Title: Model-Based Design of Clinical Information Systems

Authors: J. Mathe[1], J. Werner[1], Y. Lee[1], B. Malin[1,2], A. Ledeczi[1]

Affiliations:

[1]Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, Tennessee, USA

[2]Department of Biomedical Informatics, Vanderbilt University Medical Center, Nashville, Tennessee, USA

To Whom Correspondence and Proofs Should Be Addressed:

Janos L. Mathe
2015 Terrace Place
Nashville, Tennessee 37203, USA
Phone: +1 615 3434641
Fax: + 1 615 3437440
Email: jmathe@isis.vanderbilt.edu

Summary

Objective: The goal of this research is to provide a framework to enable the model-based development, simulation, and deployment of clinical information system prototypes with mechanisms that enforce security and privacy policies.

Methods: We developed the Model-Integrated Clinical Information System (MICIS), a software toolkit that is based on model-based design techniques and high-level modeling abstractions to represent complex clinical workflows in a service-oriented architecture paradigm. MICIS translates models into executable constructs, such as web service descriptions, business process execution language procedures, and deployment instructions. MICIS models are enriched with formal security and privacy specifications, which are enforced within the execution environment.

Results: We successfully validated our design platform by modeling multiple clinical workflows and deploying them onto the execution platform.

Conclusions: The model-based approach shows great promise for developing, simulating, and evolving clinical information systems with formal properties and policy restrictions.

Keywords: hospital information systems, medical records systems, computer-aided design, software design

1. Introduction

To reduce preventable errors in patient care and minimize administrative burdens, healthcare organizations (HCOs) are migrating from traditional, paper-based records to clinical information systems (CISs), a collection of computer-based applications that enables sophisticated services for patients and health care providers. Various empirical evidence indicates that CISs can decrease healthcare costs [1, 2, 3, 4], strengthen staff productivity [5, 6], and promote patient safety [7, 8]. As a consequence, HCOs are adopting CISs to enable a wide array of functions, including data sharing, decision support, employee training and student education, research, and access to reference materials. Many HCOs are leveraging CISs to build "web portals", which can be tailored to provide a specific experience based on the role of the intended user [9]. For example, physician portals have been designed to support daily clinical workflows, such that they enable access to guidelines, educational materials, treatment and cost information, and referral directories [10]. Similarly, patient portals have been designed to provide patients with access to their electronic medical records, billing, and appointment scheduling [11, 12].

Despite their potential benefits, the design of CISs presents unique challenges, which derive from the constant evolution and interaction of an HCO's policies, technologies, and workforce. The electronic nature of CISs, for instance, in contrast to the traditional paper-based systems, increases the potential magnitude of system failures and inadvertent exposure of highly sensitive patient-specific health information. It is necessary that HCOs design CISs to adhere to diverse governmental regulations that influence both procedural, as well as access policies, such as the Privacy and Security Rules of the Health Insurance Portability and Accountability Act (HIPAA) [13, 14]. A CIS must account for the complexities of the HCOs, both at the social and the technical level, to ensure timely access to health information and services without compromising the security of the system or sensitive patient records.

Disparate HCOs, as well as departments within an HCO, can differ greatly in terms of the software and technologies that comprise their CISs. We believe that the service-oriented architecture (SOA) paradigm provides an intuitive approach to resolve system diversity and integrate CISs. Instead of relying on site-specific, ad hoc design strategies, SOA provides a mechanism to create complex distributed applications from loosely-coupled services. Workflow definitions define the structure of the applications, capturing the interaction of services with formally provable properties [15, 16, 17]. SOA inherently supports modular design, promotes reusability, and simplifies system evolution since services can be individually modified, upgraded, or completely replaced. Even interface changes can be tolerated because interface definitions are published and can be queried. Moreover, services can be implemented in a number of different languages and they can run on different operating systems, which provides true platform independence. Hence, building CISs on top of SOA leads to improved interoperability and extensibility.

SOAs have been successfully applied in the e-commerce sector [18] and, as a result, a rich infrastructure of SOA-specific tools is available [19, 20, 21]. However, the design and implementation of SOAs for CISs raises nontrivial challenges. In traditional business applications, the human-workflow interaction is often based on an "accept or deny" schema, where a person serves as an approval checkpoint that determines

if a procedure can continue. On the contrary, in the clinical domain, many workflows require human tasks that do not fit this schema, such as a physician's interpretation of laboratory test results. An emerging extension of the Business Process Execution Language for the Web Services (WS-BPEL) standard, called BPEL4People [22], partially addresses this issue, but its usage is limited, mainly because existing BPEL workflow managers do not support the extended functionality. A second challenge is that typical SOA implementations hardcode policies in the workflow logic, which is not amenable to the complex procedural and regulatory policies that are associated with HCOs. To manage intricate and dynamic privacy, security, and access control policies, a more elaborate solution is needed.

To address the aforementioned challenges, we developed the Model-Integrated Clinical Information Systems (MICIS), a software tool suite that assists HCO administrators in the design, verification, implementation and integration of CISs. The MICIS tool suite is capable of graphically representing data, workflows, organizational aspects, and regulatory requirements of the healthcare environment. MICIS translates formal models into the necessary software artifacts and deploys the system on a standard SOA platform. The formal models created in MICIS allow administrators to perform rigorous system analysis and to enforce privacy and security policies within the CIS prior to deployment. MICIS enables a rapid development cycle because CIS prototypes can be quickly generated, evaluated and changed, if necessary, by modifying the system models and regenerating the application. Furthermore, the model-based approach helps over the entire lifecycle of the application by easing maintenance and application evolution, since many aspects of the CIS can be modified via the models without touching the actual code.

In previous work, we presented a high-level overview of the MICIS architecture with respect to platform-specific engineering [23] and the type of abstractions necessary to model the clinical realm [24]. In this paper, we focus on the systems integration aspect of MICIS, which illustrates how to integrate model-based design, SOA, and policy specification in a healthcare environment. We then provide an example of how to model, as well as deploy a specific working component of a clinical information system. Finally, we elaborate on the details of MICIS modeling and deployment platforms, describe extensions used to enforce privacy and security policies, and present how the artifacts generated from the models are used for component integration.

2. Background

In this section, we provide background on the underlying technologies that MICIS is built upon.

2.1. Model Integrated Computing

Model Integrated Computing (MIC) was developed at Vanderbilt University for building software-intensive systems. The core idea behind MIC is to provide a domain-specific modeling language (DSML) and a corresponding modeling environment for the given application domain. The DSML raises the abstraction level above traditional programming languages and provides an application developer, or domain expert, with familiar concepts. MIC can be applied to create and evolve integrated, multiple-

view models using concepts, relations and model composition principles used in the given field. MIC also facilitates systems and software engineering analysis of the models, as well as enables the automatic synthesis of applications from the models. This approach has been successfully applied in several different applications, including automotive manufacturing [25], wireless sensor networks [26], and integrated simulation of embedded systems [27].

A core tool within MIC is the Generic Modeling Environment (GME) [28], which can be configured and adapted from meta-level paradigm specifications, known as metamodels. The metamodels consist of unified modeling language (UML) class diagrams and object constraint language (OCL) constraints. These are created in GME and are applied to automatically configure the software to support the new DSML. This architecture is illustrated on the left side of Figure 1. The well-documented advantages of MIC in general and the highly flexible architecture and customizability of GME in particular, make these technologies an ideal candidate for the foundation of MICIS. Most other modeling tools support a single or a small set of languages, typically UML. While UML has its well-deserved place in software development, we believe that the requirements in many domains, including CIS, call for the application of DSMLs. GME is one of the most advanced metaprogrammable tools that support the rapid development of DSMLs, hence we decided to employ it.

2.2. Service Oriented Architectures

Approaching CIS design from the perspective of SOAs is not unique. In earlier work, Kawamoto and Lobach successfully applied a service-oriented software framework to clinical decision support systems [29]. Still, clinical decision support is only one of many components in CISs and does not model patient-provider interactions, which characterize the healthcare field. The challenge is to design a CIS that is loosely-coupled to a particular SOA environment. This enables the designer to build an experimental infrastructure without being bound to design and execution environments that may not adequately represent the particular CIS in development or have the adaptability necessary to meet changing system requirements. We refer the reader to a recent paper by Jürjens and Rumm, which provides an excellent summary of SOA and model-based approaches in clinical information systems [30]. We highlight that MICIS is distinct from existing approaches in that it creates verifiable, executable workflows from domain-specific models tailored to the healthcare environment.

2.3. Policy Languages and Specification

Policy languages separate the privacy and security requirements from the implementation details of a system. As a result, policies can be changed without altering the underlying implementation [31]. The application of a formal language for policy representation provides greater reuse for the developer, but may not easily represent the high-level goals of business processes. The SECTET framework of Breu et al. facilitates the design and implementation of secure inter-organizational workflows [32]. The abstraction level is raised above the standard SOA languages by capturing workflows and security requirements primarily in the form of UML models [33]. A similar approach is presented by Zhang et al. [34] for describing access control and synthesis of machine-readable policy representation. In the solution presented in [35], privacy and security policies are tightly integrated with the workflows. This approach

provides a sound base for privacy and utility analysis; however, it is unsuitable for highly dynamic environments.

3. Methods

In this section, we present the MICIS framework, an approach to rapidly develop, simulate, and deploy CIS prototypes with automatic security and privacy policy enforcement. The architecture of the MICIS framework is outlined in Figure 1, which illustrates the two primary contributions: 1) the component integration platform (MICIS-CIP) and 2) the model integration platform (MICIS-MIP).

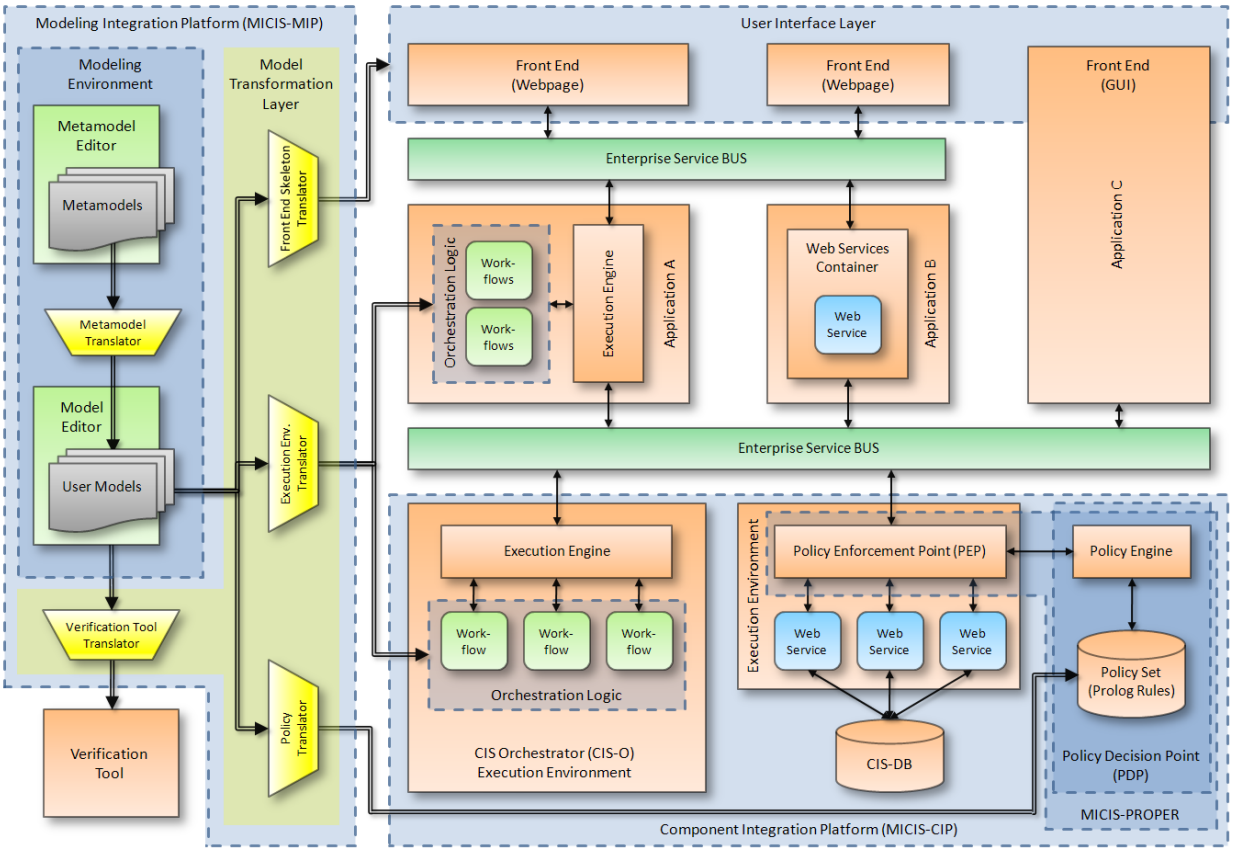

Figure 1. The MICIS architecture.

3.1 Component Integration

MICIS-CIP serves as a base of the clinical information system by providing a set of basic functionalities for the other components of the CIS. These include persistence, session management, and access control. In addition, MICIS-CIP provides access to information stored in the HCO's database. This basic

set of services is then extended with the integration of different software components in a modular, security and privacy preserving manner. The integrated software components implement various higher-level services, such as billing, care provider messaging and process management. The modular approach of MICIS allows for the integration of new services which can reuse existing ones.

The modularity in MICIS is achieved by adopting a set of SOA standards that form the base of the MICIS-CIP (shown on the bottom of the Figure 1). In MICIS-CIP, both basic and extended functionalities are implemented through communicating web services. The orchestration of these services is managed by the CIS Orchestrator (CIS-O), which consists of a SOA compliant execution engine and the web services it manages.

By integrating an SOA compliant execution engine as a part of CIS-O, we gain the ability to specify the orchestration logic of our CIS in terms of workflows. MICIS makes use of workflows to capture the business logic of complex applications and to orchestrate the execution of the corresponding services. Our workflows are defined using the Business Process Execution Language for Web Services (WS-BPEL) [36] standard and are deployed in the CIS-O as executable web services. A benefit of adopting WS-BPEL is that it provides the user with the capability to define the logic of services for a particular CIS in an explicit manner, instead of hard coding the functionality into a web service. This decouples the development of application logic from the specific details of deployment.

In the CIS-O, the execution of these web services is performed by an OASIS compliant BPEL [37,38,39] execution engine, which takes care of managing multiple sessions of instantiated web services. In its current implementation, MICIS utilizes the Apache Orchestration Director Engine (Apache ODE). Examples of how we implement functionality and express the orchestration logic using WS-BPEL can be found in the later sections of the paper.

Communication of the web services, including the connected applications, is achievable with the assistance of an Enterprise Service Bus (ESB), where the communicating web services exchange SOAP [40] messages, which are orchestrated with the Apache ODE workflow manager. In our framework, we rely on the ESB to support messaging, routing, invocation, and transaction management.

3.2 Security and privacy enforcement

Security and privacy enforcement in MICIS is achieved through mechanisms that allow for the definition and deployment of policies in the MICIS-CIP. Specifically, we have developed reusable application-independent Prolog-based Policy Decision Point and Policy Enforcement Point (MICIS-PROPER) components and integrated them with the Apache ODE.

In MICIS, selected services can be *secured* using MICIS-PROPER with its dedicated policies. This means that if a protected service is invoked, the information flow between the service and its invoking client is carefully monitored. In cases where non-compliance with the defined policies is discovered during the information exchange, the execution of the undesired operation is prevented and a exception is returned to the workflow manager. For example, the system can notify administrators of an

unauthorized request for patient information or prevent a physician from ordering a medication that, in combination with a patient's existing medications, could cause an adverse drug effect.

In the MICIS architecture, we use a set of protected services to access the contents of CIS-DB, the database that stores all sensitive information of the HCO. The CIS-DB contains patient-specific information (e.g., demographics, lab results, vitals and treatment information) and organization-specific information (e.g., personnel, departments, physical locations, etc.). Access to protected information is achieved though the ESB and the information flows are controlled by MICIS-PROPER. The details of this protection mechanism are explained in section 3.6.

It should be noted that the architecture is inherently distributed. It allows for an arbitrary number of applications, services and databases, and even multiple component integration platforms (CIP). For example, the CIS-DB does not need to be a centralized database. Multiple databases can be supported as long as access to the data is controlled by an appropriate PEP.

3.3 Extending functionality

MICIS supports three distinct approaches for extending the functionality of a particular CIS. The basic functions can be added as additional web services (1). More complex behavior involving, for example, multiple entities can be specified as workflows and deployed by the CIS Orchestrator (2). Finally, complete application modules, such as Billing Services or Outpatient Monitoring, can be seamlessly integrated into the system (3).

The range of functionality that integrated applications can provide in MICIS is limited only by the ESB and the available set of services deployed as part of the MICIS-CIP. In the upper right corner of Figure 1, we present three different kinds of application modules integrated in the CIP. The modules are A) an application with its own execution engine and explicitly defined workflows, B) an application whose functionality is implemented as a web service and finally, C) a standalone application with its own communication infrastructure directly connected to the ESB. Regardless of the architecture of a particular application, MICIS-CIP provides the ability for users to access the services of a particular CIS with the help of its own front-end.

3.4 Modeling framework

The MICIS model integration platform (MICIS-MIP) is built on Vanderbilt's metaprogrammable Model-Integrated Computing (MIC) tool suite called GME (Modeling Environment in Figure 1) [28]. MICIS was developed in the following steps: (1) specification of domain-specific modeling languages that capture all relevant architectural and policy modeling aspects of CIS applications, and (2) development of model transformations for mapping the domain specific models on the MICIS component integration platform. CIS prototypes were then developed with MICIS by building application models, automatically generating the necessary software artifacts, and finally running the applications.

We enable the framework to model and implement the higher level functionalities of the CIS by capturing the logic that orchestrates their execution and interaction. In MICIS, system models are built

to capture workflows, services, organizations, roles, messages, message attributes, and deployment configurations, as well as access control and security policies. We find that the explicit representation of policies over the orchestration logic, represented by workflows is advantageous in the CIS domain. Our policy modeling language is based on the work of Mitchell et al. on contextual integrity [35]. It enables the formal representation of permitted communications and considers past, as well future, communication instances. Later in the paper we discuss some of the aspects of policy representations in the Policy Evaluation and Enforcement section. Detailed example models are shown in the Illustrative Example section. In the MICIS architecture, the Policy Decision Point (PDP) implements all of the decisions that are to be made in the execution of a workflow, thus enforcing the defined policies.

3.5 Model Transformation Layer

The models captured in MICIS represent the logic that drives a CIS, or a certain part of it. However, without the Model Translator Layer, the models serve as formal documentation only. It is the job of the translators to generate the code that implements the corresponding CIS on top of a SOA platform. The models are automatically translated to application components, assembled, and deployed. The generated artifacts include workflow descriptions in WS-BPEL, web service descriptors in WSDL, and access control and privacy policies in Prolog. Also, verification and simulation tools, which ensure the correct behavior of the system once deployed, can be easily integrated by creating appropriate model translator that configure the given tools automatically from the models.

The Model Transformation Layer is composed of four sets of model translators for (1) workflow orchestration (Execution Environment Translator), (2) creation of the policy rules for policy enforcement (Policy Translator), (3) creation of the front-end interface for users in form of html/jsp pages (Front End Skeleton Translator) and (4) driving external tools to verify the design and to simulate the execution of the system (Verification Tool Translator).

3.6 Policy Evaluation and Enforcement

In order to satisfy the privacy and security requirements in clinical workflows, we have developed an application-independent Prolog-based Policy Evaluation and Policy Enforcement Point (MICIS-PROPER) and a policy modeling suite. MICIS-PROPER addresses the challenges of modeling and enforcing privacy and security policies, dynamic redeployment of existing policies, and customization of the behavior of the policy enforcement point. It is necessary to solve the above mentioned issues to represent the multifaceted requirements specified by laws and regulations such as HIPAA [13, 14], address the organization evolution and introduce the changes with minimum system down-time.

The presented solution consists of the policy and workflow models, policy generators, a policy verification tool, a policy enforcement point (PEP), and a policy decision point (PDP). Figure 2 depicts the MICIS-PROPER architecture.

In our approach, policy, workflow and data models are tightly integrated, which has a significant advantage: information duplication and synchronization between multiple aspects of the model can be avoided. Despite the integration of the models, the generation of the policies is independent from the

generation of the workflow descriptions. Consequently, the underlying policy language could be easily changed by modifying the generators and replacing the policy decision point.

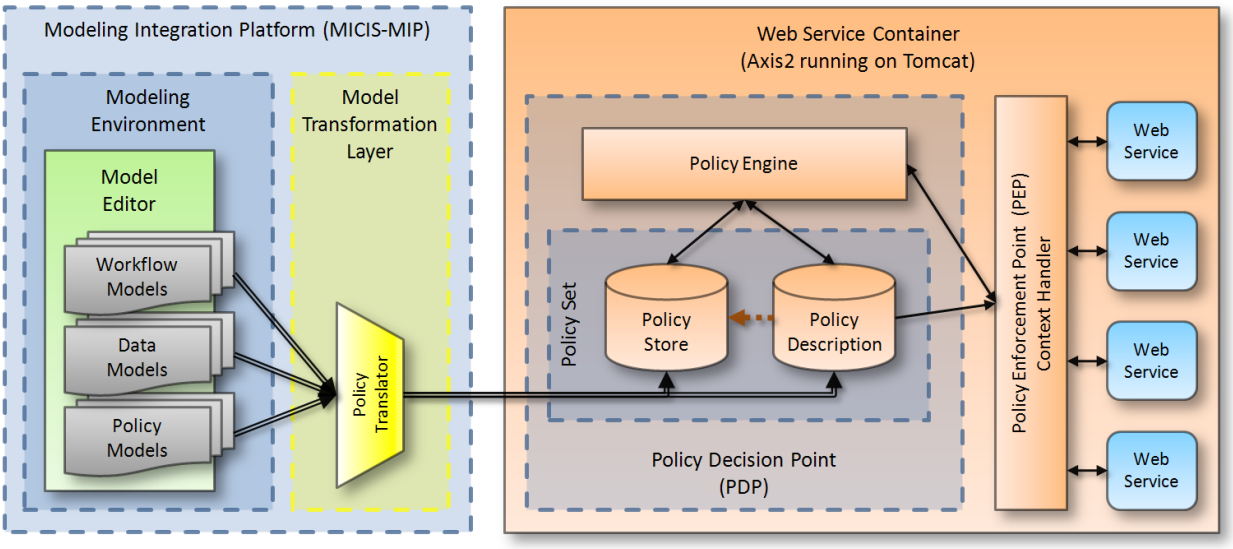

Figure 2. The MICIS-PROPER architecture in relation with the MICIS-MIP.

In the MICIS modeling language, the purpose of a policy model is to represent privacy and security constraints. A policy model is then used to generate the policies and policy descriptions. The generation is performed by the Policy Translator. In MICIS-CIP, the policy description is applied to inform the policy enforcement point about the inputs and outputs of the services and to drive the policy decision point. A simple policy description contains such data as arguments passed to the service, fields returned by the service, fields used by the policy decision point, timing of policy evaluation, and additional obligations. We employ Horn clauses in Prolog to describe the policies. By implementing the Prolog language directly, MICIS imposes stricter requirements on the modeler (i.e., knowledge of the Prolog syntax); however, it provides more expressive power to the policy designer.

Policies and policy descriptions generated from the models are manually deployed in the local policy repositories. The policies are accessed directly by the PDP and PEP. Policies can be thought of as independent documents in the policy container. The policy enforcement point is built into the SOA environment as the request interceptor in the web service container and is completely transparent to services and clients invoking them. This allows for the seamless replacement of policies, addition of protected services, and modification of the policies. Note that redeployment of the policies to the repository does not cause system downtime. Whenever the PEP decides to block the client request, an exception is generated and the flow of control is changed. Fault handling is the responsibility of the workflow manager and consequently the PEP is service and location independent. The policy decision point can be readily substituted, hence MICIS can facilitate decision points from different vendors and leverage different policy languages with different expressiveness. Currently, MICIS supports the Sun XACML implementation [41] for the OASIS XACML standard [42] and the SWI-Prolog engine [43] for

Prolog-based policies. We also integrated an additional obligation module to support custom, user defined functions to be executed upon policy evaluation.

3.7 Using MICIS

When using MICIS, developers need to work with both the Modeling Integration Platform and the Component Integration Platform; however, we suspect that the developers' work with the CIP will be minimal. Currently, if a developer wants to create and integrate a new application into an existing CIS, any new components including databases or user front ends would have to be manually developed within the CIP. Existing components can be simply reused. We intend to automate the generation of simplified web-based user interfaces to the services deployed and to automate the generation of the skeleton of databases.

4. Illustrative Example

In this section, we demonstrate the capabilities of the MICIS platform by implementing an example scenario, an outpatient monitoring system (OPM). To explain the context of the scenario, assume that an elderly patient is monitored in his home with the assistance of sensors mounted on his body. An outpatient monitoring station constantly examines the data from the sensors, looking for deviations from normal values in the patient's vital signs. Detected deviations are reported to a monitoring station in the hospital.

The scenario is initiated when the patient experiences an abrupt change in blood pressure. When the monitoring system detects abnormal values or trends, an alert message is generated. Figure 3 shows the process execution path for handling such an outpatient alert message.

As soon as the clinical information system orchestrator (*CIS-O)* receives the alert message, it begins to execute the business process defined for this scenario. After logging the alarm status in the electronic medical record (EMR) system, the *CIS-O* sends the message to the *Alert Monitor System*, which renders it on a monitoring station. When the nurse checks the message, she requests the patient's medical record to evaluate the situation. The *CIS-O* provides the information from the *EMR* to the *Alert Monitor System*. This includes the medical history and the contact information, which the nurse can use to validate the alert. If the alert is deemed important, she writes the status to the patient medical record. Finally, the *CIS-O* forwards the alert message to the designated physician(s) by using the *Message Delivery System*. Otherwise, the alert message is stored in the EMR system and the process is terminated.

To describe the workflow in charge of handling the alert message, we define three different business processes (*OPMAlertMain, OPMAlertStore* and *PatientInformation,*) along with Prolog-based policies within the *OPMAlertStore* and *PatientInformation* process. In this example, the nurse intervenes in the process flow twice: first, to collect the patient's record and second, to validate the alert message. To protect sensitive patient information, we introduce appropriate policies for the two processes

OPMAlertStore, *PatientInformation.* These policies are enforced by the Prolog-based Policy Decision Point and Policy Enforcement Point.

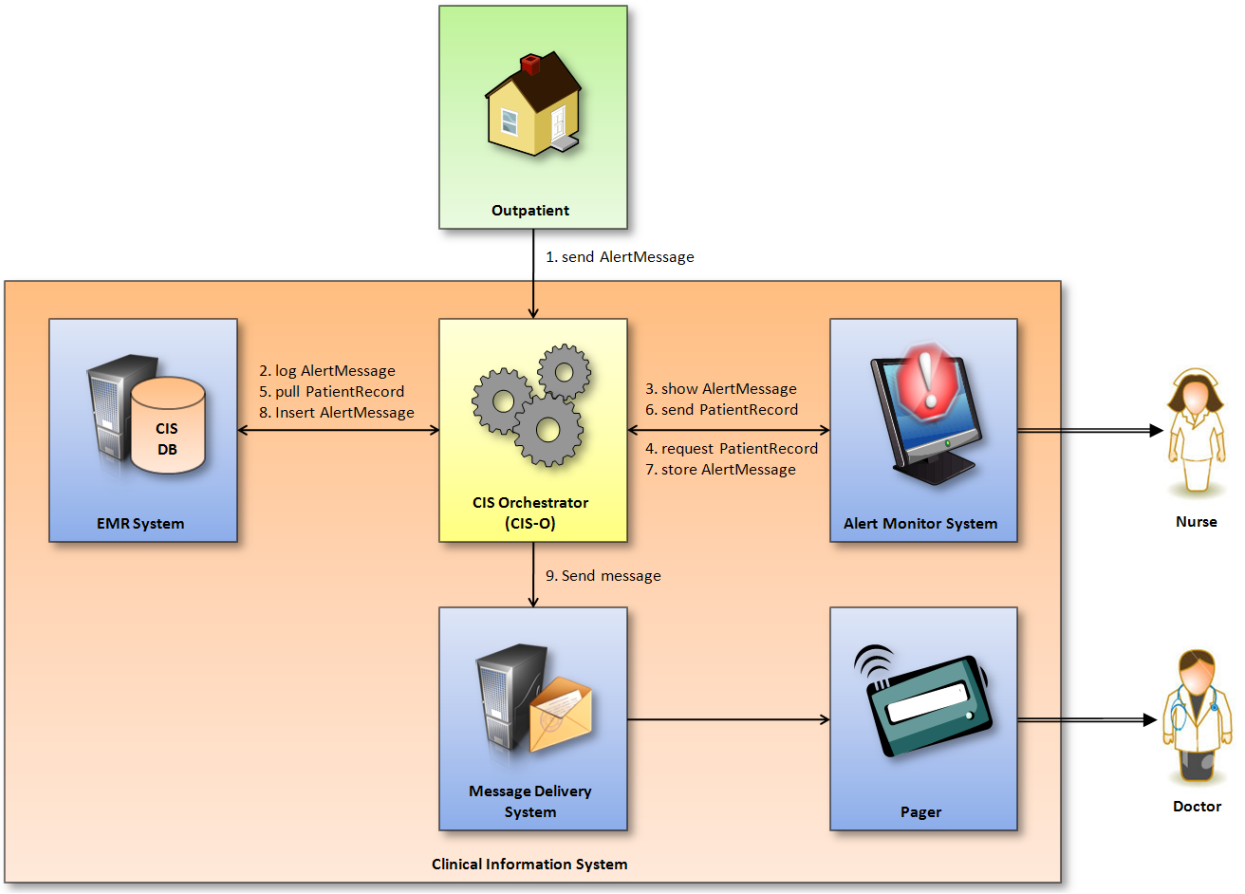

Figure 3 – A collaboration diagram of the outpatient monitoring scenario.

Figure 4 shows the models that characterize the *PatientInformation* and *OPMAlertStore* processes. *OPMAlertMain* is initiated when the *CIS-O* receives the alert message triggered by abnormal values from the outpatient monitoring system. Implementation details of *OPMAlertMain* are omitted. The *PatientInformation* process (Workflow A) in Figure 4 is invoked when a nurse requests a patient's medical information from the EMR after reviewing the alert message. Once the *receiveEMRRequest* activity (1) receives the request including the patient and staff identifiers, the activity uses this information to retrieve the requested patient medical record from the *EMR System*. The *invokeEMR* activity (2) invokes the *PatientInformation* Web Service and the *receive* activity (3) retrieves the patient medical record from the web service. Between these two activities, we apply the privacy policies to restrict unauthorized access to the electronic medical records. Finally, to send the patient medical record to the nurse, it is assigned to the output variable of the *PatientInformation* workflow (4).

The purpose of the OPMAlertStore process depicted on the Figure 4 (Workflow B) is to store the result of the nurse's validation of the alert. After the alert status is assigned to the *OPMAlert* data type (1) , the *invokeEMRStore* activity (2) invokes the *PatientInformation* web service to store the validation results in

the EMR System. When the *invokeEMRStore* activity invokes the *Patient Information* web service, we also apply the privacy policies. After the *receive* activity (3) receives the acknowledge message from the web service, it is assigned to the AlertMessage variable. The *InvokeMessageSender* activity (4) invokes the *MessageSender* web service to forward the alert message to the designated doctors by using the Message Delivery System.

For this scenario we introduced the following restrictions:

- Only medical staff is permitted to access alert messages.
- Only primary care physicians are permitted to access patients' medical records.
- Nurses are permitted to access the records of patients under surveillance by the OPM system.
- Medical staff is permitted to access patients' records in emergency situations, which trigger the "Break Glass" policy which allows for an emergency elevation of access privileges. This action is subsequently audited by the HCO's policy administrators [44].

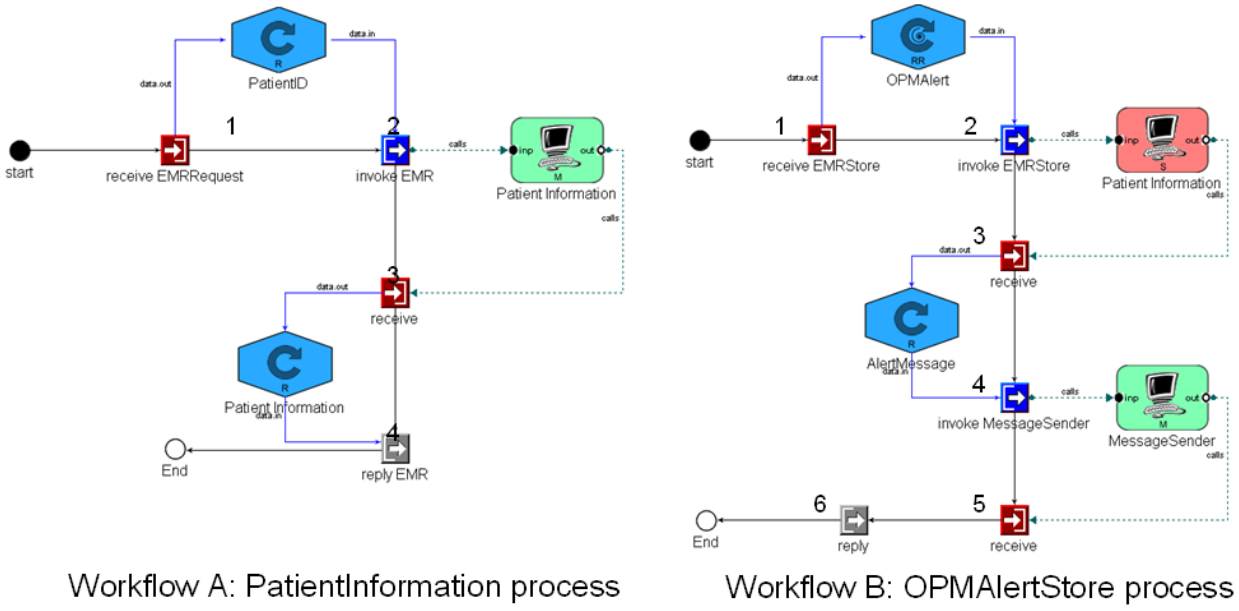

Figure 4. Patient Information process descriptions examples.

These policies are enforced for all invocations of the *PatientInformation* service. A policy for the *PatientInformation* service is presented on the left side of Figure 5. The policy description includes the definitions of incoming and outgoing data, the evaluation point, obligations, and additional datasets for policy evaluation. A detailed policy model is depicted on the right side of Figure 5. The model contains information required to generate the policy: the query is evaluated to determine access rights, attribute relations used for policy evaluation and a textual policy description. In this example, we evaluate the policy with a query *retrievedata(PatientID, staffID)* after the service has been executed. To decide on

that policy, the Prolog-based PDP uses the predefined set of predicates and attribute relation, i.e. *critical()* and *treats(staffID,MRN),* which are generated from the incoming and outgoing data by the PEP.

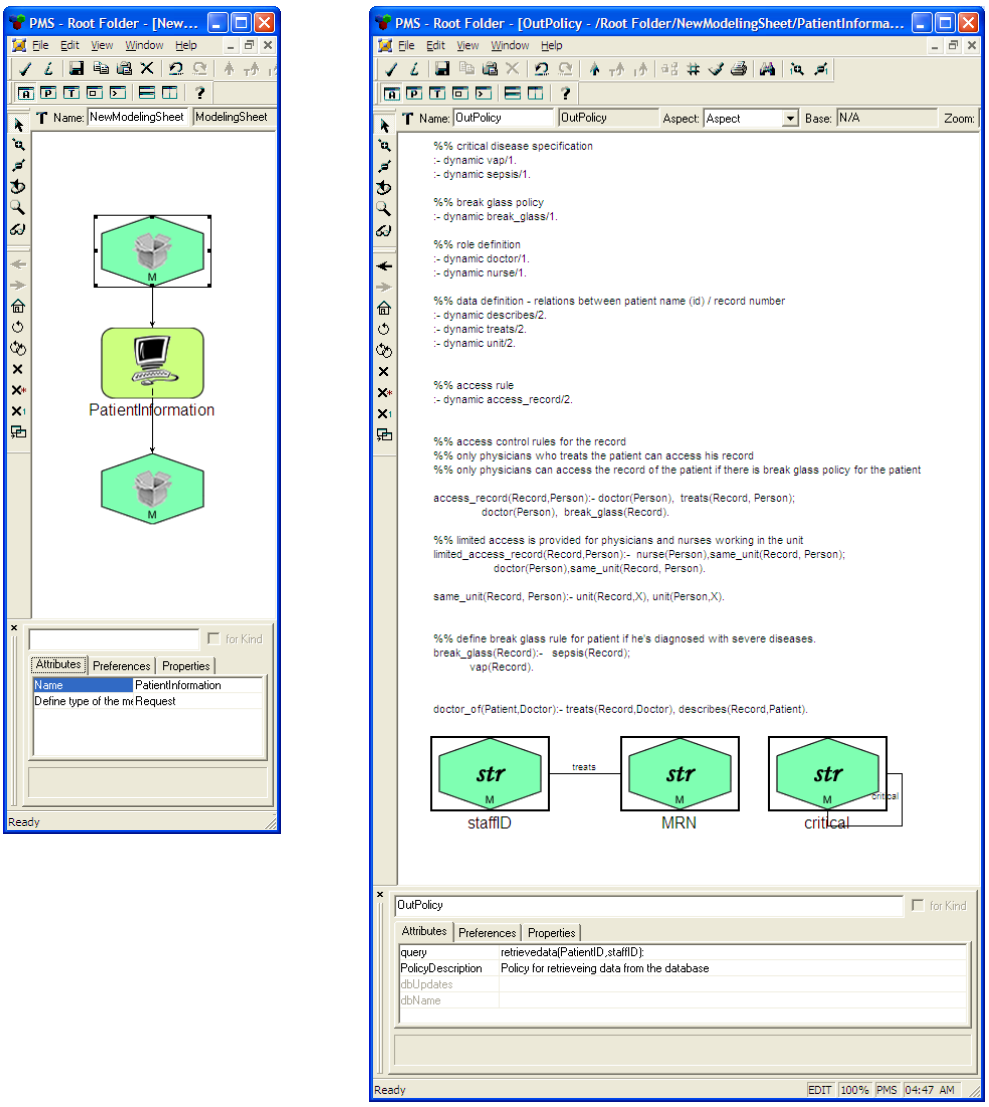

Figure 5. An excerpt of the policy model.

The workflow and security policy models of the *OPMAlert* are applied to generate the source files by the *Execution Environment Translator* and *Policy Translator*. The source files are required by the ODE BPEL engine for correct execution of the workflow. These include 1) the BPEL source file, which describes the orchestration logic; 2) the Web Service Definition (WSDL) interface file with the necessary data structures in XML Schema Definition (XSD) format, where the latter defines the input and output message for the services to allow other processes to connect to them, and 3) the compilation and deployment script file (deploy.xml), which is in charge of deploying the aforementioned files onto the ODE process manager. The *Policy Translator* generates two files: 1) the policy description in an XML format, which drives the PEP and 2) a Prolog document that describes the policies to be evaluated by the PDP.

5. Discussion

The outpatient monitoring example described previously demonstrates the potential of this approach. A working prototype implementation was successfully created using MICIS and deployed in a SOA environment (Apache ODE + Axis2) in a few weeks. In fact, most of the effort was dedicated to the design and implementation of the individual services comprising the application. Once the services were completed, the creation of the system models and the automatic generation, configuration, and deployment of the running application, which included several testing and debugging cycles, required only a couple of days. This suggests that, if a rich set of services are available, CISs can be rapidly developed and evolved using our approach. While currently this is not necessarily the case in many HCOs, the investment to migrate existing applications to SOA would be well worth it. It would make them easier to reuse, maintain, and evolve. Coupled with MICIS, these benefits would be even more evident. For example, MICIS decouples policy representation and enforcement from the application logic, hence, any application created from the available library of services would be guaranteed to comply with all applicable policies automatically.

The current implementation of MICIS serves as a proof-of-concept that model integrated development and integration of the clinical information systems on top of SOA is a promising way to support health care organizations. We recognize that the widespread deployment of a tool such as MICIS will be slow, even within the Vanderbilt environment, for two main reasons. One is the current lack of available SOA-compliant services in the complex IT infrastructures currently being used in many HCOs. The other is the understandable reluctance of the organization's management to accept new technologies in a mission critical environment. Despite these challenges, we believe that MICIS can be applied to develop SOA-based clinical information systems in parallel to existing solutions. Then, when the organization is ready, the model-based system can replace the legacy code and systems that comprise the HCO's information technology infrastructure. At the present time, we are making progress within Vanderbilt and have demonstrated how the model-based technology can assist in different well-defined, smaller-scale scenarios. For example, we have demonstrated how a similar approach can be applied to design a patient portal [25]. Vanderbilt researchers are also working with MICIS to develop a patient treatment monitoring tool using evidence-based protocol models.

While we have successful developed and demonstrated the MICIS toolkit, we recognize there are certain limitations to the presented work. Here, we touch upon several of these issues, which provide a basis for the extension of this research. First, our results are limited by the fact that our implementation is only a proof-of-concept system and has not been tested in a large scale, heavily distributed environment. Second, during our system testing, we did not perform a load analysis to evaluate the performance of the off-the-shelf SOA tools that MICIS relies upon. Third, our current presented policy language is somewhat restrictive in the sense that it is incapable of representing policies that express rules where the relative order of events is stated. Examples for such policies can be found in [35]. We consider the above mentioned issues to be part of our future work.

6. Conclusions

In this paper, we described an experimental development environment called the Model-Integrated Clinical Information Systems (MICIS). The incorporation of model-based design techniques and the utilization of high-level modeling abstractions provide many advantages over ad hoc design approaches that many HCOs have relied upon. The explicit representation of the application architecture and other aspects of a CIS can support verification, simulation, code generation, automatic deployment and documentation of CIS prototypes, all from the same set of models, which provides consistency eliminating many potential sources of error. Specifically, the proposed MICIS framework offers a novel way to represent complex medical workflows in the service-oriented architecture paradigm. More importantly, it allows for the formal representation of security and privacy policies at design time and transparently enforces them during run time. Policies are driven by multiple federal, state and local laws and standards regulating access to sensitive information, such as patient data. Embedding the enforcement of these policies in application logic is error-prone and results in difficult to maintain code. By using MICIS, compliance can be strengthened and changes in policies can be accommodated without any changes to actual application code.

Acknowledgements

To design MICIS for a real world clinical environment, we collaborated with doctors, administrators and software engineers from the Vanderbilt University Medical Center. We are particularly thankful to Peter Miller, Jim Jirjis, Jason Martin, Jim Weaver, Lisa Weavind and David Maron. We appreciate all the comments and questions from anonymous reviewers, which allowed us to improve this article. This work was supported by National Science Foundation grant CCF-0424422, the Team for Research in Ubiquitous Secure Technology.

References

1. Kaushal A, Jha A, Franz C, Glaser J, Shetty K, Jaggi T, et al. Return on investment for a computerized physician order entry system. J Am Med Inform Assoc. 2006; 13(3): 261-6.

2. Shabo A. A global socio-economic-medico-legal model for the sustainability of longitudinal electronic health records. Methods Inf Med. 2006; 45(3): 240-5.

3. Simon SJ, Simon SJ. An examination of the financial feasibility of Electronic Medical Records (EMRs): a case study of tangible and intangible benefits. Int J Med Inform. 2006; 2(2): 185-200.

4. Menachemi N, Brooks RG. Reviewing the benefits and costs of electronic health records and associated patient safety technologies. J Med Syst. 2006; 30(3): 159-68.

5. Lo HG, Newmark LP, Yoon C, Volk LA, Carlson VL, Kittler AF, et al. Electronic health records in specialty care: a time-motion study. J Am Med Inform Assoc. 2007; 14(5) 609-15.

6. Pizziferri L, Kittler AF, Volk LA, Honour MM, Gupta S, Wang T, et al. Primary care physician time utilization before and after implementation of an electronic health record: a time-motion study. J Biomed Inform. 2005; 38(3): 176-88.

7. Silver MR, Lusk R. Patient safety: a tale of two systems. Qual Manag Health Care. 2002; 10(2): 12-22.

8. Frank L, Galanos H, Penn S, Wetz HF Jr. Using BPI and emerging technology to improve patient safety. J Healthc Inf Manag. 2004; 18(1): 65-71.

9. Shepherd M, Zitner D, Watters C. Medical portals: web-based access to medical information. In: Sprague, RH Jr., ed. Proceedings of the 33rd Annual Hawaii International Conference on System Sciences (HICSS-33); 2000 Jan 4-7; Maui, HI. New York: IEEE Computer Society. 2000: p. 5003-13.

10. Barnett GO, Barry MJ, Robb-Nicholson C, Morgan M. Overcoming information overload: an information system for the primary care physician. In: Fieschi M, Coiera E, Li YC, eds. Proceedings of the 11th World Congress on Medical Informatics (Medinfo 2004); 2004 Sept 7-11; San Francisco, CA. Amsterdam: IOS Press. 2004; 11(Pt 1): p. 273-6.

11. Masys D, Baker D, Butros A, Cowles KE. Giving patients access to their medical records: the PCASSO experience. J Am Med Inform Assoc. 2002; 9: 181-91.

12. Cimino JJ, Patel VL, Kushniruk AW. The patient clinical information system (PatCIS): technical solutions for and experience with giving patients access to their electronic medical records. Int J Med Inform. 2002; 68: 113-27.

13. U.S. Department of Health and Human Services. Standards for privacy of individually identifiable health information; Final Rule. Federal Register, 2002 Aug 12; 45 CFR: Parts 160-164.

14. U.S. Department of Health and Human Services, Office for Civil Rights. Standards for protection of electronic health information; Final Rule. Federal Register, 2003 Feb 20; 45 CFR: Part 164.

15. Yanchuk A, Ivanyukovich A, Marchese M. Towards a mathematical foundation for service-oriented applications design [cited 2008 Jul 21]. Available from: http://www.science.unitn.it/~marchese/pdf/Towards_SOAD_JoS_06.pdf

16. Portier B. SOA terminology overview, Part 1: Service, architecture, governance, and business terms [cited 2008 Jul 21]. Available from: http://www-128.ibm.com/developerworks/library/ws-soa-term1/index.html

17. Portier B. SOA terminology overview, Part 2: Development processes, models, and assets [cited 2008 Jul 21]. Available from: http://www-128.ibm.com/developerworks/library/ws-soa-term2/index.html

18. Gray J. A conversation with Werner Vogels, CTO, Amazon.com. Web Services. 2006 [cited 2008 Jul 21]. Available from: http://portal.acm.org/ft_gateway.cfm?id=1142065&type=pdf

19. Service-Oriented Architecture [cited 2008 Jul 21]. Available from: http://www.oracle.com/technologies/soa/index.html

20. Service-Oriented Architecture (SOA) [cited 2008 Jul 21]. Available from: http://www.sun.com/products/soa/index.jsp

21. SOA Software – Solutions – SOA Fabric [cited 2008 Jul 21]. Available from: http://www.soa.com/index.php/section/solutions/soa_fabric/

22. WS-BPEL Extension for People – BPEL4People, http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_white_paper.pdf

23. Werner J, Mathe J, Duncavage S, Malin B, Lédeczi Á, Jirjis J, Sztipanovits J. Platform-based design for clinical information systems. In: Dietrich D, Hancke G, Palensky P, eds. Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN); 2007 June 23-27; Vienna, Austria. New York: IEEE Computer Society. 2007: p. 749-55.

24. Duncavage S, Mathe J, Werner J, Malin B, Lédeczi Á, Sztipanovits J. A modeling environment for patient portals. In: Teich JM, ed. Proceedings of the 2007 American Medical Informatics Association Annual Symposium (AMIA); 2007 Nov 10-14; Washington, DC. Philadelphia: Hanley & Belfus. 2007: 201-6.

25. Long E, Misra A, Sztipanovits J. Increasing productivity at Saturn. IEEE Computer Magazine. 1998; 31(8): 35-43.

26. Völgyesi P, Maróti M, Dóra S, Osses E, Lédeczi Á. Software composition and verification for sensor networks. Science of Computer Programming. 2005; 56(1-2) 191-210.

27. Lédeczi Á, Davis J, Neema S, Agrawal A. Modeling methodology for integrated simulation of embedded systems. ACM Trans on Modeling and Computer Simulation. 2003; 13(1): 82-103.

28. The Generic Modeling Environment website [cited 2008 Jul 21]. Available from: http://www.isis.vanderbilt.edu/projects/gme/

29. Kawamoto K, Lobach D. Proposal for fulfilling strategic objectives of the U.S. roadmap for national action on decision support through a service-oriented architecture leveraging HL7 services. J Am Med Inform Assoc. 2007; 14: 146-55.

30. Jürjens J, Rumm R. Model-based security analysis of the German health card architecture. Methods Inf Med. In this issue.

31. Sloman MS. Policy driven management for distributed systems. Journal of Network and Systems Management. 1994: 2(4): 333-360.

32. Alam M, Hafner M, Breu R. A constraint based role based access control in the SECTET: a model-driven approach. Journal of Computer Security. 2008; 16(2): 223-260.

33. Alam M, Breu R, Hafner M. Modeling permissions in a (U/X)ML world. In: Wagner R, Pernul G, Takizawa M, Quirchmayr G, Tjoa A, eds. Proceedings of the 1st International Conference on Availability, Reliability, and Security (ARES); 2006 April 20-22; Vienna, Austria. New York: IEEE Computer Society. 2006: p. 685-92.

34. Zhang N, Ryan M, Gueley D. Synthesising verified access control systems in XACML. In: Backes B, Basin D, Waidner M, eds. Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering (FMSE); 2004 Oct 29; Washington DC. New York: ACM Press. 2004: 56-65.

35. Barth A, et al. Privacy and utility in business processes. In Proc: IEEE CSF. 2007: 279-294.

36. OASIS Web Services Business Process Execution Language (WS-BPEL) [cited 2008 Jul 21]. Available from: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

37. Apache Foundation, Apache Orchestration Director Engine [cited 2008 Jul 21]. Available from: http://ode.apache.org.

38. Oracle BPEL Process Manager website [cited 2008 Jul 21]. Available from: http://www.oracle.com/technology/bpel/index.html

39. ActiveBPEL Open Source Engine Project website [cited 2008 Jul 21]. Available from: http://www.active-endpoints.com/active-bpel-engine-overview.htm

40. SOAP standard [cited 2008 Jul 21]. Available from: http://www.w3.org/TR/soap/

41. Sun's XACML Implementation [cited 2008 Jul 21]. Available from: http://sunxacml.sourceforge.net/

42. OASIS eXtensible Access Control Markup Language (XACML) TC [cited 2008 Jul 21]. Available from: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

43. SWI-Prolog engine [cited 2008 Jul 21]. Available from: http://www.swi-prolog.org

44. Joint NEMA/COCIR/JIRA Security and Privacy Committee (SPC). "Breakglass – an approach to granting emergency access to healthcare systems" [cited 2008 Jul 21]. Available from: http://www.nema.org/prod/med/security/.