Implementing a Model-Based Design Environment for Clinical Information Systems

Janos Mathe[1], Sean Duncavage[1], Jan Werner[1], Bradley Malin[1,2], Akos Ledeczi[1], and Janos Sztipanovits[1]

[1]Department of Electrical Engineering and Computer Science, School of Engineering
[2]Department of Biomedical Informatics, School of Medicine
Vanderbilt University, Nashville, TN 37232 USA
`{janos.l.mathe, sean.duncavage, jan.werner, akos.ledeczi, b.malin, janos.sztipanovits}@vanderbilt.edu`

Abstract. Health care is a rapidly evolving field that is increasingly supported through clinical information systems (CIS) that integrate care providers, patients, and computer applications. Local and federal regulations require health care systems to define and enforce privacy and security policies to protect sensitive patient data within CIS. Service-oriented architectures (SOA) have been successfully applied to specific clinical services, such as decision support, but have yet to be adopted for large-scale CIS that need to account for diverse information technology architectures and complex person-computer interactions. In this work, we demonstrate that the incorporation of model-based design techniques and high-level modeling abstractions provide a framework to rapidly develop, simulate, and deploy CIS prototypes. This paper describes the implementation of a graphical design environment that allows CIS architects to develop formal system models and from these automatically generates executable code for deployment. The design tool leverages SOA to create reusable services that can be rapidly adapted. We illustrate the functionality of the tool by modeling a secure messaging service in the MyHealth@Vanderbilt patient portal, a portion of the Vanderbilt University Medical Center CIS.

Keywords: Model Integrated Computing, Privacy, Patient Portal, Workflow

1 Introduction

The treatment of patients is paramount in the health care community, but an information system with errors that are difficult to find and address can lead to serious mistakes in patient care. To reduce these errors and minimize administrative burdens, health care organizations (HCOs) are migrating from traditional, paper-based records to clinical information systems (CIS) that provide a collection of computer-based applications that enable sophisticated services for patients and health care providers. Already, electronic medical records (EMR) have been shown to both increase staff productivity and patient safety [1]. As CIS evolve, HCOs are integrating new applications to provide access to information and manage organizational relationships within the healthcare environment.

CIS leverage and incorporate a variety of technologies, such as electronic medical record systems that provide a gateway to numerous information and organizational components of the healthcare environment. As a result, CIS can enable a wide array of functions, including data sharing, decision support, training, research, and access to reference materials. CIS "web portals" can be tailored to provide a specific experience based on the role of the user [2]. For example, physician portals can be designed to support the daily clinical workflow, so that they have access to guidelines, educational materials, treatment and cost information, referral directories [3]. Alternatively, patient portals can be designed that provide patients with access to their electronic medical records, billing, and appointment scheduling. [4], [5].

The design of CIS presents unique challenges that mainly derive from the fact that HCOs are dynamic entities with constantly evolving policies and technologies. HCOs require complex technical, as well as socio-technical, interactions in the clinical environment. For instance, workflows in hospitals can vary between departments and each department has continuous turnover of employees with differing roles. Moreover, CIS administrators must support diverse regulations at the federal, state, and local levels that influence both procedural, as well as access policies. Nonetheless, due to the sensitivity of patient information and the potential for an increased magnitude in errors, the design of CIS is a critical issue that directly affects the HCOs, in addition to the well-being of patients. Complexities in HCOs must be modeled in CIS to ensure secure and timely access to health information and services.

To address this problem, we have evaluated the necessary requirements and developed a software tool suite, called Model Integrated Clinical Information Systems (MICIS) that assists in the formal design, verification and rapid prototyping of CIS. Previously, we presented a high-level overview of the MICIS architecture with respect to platform-specific engineering [6] and the type of CIS "abstractions" that are necessary for modeling the clinical realm [7]. In this paper, we describe the implementation details of MICIS. The MICIS tool is able to graphically represent data, workflow, and organizational aspects of the healthcare environment. MICIS translates the formal models into a Service-Oriented Architecture (SOA) and transforms them into a secure web-accessible portal that includes both procedural, as well as, access policies. The formal models created in MICIS allow us to perform rigorous systems analysis, as well as investigate the privacy and security implications of CIS, including the data passed among care providers and where patient information is stored.

The remainder of this paper is organized as follows. In Section 2 we provide motivation for SOA in the clinical realm and review research in related areas. In addition, we provide background into the underlying technologies that MICIS is built upon, such as model integrated computing. In Section 3, we describe the architectural design of MICIS. We then present how MICIS was implemented using the Oracle BPEL server and a model integrated computing toolkit. Then, in Section 4, we discuss a challenge regarding the representation of policies and their enforcement. Finally, in Section 5, we discuss some of the limitations and next steps in the development of the MICIS tool suite.

2 Background and Motivation

Service Oriented Architectures and the Complexity of CIS

The healthcare environment is highly variable, and may differ greatly between disparate HCOs in terms of alternative software systems, as well as within hospitals in terms of the responsibilities of particular clinics. SOA provide an intuitive means to resolve and integrate such diversity. Instead of relying on site-specific, ad hoc design strategies, SOA provide a formal way to coordinate services by using a web-inspired architectural style that relies on loosely-coupled, interacting services to compose complex applications [8]. Services, in the context of SOA, are independent, heterogeneous components, which can be accessed through predefined interfaces and composed into workflows representing business logic [9][10]. The principal design goals of using services are composability, adaptability, and platform independence, all of which lead to improved interoperability among systems as well as future extensibility.

SOA have been used successfully in the business sector by companies such as Amazon [11], and as a result, a rich infrastructure of SOA tools is available [12], [13], [14]. By using an existing service-based approach for CIS, we gain maintainability, scalability, and generalizability [15]. However, the design and implementation of SOA in CIS raises nontrivial challenges. For instance, the abstractions used for service representations in an off-the-shelf product may not adequately capture the role of human processes prevalent in a clinical setting. In traditional business applications, the human- workflow interaction is often based on a simple “accept or deny” schema where a person serves as an approval checkpoint in order to determine if execution can continue. Yet in the clinical domain, many workflows require human tasks that cannot be cast into a binary decision, such as interpreting chart data or diagnosing patients; existing SOA tool suites do not provide a way to capture these.

Furthermore, CIS have unique requirements due to the complexity of their policies. Procedural policies require secure and timely delivery of health information while privacy policies mandate particular accessibility rules for both patients and healthcare providers. The policies specified in the Privacy and Security Rules of the Health Insurance Portability and Accountability Act (HIPAA) present both procedural and access policies that must be supported by CIS [16][17]. However, existing SOA implementations require *procedural policies* to be hard-coded into the workflow logic and do not provide a uniform method for representing *access policies*. Simple access polices, for example encryption of messages exchanged by service provider and invoker can be addressed by Oasis Web Services Security [18], [19]. Although we adopt it for our execution engine in MICIS it is clear that for expressing general access policies (e.g. restricting the access to personal data) a more elaborate solution would be required; otherwise, additional security enforcement is left for the system designer. By coupling design and deployment environments, SOA implementations binds potential CIS developers to particular implemented technologies and limit future system evolution.

MICIS is distinct in that it creates verifiable, executable workflows from domain-specific models tailored to the healthcare environment. Approaching CIS with SOA is not unique. Kawamoto and Lobach successfully applied a service-oriented software framework to clinical decision support systems [20]. However, clinical decision support is only one of many components in CIS 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.

Formal Modeling Tools

MICIS makes use of workflows to capture the business logic of a health portal and orchestrate the execution of services. An orchestration language creates a coherent story of service execution from one viewpoint, such as the chronology of the services invoked when a patient sends a message to a physician. The Business Process Execution Language (BPEL) is one such language that relies on workflow descriptions to represent business logic [21]. By using the BPEL standard as the basis for workflow modeling, MICIS is compatible with any OASIS compliant BPEL execution engine, which decouples the development of the health portal from deployment specific details.

Policy specification languages are able to separate abstract security policies from implementation details. As a result, policies can be dynamically changed without altering the underlying implementation [22]. Sun's implementation of the eXtensible Access Control Markup Language (XACML)[23] is a formal policy language specification based on the OASIS standard [24]. Using a formal language for policies provides greater reuse for the developer but may not easily represent the high-level goals of business processes. Bridging the gap between low-level abstractions and high-level goals is explored in [25], which presents a model-driven approach for access policies. MICIS uses a similar approach by automatically transforming domain models into machine-enforceable XACML.

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 the application developer/domain expert with familiar concepts. MIC is used to create and evolve integrated, multiple-view models using concepts, relations and model composition principles used in the given field. It also facilitates systems/software engineering analysis of the models, and enables the automatic synthesis of applications from the models. The approach has been successfully applied in several different applications, including automotive manufacturing [26], wireless sensor networks [27], and integrated simulation of embedded systems [28], to name a few.

A core tool in MIC is the Generic Modeling Environment (GME) that can be configured and adapted from meta-level paradigm specifications, known as metamodels. These metamodels consist of UML class diagrams and OCL constraints. They are created in GME and are used to automatically configure it to support the new DSML. Specifically, a software tool called the metamodel translator parses the metamodels and generates an XML file containing the DSML specifications in a concise format. GME in turn reads this file and configures itself to support the new DSML. This architecture is illustrated on the top half of Figure 1. GME has a sophisticated user interface and a flexible extension mechanism making it easily customizable even beyond supporting a wide variety of modeling languages. For example, several high-level APIs in different programming languages make it easy to create additional tools interfacing with GME as well as model translators. The most widely used API is a domain-specific, high-level C++ interface automatically generated from the metamodels by the metamodel translator.

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 laying the foundation of MICIS.

3 MICIS Design and Implementation

To design MICIS for a real world clinical environment, we collaborated with administrators and software engineers from the Vanderbilt University Medical Center. Specifically, we based MICIS on information learned from the MyHealth@Vanderbilt (MHAV) Patient Portal, which is currently in use, and was designed by the Vanderbilt University Medical Center (VUMC). For illustrative purposes in this paper, we use workflows that depict services similar to those offered by MHAV [29**].** The MHAV portal provides services for patients, including access to lab results, billing information, scheduling of appointments and secure messaging with doctors.

Architecture Overview

Figure 1 provides an overview of the MICIS architecture. MICIS achieves an agile design with the assistance of the GME tool suite **(Modeling block)** [30]. With the help of the GME tool suite, we utilize existing SOA tools in order to provide a runtime environment for a designed CIS (**Execution Environment** block) and provide designers with the opportunity to incorporate verification and simulation tools.

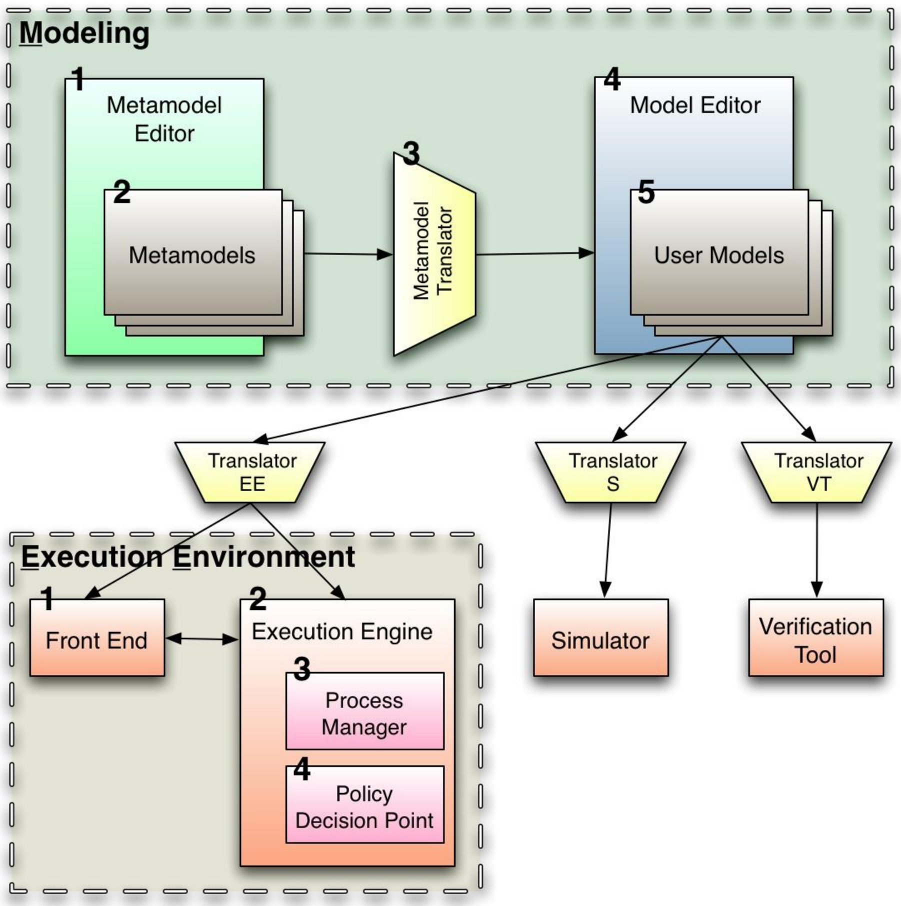

Fig 1. MICIS Architecture, where we facilitate the GME tool suite in the Modeling block to create a design environment for CIS. The models that are created in this design environment (the Model Editor) then can be translated as input for various analysis and execution engines.

Components and their implementations

The first component of MICIS is the **Domain-specific modeling language editor**, which is shown in Figure 1 as component 1) of the Modeling block. The editor is used for the design and creation of domain-specific modeling languages, or "metamodels".

We used the editor to create the component **Metamodels**, a formal language that represents the necessary abstractions of the CIS domain. This is shown as component 2) in the Modeling block of Figure 1. Tailoring the clinical abstractions required a series of interactions with the designers of the MHAV patient portal and the hospital staff at VUMC. Through these interactions, we have created the abstractions that drive the Domain-specific model editor (component 4 of the Modeling block). The development of these abstractions was an iterative process requiring several revisions of the (meta)models. In the current iteration, we partition the abstractions into five classes: A) workflows, B) data structures, C) policies, D) deployments, and E) organizational structures. The details of the abstractions are described in [7]. Through this set of abstractions, a CIS designer can specify the orchestration logic for a CIS. During this process, called the modeling process, the designer can identify the services the CIS should provide. The designer can also specify the manner by which people and computer-based entities interact with the components that are in charge of implementing these services.

The **Metamodel Translator** of the GME tool suite – shown in Figure 1 as component 3) of the Modeling block – uses the modeling language to automatically configure GME for the domain thus enforcing our abstractions. This operation allows the **Domain-specific model editor**, to enable the creation of instances of the abstraction in form of models in our graphical modeling environment. The Domain-specific model editor is depicted in Figure 1 as component 4) of the Modeling block.

In the model editor we created sample **Models**, – shown in Figure 1 as component 5) of the Modeling block, – which are based on MHAV. An example workflow model taken from the model editor is presented in Figure 2. This workflow depicts a scenario in which a patient, currently logged in to MHAV, attempts to retrieve the history of a messaging session with the medical staff.

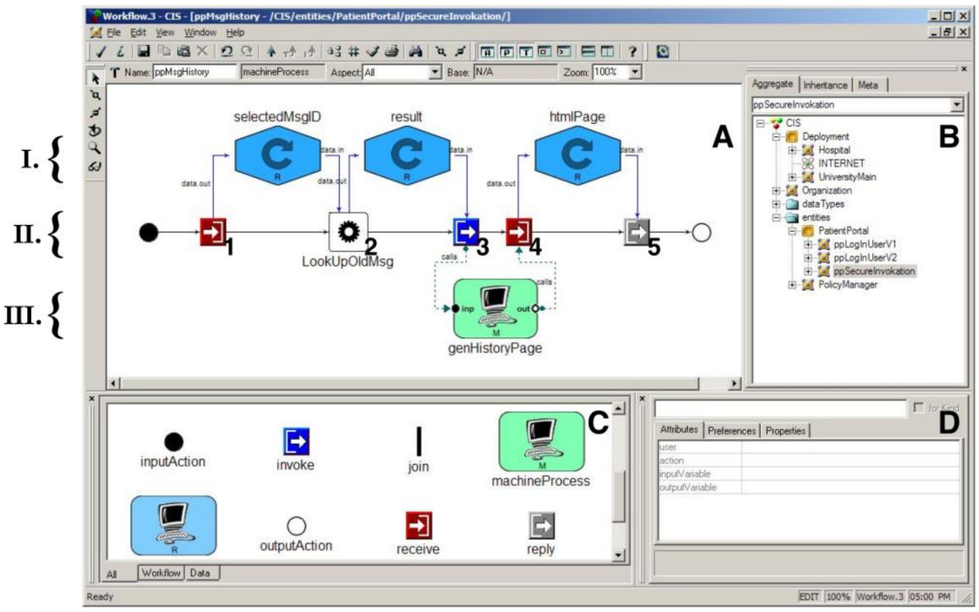

Fig. 2. Service ***ppMsgHistory*** retrieves the history of a messaging session.

Figure 2 shows the model editor, which has four principal components: A) the model builder pane (shown in the upper left corner of the figure), B) the tree view of the hierarchical component structure (shown in the upper right corner of the figure), C) the model parts browser (shown in the lower left corner of the figure) and D) the model information pane, with Attributes, Preferences and Properties tabs that allow for the configuration of models (shown in the lower right corner of the figure).

In the model builder pane (block A of Figure 2) the example workflow model walks through a series of steps described by the *control flow*. Here, we use the term model to describe any element in the hierarchical structure of models; i.e., for both basic building blocks and complex structures. Complex structures, such as the example workflow model in Figure 2, can be built from both simple and other complex building blocks. In block A the roman numbers describe the three basic categories of the models: I. contains the data structures, II. shows the workflow logic and III. presents the participating services. In the workflow, unlabeled black lines are drawn between the *inputAction* (black filled circle), which represents the starting point, and the *outputAction* (empty circle), which represents the endpoint. In this example, the series of steps are a simple sequence of operations. The operations in the sequence are 1) a reception of the user input (*receive* model); 2) a local operation called "LookUpOldMsg" *task*, which performs a lookup in a database table with a message ID as an input and returns with the message history text as an output; 3) a service *invocation* that invokes the remote service called "genHistoryPage" that assembles the results into a viewable format; 4) a reception of the result of the invoked service; and finally 5) the sending of the requested information back to the user with a *reply*.

The flow of information (*data structures* marked by blue hexagons) travels in and out of components of the control flow and is described with *data flows* (the blue, tagged lines in the diagram). This mechanism helps the creator of a workflow tie two distinct aspects of two separate control structures together with the help of a common data schema. For further explanation on how the model editor works we refer the reader to [30] and for information on what the language is capable of expressing, as well as additional examples, we refer the reader to [7].

A collection of the (MICIS) models are intended to illustrate a formal representation of the logic that drives a CIS, or a certain part of it. However, the collection only serves as a formal documentation in this form, because it lacks a model interpreter, which would translate the models into executable code. This is the purpose of the **Translator for the Execution Environment (Translator EE)**, which connects the Models block with the Execution Environment block in Figure 1. It traverses the models with the help of the GME interface [30] and produces executable code, in the form of configuration files for the Execution Environment, using a layer of functions that build on C++ and TinyXML [31].

To generate code in a cost efficient manner we facilitate the powerful arsenal of applications created for Service-Oriented Architectures in order to implement the **Execution Environment** block [Fig.1]. The Execution Environment is a group of applications running on a set of servers that contain the **Execution Engine**, the **Policy Decision Point** and the **Front End**. The Execution Engine is in charge of providing the defined services for a CIS, such as managing the incoming requests and executing the defined workflows based on the defined policies. The Front End provides access to the services maintained by the execution engine. To find a suitable application serving as an Execution Engine we have examined various applications that could be configured with workflow descriptions. We decided on the Oracle BPEL Process Manager (Version 10.0.1.3) [32], which uses a BPEL-based workflow representation and has a mature set of tools including a web-based console that provides access for managing workflow instances. We recognize that there are alternative tools and we note that we have also experimented with the open-source ActiveBPEL [33] and have evaluated the compatibility of our workflows with it. We have concluded that the source files can be exchanged between the two engines with minor changes.

We found that the explicit representation of policies over the orchestration logic (represented by workflows) in the CIS domain is a necessity. We discuss some of the aspects of policy representations in the following section. In our architecture the **Policy Decision Point (PDP)** implements all the decisions that have to be made in the execution of a workflow, which enforces the existing defined policies. A subset of these policies defines access control within the CIS. In order to implement these policies we have chosen to generate XACML expressions (using Translator EE) and enforce them with the help of Sun's XACML Implementation, Axis 1.2 and Tomcat 5.5 – installed on dedicated servers.

To make MICIS compatible with the SOA tools that form the Execution Engine we implemented **Translator EE** so that it generates code (based on our models in the modeling environment) in a language that can be interpreted by the components of the Execution Engine. The model translator **Translator EE** is composed of three main components: A) the model translator for workflow orchestration (with policy enforcement) that generates input for the Oracle BPEL Process Manager, B) the model translator for creating policy decisions and C) the model translator for creating the front-end interface for users (in form of html/jsp pages).

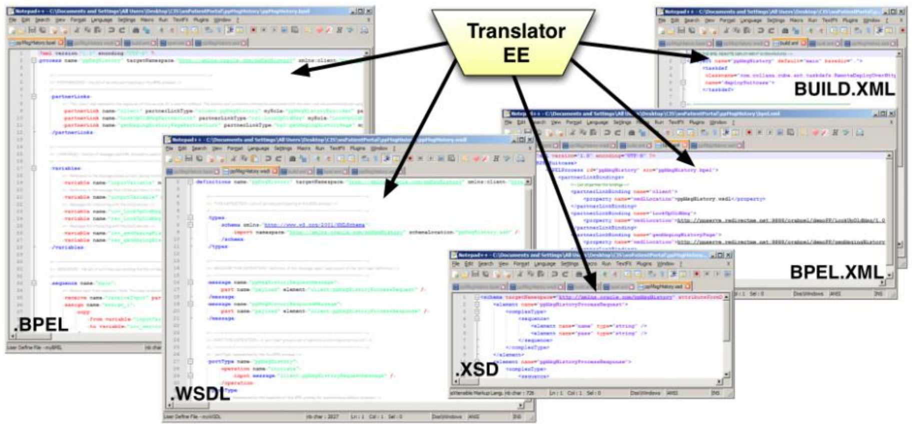

Fig 3. Source files generated with *Translator EE* based on *ppMsgHistory* workflow.

Figure 3 shows the source files that are generated by component A) of Translator 1. The source files are required by the Oracle Process Manager for correct execution of workflows. These correspond to 1) the BPEL source file, which describes the orchestration logic; 2) the Web Service Definition Language (WSDL) interface file with the necessary data structures in the form of XML Schema Definition (XSD) files, which defines the input and output messages for the service to allow other processes to connect it; 3) the BPEL deployment descriptor file (bpel.xml), which defines the location of the used WSDL files; and 4) the compilation and deployment (Apache Ant) script file (build.xml), which is in charge of deploying the previous files onto the Process Manager. The policy enforcement is integrated into workflows by A) and also translated to XACML-based policy decision points by B). The C) component of Translator EE has not been developed yet, which means that when we generate input for the Execution Engine we have to manually create the front-end web pages to be able to interact with the services. The development of C) is currently work in progress. In order to generate the front-end pages we plan to utilize the input and output data structures of the workflow models.

Future components of MICIS

The purpose of the Simulator **Translator** (Translator S) working together with the **Simulator** of [Fig. 1] is to provide CIS developers with the possibility to simulate and test the implemented orchestration logic. This is a future component of our architecture and we are looking into using UPPAAL [34], CPN Tools [35] or even the built-in simulator of the ActiveBPEL design environment [33].

The **Verification Tool Translator** (Translator V) is the model translator for policy and workflow verification. It generates input for the **Verification Tool** [Fig. 1], which is a component of the proposed architecture that works tightly together with the Simulator. The verification tool creates the possibility to reason about and verify certain properties of a given CIS system. These properties could be anything from a simple reachability analysis of workflow structures to a policy validation. This is a component that has not been implemented yet, we are currently looking into using Prolog and CPN Tools.

4 Challenges to Policy Representation

It required several iterations to find a suitable representation for the workflows so that it is tailored to the problem set and easily interpretable by the CIS design staff. Defining abstractions for representing a broad range of policies is a similarly difficult challenge.

When the Execution Engine instantiates and executes the example workflow in Figure 2, we assume that the user who invoked the service has already been validated against a user data-base. We found that this type of access control policy validation can be achieved by implementing policy decisions points in the workflows. However, this approach requires the designer to insert a decision point into the control flow of each workflow that requires the particular access policy.

One can avoid inserting multiple policy decision points that enforce the same policy by combining the relevant workflows into another that implements the PDP. This approach is depicted in Figure 4. The ppSecInvocation service implements the services and policies required by our example Patient Portal messaging. In the example, we create one, main service, which sends and receives messages to and from patients and clinical staff. Such a service requires the invocation of subservices, like retrieving a contact list and sending a message, where each subservice adheres to the same access policy. Instead of adding a PDP to each workflow for user validation, we created a higher-level workflow that groups together services requiring the same access policy, creating a simple visual confirmation that all messaging subservices conform to our specified policy. The example workflow results in an “unauthorized user” fault if the current user does not have access privileges for a given service.

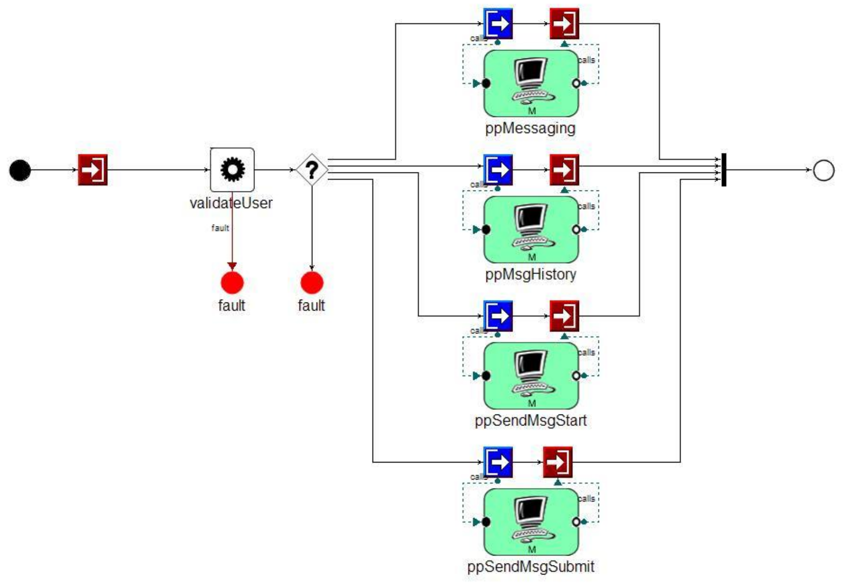

Fig 4. Service ***ppSecInvocation*** shows how invocation of various services can be tied together in a workflow. First the user gets authenticated by the *validateUser* task, which is a policy decision point implemented by the workflow. Assuming that the user gets successfully authenticated, the workflow decides which service to invoke based on user’s input (such as a URL reference).

If we tie together all of the subservices it would result in a series of steps described by the picture in Figure 5. Figure 5 is a sequence diagram that illustrates an example case in which a user logs in to the example Patient Portal system and after a successful login invokes the messaging service (ex: with clicking on a URL reference on the main page). The user invocation causes the Execution Engine to instantiate the previously described *ppSecInvocation* service, which would then validate the user before presenting him or her the messaging options. In the example the user then invokes the service (*ppMsgHistory*) that would display the history of his or her messaging, which again would have to go through the *ppSecInvocation* service.

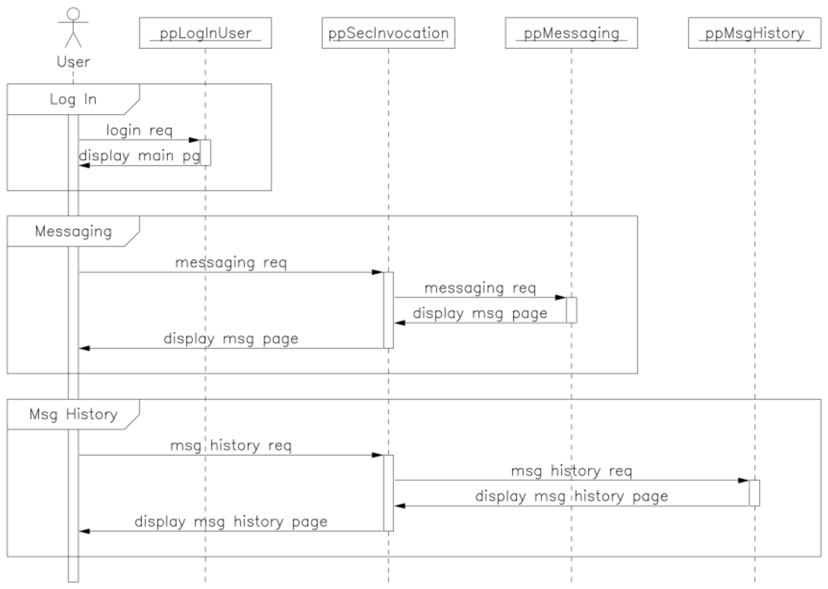

Fig 5. One possible execution path that a user (of the presented simple Patient Portal) could take. The services that the user invokes are the Log In service and some of the subservices of the ppSecInvokation service [Fig.4].

There are disadvantages of this approach. First, we are not only creating an additional service component, where we have to tie all the "real" services that we want to implement in a centralized fashion, but by flattening two different concepts into one we make it difficult to understand the underlying logic. One can imagine what kind of chaos a changing, already implemented, policy would create in a real life system, which usually have numerous workflows and policies defined.

An optimal case, in which policies are not modeled and implemented as part of the workflows, is depicted in Figure 6.

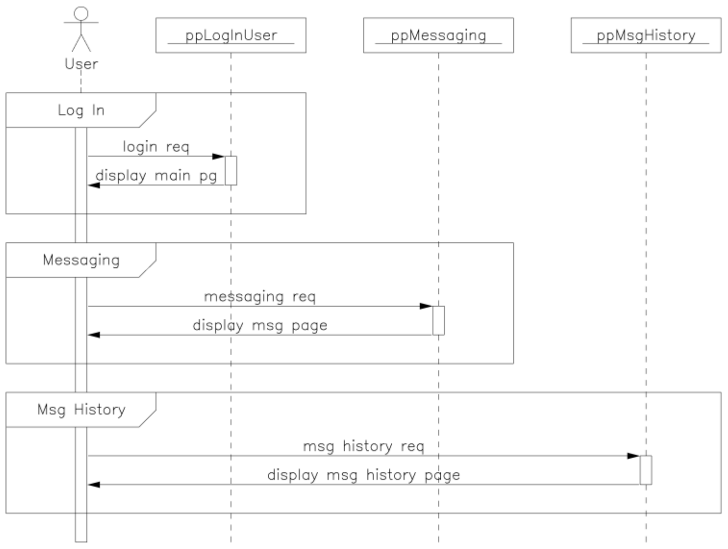

Fig 6. The execution of the same example (as seen in Fig.5) but this time with PDP implemented on a separate abstraction layer than workflows.

This method assumes the enforcement of the defined policies with the help of PDPs implemented independently of workflows. In order to achieve it operations of the control flow – defined in the workflows – would need to be intercepted and matched against the defined set of policies. Depending on the type of operation and the calling data, an operation could be either allowed or denied at the decision point.

5 Discussion and Conclusions

The MICIS tool suite provides a domain-specific modeling environment for explicitly defining workflows, data, organization, and policy in relation to health portal. It transforms CIS domain-models into executable code that can be managed by an off-the-shelf technology, such as Oracle, which, as a result, decouples the developer from a particular deployment architecture and allows greater design flexibility. MICIS incorporates several technologies that allow for deployment from a set of user-defined models.

MICIS is a work-in-progress and, as such, possesses several shortcomings. The most prominent of these reflects the incomplete status of the tool suite: Many of the code-generating translators have yet to be fully implemented and provide only a subset of functionality. Currently, translation to BPEL does not support all of the workflow constructs, and policy translation requires user data-tags instead of automatically generating them from the model. These problems are only the result of time constraints and will be addressed in the near future.

A second limitation affecting policies also results from its incomplete status; however, our short-term solution will require modification in the future. Currently, policy enforcement is achieved through XACML, which has no support for temporal policies, such as those that specify an action must occur before another action or while some event is occurring. Mitchell has proposed a language that can correct this limitation [36], but an execution engine has yet to be developed for it.

Despite these limitations, MICIS is currently capable of modeling health portals with formal domain-specific models and is able to generate executable code with policy enforcement for limited examples. By extending the functionality of MICIS to automatically deploy interacting services with temporal policy enforcement, our future goal is to overcome the current system limitations and create a tool suite capable of modeling, simulating, verifying and deploying prototype CIS.

References

1. Davies NM. Healthcare Information and Management Systems Society: The ROI of EMR-EHR: Productivity Soars, Hospitals Save Time and, Yes, Money. *HIMSS Journal*. 2006.
2. Shepherd, M., Zitner, D., Watters, C.: Medical portals: web-based access to medical information. *Proc 33rd HICSS*. 2000: 5003.
3. Barnett, G., Barry, M., Robb-Nicholson, C., Morgan, M.: Overcoming information overload: an information system for the primary care physician. In: Proceedings of Medinfo 11(Pt 1) (2004) 273-276.
4. Masys, D., Baker, D., Butros, A., Cowles, K. Giving patients access to their medical records: the PCASSO experience. Journal of the American Medical Informatics Association. 9 (2002) 181- 191.
5. Cimino, J., Patel, V., Kushniruk, A. The patient clinical information system (PatCIS). International Journal of Medical Informatics. 68 (2002) 113-127.
6. Werner, J., Mathe, J.L., Duncavage, S., Malin, B., Ledeczi, A., Jirjis, J. Sztipanovits, J. In: Proceedings of the 5th IEEE International Conference on Industrial Informatics. (2007) *Forthcoming*.
7. Duncavage, S., Mathe, J.L., Werner, J., Malin, B., Ledeczi, A., Sztipanovits, J. A modeling environment for patient portals. Proceedings of the 2007 American Medical Informatics Association Annual Symposium. 2007; *Forthcoming*.
8. Yanchuk, A., Ivanyukovich, A., Marchese, M.: "Towards a Mathematical Foundation for Service-Oriented Applications Design", http://www.science.unitn.it/~marchese/pdf/Towards_SOAD_JoS_06.pdf
9. Portier, B.: "SOA terminology overview, Part 1: Service, architecture, governance, and business terms", http://www-128.ibm.com/developerworks/library/ws-soa-term1/index.html
10. Portier, B.: "SOA terminology overview, Part 2: Development processes, models, and assets", http://www-128.ibm.com/developerworks/library/ws-soa-term2/index.html
11. Gray, J.: A conversation with Werner Vogels, CTO, Amazon.com. Web Services 2006 http://portal.acm.org/ft_gateway.cfm?id=1142065&type=pdf
12. Service-Oriented Architecture, http://www.oracle.com/technologies/soa/index.html
13. Service-Oriented Architecture (SOA), http://www.sun.com/products/soa/index.jsp

14. SOA Software – Solutions – SOA Fabric, http://www.soa.com/index.php/section/solutions/soa_fabric/
15. O'Brien, L., Bass, L., Merson, P.: Quality Attributes and Service-Oriented Architectures. Technical Note, Software Architecture Technology Initiative (2005).
16. 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.
17. 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: Pt. 164.
18. OASIS Web Services Security (WSS) TC, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss
19. 1.3 Using Oracle SOA Suite to Adopt SOA, http://download.oracle.com/docs/cd/B31017_01/core.1013/b28764/intro003.htm
20. 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.
21. OASIS Web Services Business Process Execution Language. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
22. Sloman, M.S.: *Policy Driven Management for Distributed Systems*. Journal of Network and Systems Management, 2(4), pp. 333-360, 1994.
23. Sun's XACML Implementation, http://sunxacml.sourceforge.net/
24. OASIS eXtensible Access Control Markup Language (XACML) TC, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
25. Alam, M., Hafner, M., Breu, R.: A Constraint based Role Based Access Control in the SECTET: A Model-Driven Approach
26. Long E., Misra A., Sztipanovits J.: Increasing Productivity at Saturn, IEEE Computer Magazine, August, 1998
27. Volgyesi P., Maroti M., Dora S., Osses E., Ledeczi A.: Software Composition and Verification for Sensor Networks, Science of Computer Programming (Elsevier), 56, 1-2, pp. 191-210, April, 2005.
28. Ledeczi A., Davis J., Neema, S., Agrawal, A.: Modeling Methodology for Integrated Simulation of Embedded Systems, ACM Transactions on Modeling and Computer Simulation, 13, 1, pp. 82-103, January, 2003
29. MyHealth@Vanderbilt website, https://www.myhealthatvanderbilt.com/app
30. The Generic Modeling Environment website, http://www.isis.vanderbilt.edu/projects/gme/
31. TinyXML project website, http://www.grinninglizard.com/tinyxml
32. Oracle BPEL Process Manager website, http://www.oracle.com/technology/bpel/index.html
33. ActiveBPEL Open Source Engine Project website, http://www.active-endpoints.com/active-bpel-engine-overview.htm
34. UPPAAL, http://www.uppaal.com/
35. cpntools, http://wiki.daimi.au.dk/cpntools/cpntools.wiki
36. Barth, A., Datta, A., Mitchell, J.C., Nissenbaum, H.: Privacy and Contextual Integrity: Framework and Applications. 2006 IEEE Symposium on Security and Privacy, pp. 184—198, IEEE Press, New York (2006)