for BioSTORM[A] to aid system builders in mapping source data to PSMs supported some of these activities, but it would be hard to argue that it provides a robust solution (Crubézy & Musen, 2004). We finessed the data-to-PSM mapping problem in BioSTORM[B]: instead of allowing mappings from arbitrary domain models to arbitrary PSMs, we designed a simple input specification for PSMs that had to be satisfied by the source data. We effectively bypassed the mapping problem by developing a stripped-down method ontology that did not support mappings from arbitrary domain ontologies. Instead, users are required to transform their data manually to conform with the method ontology. The task of mapping domain concepts and data to PSM inputs was thus effectively left to the developer. This shift was partly a result of our less ambitious data-integration goals for BioSTORM[B] but also reflected a desire to avoid the anticipated large amount of effort needed to implement a comprehensive mapping solution. Nevertheless, the general problem of mapping entities described by domain ontologies to the generic data types on which PSMs operate has to be addressed to facilitate the wider adoption of PSM-based systems, as the reconciliation between PSMs and domain models is a central requirement.

5.3. PSM configuration ontologies

The task–method decomposition approach adopted by Bio-STORM provided a structuring mechanism for describing the arrangement of PSMs in the system. We developed several ontologies that provided detailed computer-interpretable descriptions of all PSMs in a system and their interrelationships. The model development process benefited from the availability of robust ontology-authoring tools. In particular, the Protégé tool and associated plugins were invaluable. However, the level of detail required to fully specify all aspects of an ontology-driven application requires considerable ontology development and encoding efforts. Also, because these ontologies directly control all components in the system, iterating through different versions of a design may require parallel changes in both software and ontologies. For example, if an ontology is significantly modified to reflect a new design emphasis, the associated software component may have to be re-implemented as well. When the design stabilizes, the system should be reconfigurable by ontology modifications only, but until that point is reached the parallel development of software and ontologies can be a challenge. In BioSTORM[B] at least, the benefits of this approach outweigh the initial specification overhead because of the ultimate ease of configuration and reconfiguration of PSMs. The surveillance studies carried out using BioSTORM[B] required the deployments of a large number of PSM configurations and hand-configuring each study manually would have been prohibitively expensive. The use of a configuration ontology to specify studies allowed new studies to be deployed with minor modifications to existing configurations.

5.4. PSM deployment software

Deployment software must take a configuration model and a set of PSM implementations and deploy the PSMs to produce a running system. Although there are many tools that help developers to author ontologies, there are few widely adopted techniques or tools for constructing component-based applications that use ontologies. Furthermore, existing tools do not typically provide the fundamental infrastructure components necessary to construct ontology-oriented systems. There are no ontology-based equivalents of CORBA or J2EE, for example. Hence, developers are on their own when implementing their systems. In many cases, the lack of tools is a reflection of the modest goals for use on ontologies in many contemporary knowledge-based systems: many ontologies are developed to formalize basic terms and concepts in specific parts of a domain; there is often little or no effort to design ontologies to be usable with external analytic methods. As a result, there has not yet been significant industrial effort to develop software environments for building large-scale

ontology-based applications. UPML (Fensel et al., 2002) attempted to provide a high-level markup for consistently describing interfaces to components in a PSM-based system, but there are no known implementations of UPML-based tools. Thus, in both versions of BioSTORM we developed custom deployment software. This software included a number of interactive tools to support PSM configuration and deployment and real-time system debugging. Developing the configuration and deployment software involved challenges common to all component-based systems. However, the process was complicated by the need to develop custom middleware for many tasks. The configuration software for the JavaSpaces and JADE systems and the inter-PSM communication process were developed from scratch.

5.5. Inter-PSM communication software

Communication software is required to control the flow of information among PSMs in a deployed system. Such software must ensure that all components are delivered their requisite data. Efficiently routing data from secondary storage to possibly distributed PSMs in a deployed system is important, particularly if scalability is a concern. Because data integration and mapping may add significant overhead, streaming software may have to adopt specialized optimization techniques. Mapping adds a layer of indirection, which also may be a problem with large data sets. We tackled this problem in BioSTORM^{A} by developing a novel mapping software to optimize data access (Buckeridge, Burkom, et al., 2004). BioSTORM^{B} used a simple data annotation ontology that could be custom-tailored by system developers, and that supported direct (and thus more efficient) data access. Irrespective of the approach adopted, efficient data access is essential to ensure scalability.

Both versions of BioSTORM required the development or extension five nontrivial software components and ontologies. In particular, as described above, the data-integration and mapping components in BioSTORM^{A} demanded major development resources. In the construction phase, for example, mapping an arbitrary domain ontology to match the input–output specifications of a decomposition of tasks in a typical PSM can be time consuming, labor intensive, and error prone. Robust semiautomated mapping tools to support this process could make it more tractable, but current tools are not flexible enough. This shortcoming also occurs in run-time PSM deployment. After specification, PSMs must be deployed in a running system, and the data on which they operate must be supplied as required. Again, there is a lack of software tools to automate this process. There are also efficiency considerations if systems have to deal with a lot of data, because of levels of indirection created by the domain mapping process. It is worth noting that the BioSTORM development team has an ideal software-engineering environment, given both the availability of experienced, knowledgeable practitioners, and access to a variety of tools based on PSM principles. Despite these resources, we faced significant challenges in our work. The corresponding effort and the initial learning curve may be greater elsewhere.

6. DISCUSSION

PSMs were developed by the artificial intelligence community as a technique to model procedural knowledge within intelligent systems. Despite considerable early promise, PSM-based techniques have not achieved widespread adoption. Various reasons have been proposed for this low adoption rate. Chandrasekaran (2009, this issue) argues that the approach fails to consider cognitive processes, and thus provides an incomplete picture of the problem solving task. He argues that there is a poor match between the techniques provided by PSMs and the way in which human cognition operates. Clancey et al. (2009, this issue) assert that the PSM approach fails to consider the workflow processes in system design. He claims that the approach assumes that PSMs are embedded in relatively static

systems and that, as a result, the systems that they automate are not sufficiently flexible to address the situated behavior of many computational systems.

Although both perspectives are valid, our view of PSMs as software-engineering building blocks is more pragmatic. We do not suggest that PSMs offer a general theory of intelligent problem solving. Rather, we view the use of PSMs as an approach to manage complexity in the design and implementation of software systems. To argue that PSMs do not model true cognitive behavior or the nuances of workflow is to miss their contribution to the engineering of software systems. Research on PSMs has provided tractable methods for human engineers to decompose complex problems. Although there is great value in a theory of intelligence that can be tied to computational mechanisms that would allow PSMs to operate as situated agents, such approaches are not required for many systems. Not all computational artifacts require this level of generality and not all PSMs need to be agents in the world. BioSTORM, for example, provides an interactive laboratory for experimenting with surveillance algorithms. It does not require agents that are situated in the real world. In BioSTORM, it is the interaction among PSMs that provides the greatest challenges, not the system's interaction with the environment. Workflow, for example, is outside of the system's scope.

The PSM approach provides a powerful set of techniques that can give software engineers a handle on managing complexity. Our work on BioSTORM and other systems shows how PSMs can assist developers in managing that complexity. In the BioSTORM system, PSMs allowed us to do what others have not attempted to do in designing systems for syndromic surveillance: modeling the intricate problem solving elements needed to define surveillance algorithms. Other surveillance systems are essentially monolithic black boxes. They do not provide systematic ways of updating algorithms or of measuring the effects of substituting one problem solving component for another. These shortcomings have consequences that extend beyond the obvious difficulties of system development. The "black box" approach makes it difficult to carry out experiments to identify and categorize the subtasks required to execute surveillance algorithms, which limits advances in the field. We believe that BioSTORM can make fundamental contributions to epidemiology and public health by providing a laboratory environment for evaluating aberrancy-detection algorithms.

We concur with Domingue and Fensel (2009, this issue) that PSMs provide a useful approach for dealing with the new demands of building intelligent applications for the Web. Our experiences developing several systems, however, makes us concerned about the practicalities and scalability of this approach without sufficiently robust tools for managing the software-engineering process. The development of both versions of BioSTORM has convinced us that there are significant software-engineering and modeling difficulties when building PSM-based systems. The time and effort required for end-to-end design and deployment are nontrivial. Simply put, the complexity of modeling domain data sources in BioSTORM^{A} and the complexity of modeling the characteristics that determined appropriate selection of PSMs in BioSTORM^{B} could not be anticipated from an initial analysis of the syndromic surveillance domain. It was not until we began to model the details of the underlying systems in each case that the consequent engineering effort became apparent. A systems designer faced with the task of creating any computer program from reusable software components must determine whether the effort is worthwhile for a single system or even if it is worthwhile when amortized over multiple systems. With current tools and methodologies, it is extremely difficult to measure the required effort in advance. In the case of the BioSTORM project, however, the benefits of the PSM-based approach coupled with the extensive use of ontologies proved advantageous. The PSM-based approach allowed us to construct and reconfigure alternative surveillance algorithms rapidly. More important, we found no other tools that would have allowed us to develop sufficiently

detailed models to support the description and automated deployment of these algorithms. Thus, it is unclear how we could have implemented either version of BioSTORM without the architectural clarity that the use of PSMs and explicit ontologies offer.

The emergence of component-based systems may provide an impetus for tool development that will assist in the construction of intelligent systems. In particular, there are many parallels between the PSM-based approach and the approach used in developing component-based software, both of which have similar goals of software reuse. A large number of component-based environments have been developed, including CORBA, J2EE, .NET, and Eclipse. The Object Management Group is also developing domain standards in various industries such as health care, transportation, and life sciences research. However, the level of abstraction for describing components in current systems is usually very low. Components are primarily characterized at the interface level, with minimal encoding of higher level information. As a result, there is little automation in system assembly. Typically, developers piece together components manually and write "glue code" to integrate the various components. Nevertheless, these software platforms provide a strong basis for deploying PSM-based software. In BioSTORM[B], for example, JADE's highly flexible agent deployment platform dramatically simplified the configuration, deployment, and run-time monitoring of PSMs. However, we had to develop tools to support the richer markup required for PSM-based development because JADE provides only very coarse-grained agent characterization.

The Object Management Group's model-driven architecture (IBM, 2006) approach is addressing some of these shortcomings. Instead of manually assembling and integrating components, programmers use design languages such as UML (UML, 2009) to specify high-level models of a software application, from which tools can generate system configurations automatically. This model allows easier reconfiguration and reduces the need to write glue code. The Eclipse EMF, for example, is one of the more commonly used model-driven approaches (Steinberg et al., 2008). It has had widespread industry adoption and provides a rich set of support tools. However, despite significant tool support, the models developed using this approach typically describe only structural aspects of the system. There is little or no semantic markup of component interfaces or of information exchanged among components. More important, the systems do not use composition techniques provided by PSMs.

The Semantic Web initiative may also drive the development of tools for building PSM-based systems (Domingue & Fensel, 2009, this issue). Most application development for the Web involves assembling large systems from existing components, a process that could benefit from reuse and knowledge-level analysis techniques offered by PSMs. There has been some work on using PSMs in developing Semantic Web applications. The IBROW[3] project (Benjamins et al., 1998), for example, spawned the Internet Reasoning Service (IRS), which aimed to provide a framework for component discovery and composition on the Web (Crubézy et al., 2003). The ultimate goal of the IRS project was to make PSMs available as Web resources. Later, OWL-S (Martin et al., 2004) also aimed to add semantic descriptions to Web components, although its goals were not as ambitious as those of the IRS system. In particular, the semantic markup possible with OWL-S is at a very high level of abstraction. The resultant coarse granularity is a common feature of most current Web-service descriptions: tasks are described very broadly, often in domain-specific terms, and thus the opportunities for reuse are very small. PSMs provide the opportunity to capture problem solving at a finer level of granularity than current Web services, and thus offer the possibility of software reuse on the Web. Support for these knowledge-level descriptions would also facilitate service discovery and composition. The use of knowledge-based component composition techniques on the Semantic Web is currently in its infancy,

however. PSMs offer a mechanism that could provide building blocks for this approach (Benjamins, 2008). In particular, the task-decomposition approach can be used to construct knowledge-based Web applications. Because most Web applications are component-based, there should be a natural fit. The increased adoption of OWL as the Semantic Web's standard ontology language may ease development effort and interoperability requirements of PSM-based tools.

These developments present new opportunities to reduce the engineering costs of building PSM-based systems. As we have shown in this paper, the fundamental PSM construction techniques used when developing new applications are sufficiently powerful and flexible to build very complex systems. The problem lies in the practical difficulty of using these techniques because of poor tool support. We believe that a renewed effort to construct tools specifically enabling this process will lower costs and will allow the leveraging of the considerable power of the PSM-based approach to support the building of knowledge-based technologies, both on the Web and in conventional software.

Acknowledgments

This research was supported by Grant HK000019 from the Centers for Disease Control and Prevention under the BioSense Initiative to improve early event detection and by a contract supported by the Defense Advanced Research Projects Agency. Additional assistance was provided by the Protégé Resource supported by the National Library of Medicine under Grant LM007885. David Buckeridge is supported by a Canada Research Chair in Public Health Informatics. The authors gratefully acknowledge the contributions of Zachary Pincus and other former members of the BioSTORM team. The paper also benefited from discussions with William Grosso. We thank Valerie Natale for her valuable editorial advice. We also thank David C. Brown for organizing the *AI EDAM* Special Issue on PSMs and for his considerable patience throughout the review process.

References

Bellifemine, FL.; Caire, G.; Greenwood, D. Developing Multi-Agent Systems with JADE. Hoboken, NJ: Wiley; 2007.

Benjamins R. Near term prospects for semantic technologies. IEEE Intelligent Systems. 2008; 23(1): 76–88.

Benjamins, R.; Decker, S.; Fensel, D.; Motta, E.; Schreiber, G.; Studer, R.; Wielinga, B. IBROW3—An intelligent brokering service for knowledge-component reuse on the World Wide Web. Proc. 11th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop; 1998.

Buckeridge DL, Okhmatovskaia O, Tu SW, O'Connor MJ, Nyulas CI, Musen MA. Understanding detection performance in public health surveillance: modeling aberrancy-detection algorithms. Journal of the American Medical Informatics Association. 2008; 15(6):760–769. [PubMed: 18755992]

Buckeridge DL, Burkom H, Moore AW, Pavlin JA, Cutchis PN, Hogan W. Evaluation of syndromic surveillance systems: development of an epidemic simulation model. Syndromic Surveillance: Reports from a National Conference. Morbidity and Mortality Weekly Report. 2004; 53(Suppl): 137–143.

Buckeridge, DL.; Graham, J.; O'Connor, MJ.; Choy, MK.; Tu, SW.; Musen, MA. AMIA Annual Symp. Bethesda, MD: American Medical Informatics Association; 2002. Knowledge-based bioterrorism surveillance; p. 76-80.

Buckeridge, DL.; O'Connor, MJ.; Xu, H.; Musen, MA. A knowledge-based framework for deploying surveillance problem solvers. Int. Conf. Information and Knowledge Engineering (IKE '04); Las Vegas, NV. 2004.

Buehler JW, Berkelman RL, Hartley DM, Peters CJ. Syndromic surveillance and bioterrorism-related epidemics. Emerging Infectious Diseases. 2003; 9(10):1197–1204. [PubMed: 14609452]

Carriero N, Gelertner D. How to write parallel programs: a guide to the perplexed. ACM Computing Surveys. 1989; 21(3):323–357.

Chandrasekaran B. Generic tasks in knowledge-based reasoning: high-level building blocks for expert systems design. IEEE Expert. 1986; 1(3):23–30.

Chandrasekaran B. Problem solving methods and knowledge systems: a personal journey to perceptual images as knowledge. Artificial Intelligence for Engineering Design, Analysis and Manufacturing. 2009; 23(4):331–338. [this issue].

Chandrasekaran B, Johnson T, Smith JW. Task structure analysis for knowledge modeling. Communications of the ACM. 1992; 33(9):124–136.

Clancey WJ. From GUIDON to NEOMYCIN to HERACLES in twenty short lessons. AI Magazine. 1986; 7(3):40–60.

Clancey WJ, Sierhius M, Seah C. Workflow agents versus expert systems: problem solving methods in work systems design. Artificial Intelligence for Engineering Design, Analysis and Manufacturing. 2009; 23(4):357–371. [this issue].

Crubézy M, Motta E, Lu W, Musen MA. Configuring online problem-solving resources with the internet reasoning service. IEEE Intelligent Systems. 2003; 18(2):34–42.

Crubézy M, O'Connor MJ, Buckeridge DL, Pincus Z, Musen MA. Ontology-centered syndromic surveillance for bioterrorism. IEEE Intelligent Systems. 2005; 20(5):26–35.

Crubézy, M.; Musen, MA. Ontologies in support of problem solving. In: Staab, S.; Studer, R., editors. Handbook on Ontologies. Berlin: Springer; 2004. p. 321-342.

Domingue J, Fensel D. Problem solving methods in a global networked age. Artificial Intelligence for Engineering Design, Analysis and Manufacturing. 2009; 23(4):373–390. [this issue].

Eriksson, H.; Puerta, AR.; Gennari, JH.; Rothenfluh, TE.; Tu, SW.; Musen, MA. Custom-tailored development tools for knowledge-based systems. Proc. 9th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop; 1995.

Fensel, D.; Groenboom, R. Specifying knowledge-based systems with reusable components. Proc. 9th Int. Conf. Software Engineering; 1997. p. 18-20.

Fensel D, Motta E, Benjamins VR, Crubézy M, Decker S, Gaspari M, Groenboom R, Grosso W, van Harmelen F, Musen MA, Plaza E, Schreiber G, Studer R, Wielinga B. The unified problem-solving method development language UPML. Knowledge and Information Systems. 2002; 5(1): 83–131.

Gennari JH, Fergerson RW, Grosso WE, Crubézy M, Eriksson H, Noy N, Tu SW. The evolution of Protégé: an environment for knowledge-based system development. International Journal of Human–Computer Studies. 2003; 1(58):89–123.

Gennari JH, Tu SW, Rothenfluh TE, Musen MA. Mapping domains to methods in support of reuse. International Journal of Human–Computer Studies. 1994; 41:399–424.

Halter, S. JavaSpaces Example by Example. New York: Prentice Hall; 2002.

IBM. Model-driven software development. IBM Systems Journal. 2006; 45(3):449.

Knublauch, H.; Fergerson, R.; Noy, NF.; Musen, MA. The Protégé OWL plugin: an open development environment for Semantic Web applications. Third Int. Semantic Web Conf; Hiroshima. 2004. p. 229-243.

Lombardo J, Burkom H, Elbert E, Magruder S, Lewis SH, Loschen W, Sari J, Sniegoski C, Wojcik R, Pavlin J. A systems overview of the electronic surveillance system for the early notification of community-based epidemics (ESSENCE II). Journal of Urban Health. 2003; 80(2 Suppl 1):i32–i42. [PubMed: 12791777]

Marcus S, Stout J, McDermott J. VT: an expert elevator designer that uses knowledge-based backtracking. AI Magazine. 1988; 9(1):95–111.

Martin, D.; Burstein, M.; Hobbs, J.; Lassila, O.; McDermott, D.; McIlraith, S.; Narayanan, S.; Paolucci, M.; Parsia, B.; Payne, T.; Sirin, E.; Srinivasan, N.; Sycara, K. OWL-S: semantic markup for Web services. Proc W3C. 2004. Accessed at http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/

McDermott JP, Bachant J. R1 revisited: four years in the trenches. AI Magazine. 1984; 5(3):21–32.

McDonald CJ, Huff SM, Suico JG, Hill G, Leavelle D, Aller R, Forrey A, Mercer K, DeMoor G, Hook J, Williams W, Case J, Maloney P. LOINC, a universal standard for identifying laboratory observations: a 5-year update. Clinical Chemistry. 2003; 49:624–633. [PubMed: 12651816]

Musen MA, Schreiber G. Architectures for intelligent systems based on reusable components. Artificial Intelligence in Medicine. 1995; 7(3):189–199. [PubMed: 7581622]

Nyulas, CI.; O'Connor, MJ.; Tu, SW.; Okhmatovskaia, A.; Buckeridge, D.; Musen, MA. An ontology-driven framework for deploying JADE agent systems. IEEE/WIC/ACM International Conf. Intelligent Agent Technology; Sydney, Australia. 2008. p. 573-577.

OWL. 2004. Accessed at www.w3.org/TR/owl-ref

Pincus, Z.; Musen, MA. AMIA Annual Symp. Bethesda, MD: American Medical Informatics Association; 2003. Contextualizing heterogeneous data for integration and inference; p. 514-518.

R. 2009. Accessed at http://www.r-project.org in January 2009

rJava. 2009. Accessed at http://rosuda.org/rJava/ in January 2009

Rothenfluh TE, Gennari J, Eriksson H, Puerta AR, Tu SW, Musen MA. Reusable ontologies, knowledge-acquisition tools, and performance systems: PROTEGE-II solutions to Sisyphus-2. International Journal of Human–Computer Studies 44(3–4). 1995:303–332.

Schreiber, G.; Akkermans, H.; Anjewierden, A.; de Hoog, R.; Shadbolt, N.; Van de Velde, W.; Wielinga, B. Knowledge Engineering and Management: The CommonKADS Methodology. Cambridge, MA: MIT Press; 1999.

Steels L. Components of expertise. AI Magazine. 1990; 11(2):30–49.

Steinberg, D.; Budinsky, F.; Paternostro, M.; Merks, E. EMF: Eclipse Modeling Framework. 2. Reading, MA: Addison–Wesley; 2008.

Tu SW, Eriksson H, Gennari JH, Shahar Y, Musen MA. Ontology-based configuration of problem-solving methods and generation of knowledge-acquisition tools: application of PROTÉGÉ-II to protocol-based decision support. Artificial Intelligence in Medicine. 1995; 7(3):257–289. [PubMed: 7581625]

Tsui FC, Espino JU, Dato VM, Gesteland PH, Hutman J, Wagner MM. Technical description of RODS: a real-time public health surveillance system. Journal of the American Medical Informatics Association. 2003; 10(5):399–408. [PubMed: 12807803]

UML. 2009. Accessed at http://www.uml.org/ in January 2009

Biographies

Martin O'Connor received his BA and MS degrees in computer science from the University of Dublin, Ireland. Before joining Stanford Center for Biomedical Informatics Research, he worked for several years at IBM's T.J. Watson Research Center in Yorktown Heights, NY. Since coming to Stanford, he developed the Chronus II temporal query system, which is used to perform temporal queries on biomedical data. His current research centers on the development of technologies to support querying, data integration, and analytic method deployment on the Semantic Web.

Csongor Nyulas received a degree in computer science from the Technical University of Cluj-Napoca, Romania. Before joining Stanford Center for Biomedical Informatics Research, he was employed at DaimlerChrysler Research in Berlin for several years, where he worked on the development of several ontology-driven tools. Since coming to Stanford, he has implemented the primary components of the BioSTORM system using the JADE agent environment.

Samson Tu is a Senior Research Scientist at the Stanford Center for Biomedical Informatics Research, where he has worked for almost 20 years on the development of domain models and reasoning components for general knowledge-based architectures. He is recognized internationally for his seminal contributions related to automation of medical therapy planning and to the modeling of protocols and guidelines. With colleagues from major academic institutions and from industry, both in the United States and abroad, he has successively developed a number of guideline models. Mr. Tu has been elected to the American College of Medical Informatics.

David L. Buckeridge is an Assistant Professor of epidemiology and biostatistics at McGill University. He developed a simulation model for evaluating outbreak detection for his doctoral dissertation at Stanford before moving to McGill University, where he holds a Canada Research Chair in Public Health Informatics. Dr. Buckeridge is a Board Member of the International Society for Disease Surveillance and is the editor of a recently published book on the informatics of disease surveillance. He currently has funding from the CDC, the Canadian Institutes of Health Research, and other agencies to evaluate automated surveillance systems.

Anna Okhmatovskaia is a Research Associate in the Department of Epidemiology and Biostatistics of McGill University. She received BS and PhD degrees in psychology from Moscow University and an MS in computer science from the University of Southern California. She joined the Health Informatics Research Group at McGill University after working at the University of Southern California Institute for Creative Technologies, where she focused on computational modeling of emotion. Dr. Okhmatovskaia's research interests lie in applied artificial intelligence, particularly knowledge modeling.

Mark A. Musen is a Professor of medicine and computer science and Head of the Stanford Center for Biomedical Informatics Research. He has been investigating issues related to intelligent systems in biomedicine since the 1980s. His work was recognized by the American Association for Medical Systems and Informatics, which presented him with its Young Investigator Award for Research in Medical Knowledge Systems in 1989. He received an NSF Young Investigator Award in 1992 for his fundamental research in computer science. Dr. Musen was elected to the American College of Medical Informatics and to the American Society for Clinical Investigation. He is Co-Editor-in-Chief of the new journal *Applied Ontology*.

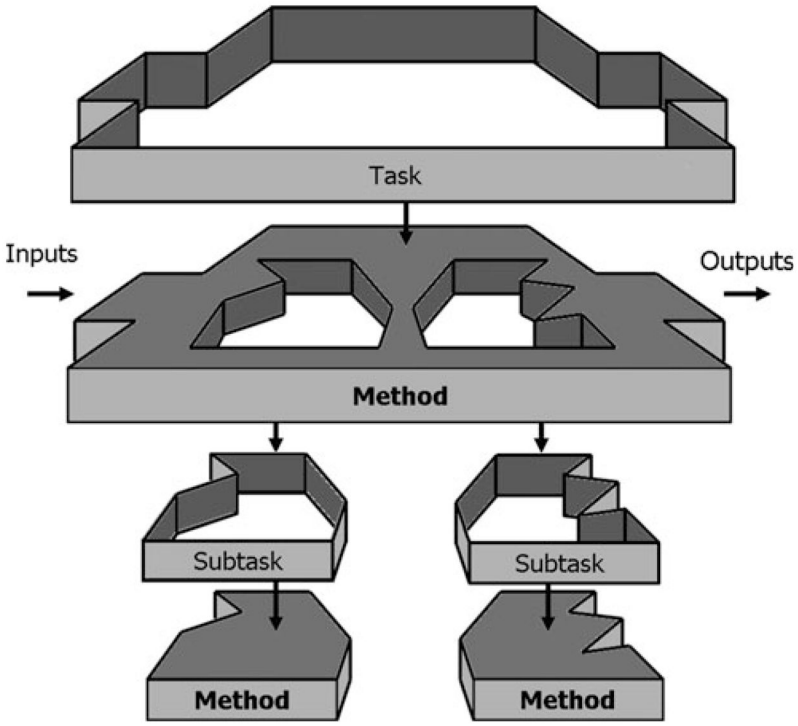

Fig. 1.
An example of task–method decomposition illustrating how task models are constructed using PSMs as building blocks.

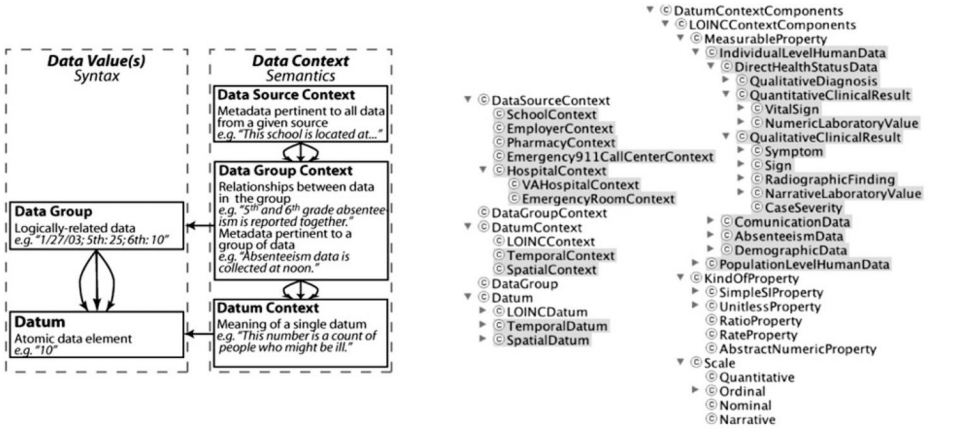

Fig. 2.

The template data-source ontology for data and metadata, customized for syndromic surveillance. (Left) Data values are associated with metadata describing the data and other relevant context. Arrows indicate one–one and many–one relationships between concepts. (Right) Highlights show additions to our template data-source ontology that are specific to syndromic surveillance. The first snapshot shows the structure of the template with our added context classes. The second snapshot shows the top levels of the taxonomy of metadata attributes, expanded with our vocabulary of "Measurable Properties," used to build LOINC objects.

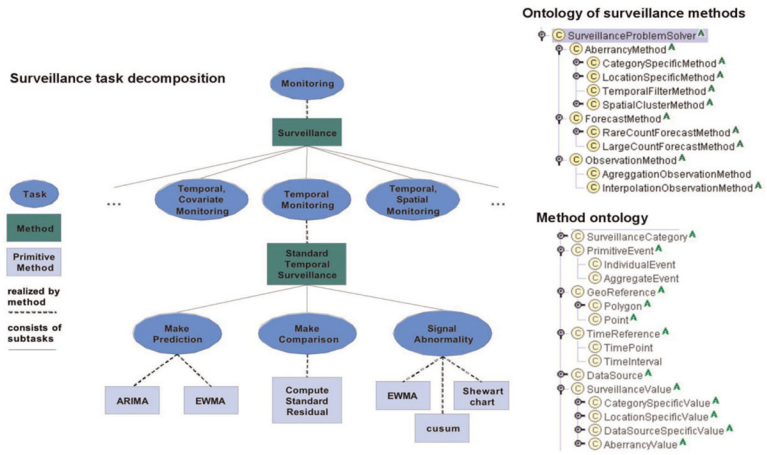

Fig. 3.

BioSTORM^{A}'s surveillance problem solving ontology. (Left) A schematic representation of the refinement of the monitoring task into subtasks defined by information scenarios in public health surveillance and the decomposition of standard temporal surveillance into further subtasks. (Right, top) A snapshot of our resulting surveillance problem solving ontology that is focused on the *ontology of surveillance methods*. (Right, bottom) Another snapshot of the problem solving ontology that is focused on the *method ontology* in which each problem solving method declares its expectations on input and output data. [A color version of this figure can be viewed online at journals.cambridge.org/aie]

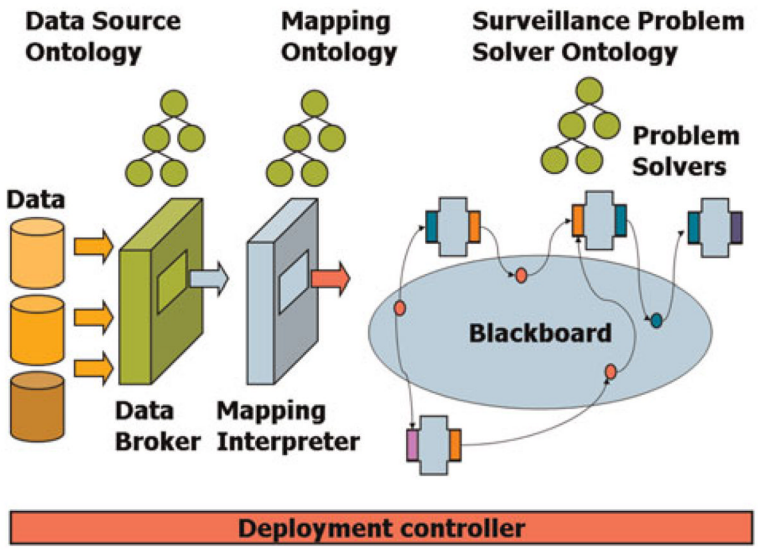

Fig. 4.

An overview of BioSTORM[A]'s architecture and components. Using a blackboard mechanism, the deployment controller orchestrates the problem solvers' deployment. It also instructs the Data Broker to process incoming data streams. These processed streams are then fed to the Mapping Interpreter, which routes them through the blackboard to the appropriate PSMs. [A color version of this figure can be viewed online at journals.cambridge.org/aie]

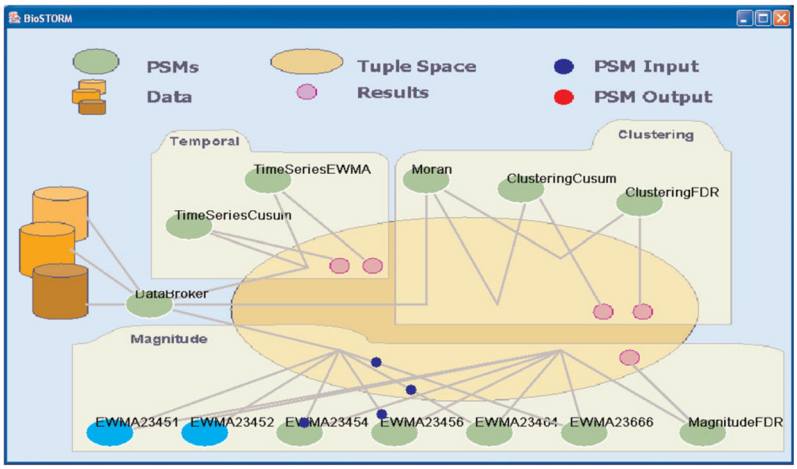

Fig. 5.

A screenshot of a BiOSTORM[A] monitoring tool showing a deployment of a set of PSMs. The tool displays groups of PSMs tackling a particular task (temporal, magnitude, and clustering in this example) and shows the communication paths between them. The tool monitors all communication between entities in a deployed system and visually displays these interactions. [A color version of this figure can be viewed online at journals.cambridge.org/aie]

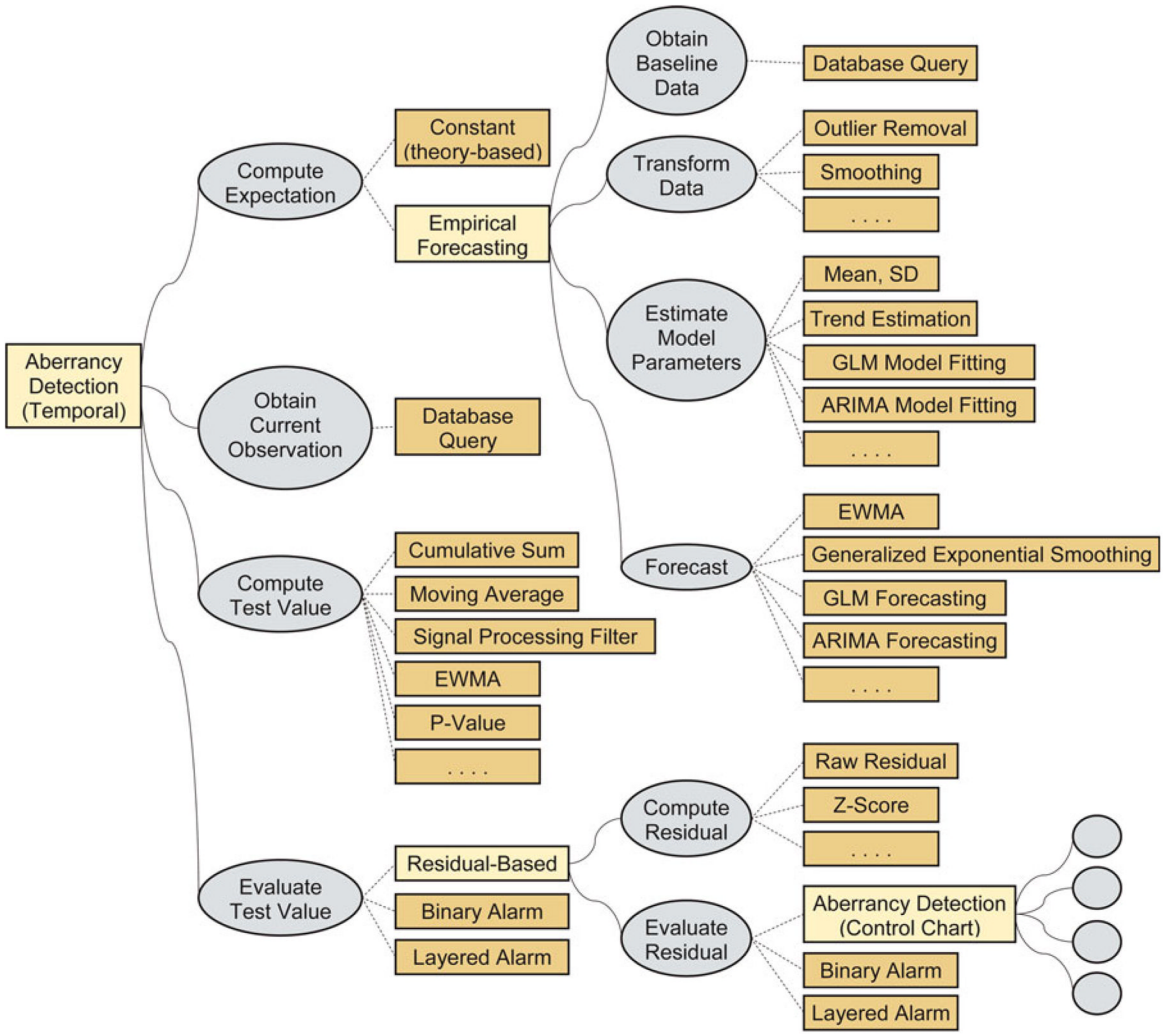

Fig. 6.

The general task structure of temporal aberrancy detection algorithms. Temporal algorithms are represented as instances of a task–decomposition method, *Aberrancy Detection(Temporal)*, that performs the task of detecting aberrations in the surveillance data by decomposing this task into four subtasks (ellipses). Each subtask can be accomplished by different methods (rectangles), some of which perform the task directly (primitive methods, dark rectangles), and some further decompose the task into subtasks (task decomposition methods, light rectangles). For instance, the *compute expectation* task, which constitutes one of the steps (subtasks) of aberrancy detection, can in turn be decomposed into four subtasks if the *empirical forecasting* method is used. [A color version of this figure can be viewed online at journals.cambridge.org/aie]

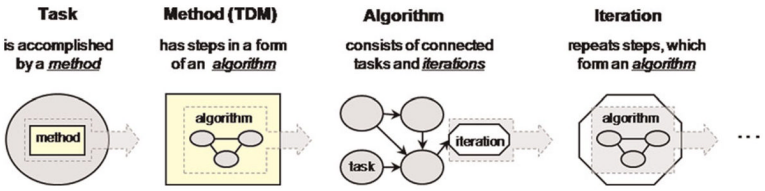

Fig. 7.

The relationships among tasks, methods, iterations, and algorithms. When a task is accomplished by a task–decomposition method (TDM), this implies that the method performs several steps in a particular order; that is, the method has an algorithm associated with it. An algorithm in turn consists of interconnected tasks (these are the subtasks of the original, higher level task) and iterations. An iteration specifies repetition of a sequence of tasks (or other, nested iterations); this sequence is also represented by an algorithm. [A color version of this figure can be viewed online at journals.cambridge.org/aie]

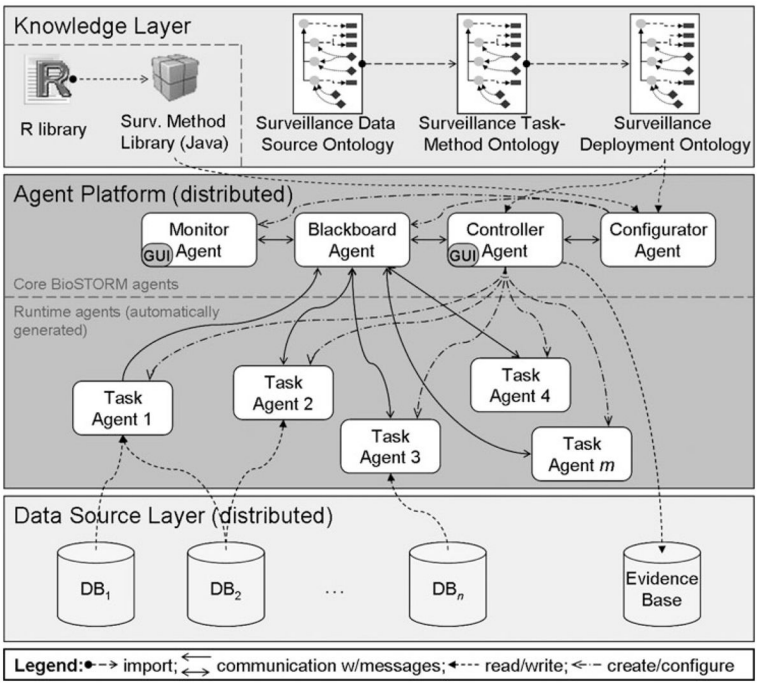

Fig. 8.

The BioSTORM^{B} system architecture illustrating the ontology-driven deployment of PSMs as JADE agents. The architecture has three layers: (1) a knowledge layer, containing all of the ontologies that describe the problem decomposition; (2) an agent platform, which contains the deployed agents in a system; and (3) a data-source layer, which represents a semantically characterized view of the external environment of the agents.