A Framework for the Maintenance and Evolution of ePolicy-guided Web Applications

S. Bergin
Department of Computer Science
National University of Ireland, Maynooth
Email: susan.bergin@may.ie

J.G. Keating
Department of Computer Science
National University of Ireland, Maynooth
Email: john.keating@may.ie

M. Masullo
InViVoVision, Inc.
Connecticut, USA
Email: drmasullo@invivovision.com

A. Benoit
Department of Computer Science
National University of Ireland, Maynooth

INFORMS
Institute for Operations Research and the Management Sciences
Group Decision and Negotiation 2005
University of Vienna
Vienna, Austria
July 10 – 13, 2005

Abstract

In this paper we present an "ePolicy framework" that can be used to develop transactional-based ePolicy-guided Web applications. This framework incorporates a non-proprietary component based architecture, a well-defined standards-based user interface, a structured representation of ePolicies, ePolicy operations and user input data, and incorporates a maintenance management component. Each component is self-contained and can therefore be independently maintained. ePolicies and associated ePolicy operations are not embedded in the system software but are stored centrally in an external store (Policy Repository) and are dynamically loaded as required. Executable code (marshalled from XML) is automatically generated from the ePolicies and the ePolicy operations and used in policy-guided evaluation. The Policy Repository, accessible by suitably privileged components, removes ePolicy duplication and from a maintenance perspective, this approach reduces the possibility of errors being introduced by data duplication. Updates to ePolicies are seamlessly applied the next time an ePolicy is loaded. ePolicies are represented in a standard uniform format and as all components use this uniform format, maintainers do not need to understand or handle multiple data formats. They are represented using a policy hierarchy composed of three layers: meta-ePolicies, ePolicy-groups and ePolicies. Each of the components is designed using Object-Oriented principles. Our ePolicy framework will work in a centralized or distributed environment. We believe that using our framework to develop ePolicy-guided evaluation systems will reduce data maintenance and expedite software evolution.

Keywords: ePolicy, ePolicy-guided evaluation, maintenance, evolution

1. INTRODUCTION

Web systems can be classified as web sites and web applications. Web sites refer to online document repositories while web applications refer to information systems working through a web interface [1]. Web applications can be further categorized according to their content, for example, information driven applications (online books, digital libraries etc.), interactive applications (games,

customized presentations etc.), transaction-based applications (electronic shopping, banking etc.), workflow applications (inventory management, scheduling systems etc.), collaborative work environment applications (distributed authoring tools, collaborative design tools etc.), online community applications, (marketplaces, chat groups etc.) and web-portal applications (electronic shopping malls, online intermediaries etc.) [2]. Architecturally, web applications tend to involve a web server, a network, the HTTP protocol and a web browser, and can be medium-to-large in scale, involve sophisticated interactions between users and databases, and often require frequent and fast updates [3]. In addition, they often employ numerous presentation formats for the user interface, and also multiple communications channels and multiple data formats at the server-side. As a result, web application development and maintenance requires a good understanding of a variety of heterogeneous tools, technologies and concepts. These include HTTP protocol handling, persistent storage, security technologies, session management, dynamic content creation, presentational abstraction and flexible legacy system wrapping [6]. Given that web applications often require numerous swift updates, the application should be based on a careful design and a flexible architecture, or it quickly becomes very difficult to maintain [7] [8]. As web applications are typically not built using sound engineering principles [9] [10] we can expect that their maintenance and evolution can often be difficult.

Our earlier research was concerned with designing web-applications to reduce the maintenance effort as well as modelling the maintenance effort involved in changing a web application to reflect changes in the real-world [4]. In this paper we describe an ePolicy framework that we believe will greatly reduce the maintenance effort involved in web applications. This framework is particularly suitable to transactional-based web applications, such as online trading stores. In the rest of this paper, we present our ePolicy-guided framework, provide an overview of the current architecture and suggest development technologies. We present a critique of our framework and finally provide a supporting example, using a typical online trading example.

2. ePOLICY FRAMEWORK

A framework is a system that can be customized, specialized or extended to provide more specific, more appropriate or slightly different capabilities [11]. Application-specific frameworks cover precise domains, are highly reusable and significantly reduce the amount of development required to deploy further customized applications [12]. We propose an “ePolicy framework” that can be used to design transactional-based web applications that incorporate a non-proprietary component based architecture, a well-defined standards-based user interface, a structured representation of software processing, software logic, policies and user input data, and incorporates a maintenance management component.

In our framework, a policy or more precisely an electronic policy (ePolicy) should be understood as rules, guiding principles, rules-of-thumb and other similar decision-making tools. These ePolicies may be used to make electronic decisions based on electronic transactions. For example, in an online store ePolicies could be applied to a customer order to determine if a particular sale should be allowed or not. The decision making process is composed of several stages. Firstly, a user (for example, a customer) will provide the necessary input data to the system. This input data acts as a trigger that initiates a decision-making process. A component in the framework will retrieve the ePolicy or ePolicies relevant to the decision. A framework component will carry out the evaluation process and reach a decision. This decision will be enforced by the system and the user will be notified of the result.

Our framework is composed of several interconnected components. Each component is self-contained and can therefore be independently maintained. In our framework, the ePolicies and associated ePolicy operations are not embedded in the evaluation software but are stored centrally in an external store and are loaded as required. This repository, accessible by suitably privileged components, removes ePolicy duplication and from a maintenance perspective, this approach reduces the possibility of errors being introduced by data duplication. If an ePolicy changes, an update is only required once and at a centralized location. This update is seamlessly applied the next time the ePolicy is loaded. ePolicies are represented in a standard uniform format and as all components use this uniform format, maintainers do not need to understand or handle multiple data formats.

ePolicies are represented using a policy hierarchy composed of three layers: meta-ePolicies, ePolicy-groups and ePolicies. A meta-ePolicy outlines the ePolicy groups (named logical sets of ePolicies) relevant to a particular decision and each ePolicy-group in turn outlines the particular ePolicies that should be applied when making a decision. In small systems, ePolicy-groups may not be necessary in the hierarchy and meta-ePolicies can reference ePolicy instances directly. Additionally, a meta-ePolicy can provide other relevant ePolicy information, for example, it can specify where a relevant ePolicy group(s) is stored in a distributed architecture and it also can specify a priority level for an ePolicy instance or an ePolicy group. This priority level is used when ePolicies conflict during the evaluation process. An ePolicy with a higher priority will be successful in a conflict.

The structure of ePolicy operations, currently including, and, or, not, equal, greater than, less than, elementOf, subsetOf, isFirst, isLast, isValidDate, to be applied in the evaluation of ePolicies are also stored externally and loaded into the relevant evaluation component as required. Consequently, if an ePolicy operation changes, modification is only required in a single centralized location and the ePolicy-guided evaluation process does not any further modification. If a new ePolicy operation is introduced and its structure is not currently defined in the system, then the respective ePolicy-guided evaluation component would have to be modified to handle this new structure. However, our framework is sufficiently flexible so that new evaluation components can be dynamically loaded and unloaded without affecting any other components.

Meta-policies, ePolicy groups, ePolicies and ePolicy operations may be distributed across a network since policies can often originate from different jurisdictions: legal policies, social policies, organizational policies, etc. The syntactic structures of meta-policies, ePolicy groups, ePolicies and ePolicy operations are represented by well-defined, structured grammars. In addition, the user-input data is translated into a well-defined user request grammar. Well-defined structures also help to reduce the maintenance effort as the maintainer only needs to understand a standard syntax and does not need to translate changes to any other formats or languages.

Our ePolicy framework will greatly reduce the maintenance effort involved in web applications given its, standard structured grammatical representation, separation of independent role-based components, separation of ePolicies, ePolicy operations and supporting software, highly-organized policy hierarchy and standard representation of input data. ePolicies and ePolicy operations can be easily created, updated and removed and new components can be seamlessly integrated into the framework without requiring the existing ePolicy and ePolicy operations structure to change.

3. Framework Overview

In this section we present a technical overview of the current architecture for our ePolicy framework for web applications. We describe each of the components in detail and outline the interaction between each component. We present a critique of our framework and describe the maintenance effort involved in changing a system designed using the framework to reflect changes in the application domain.

As outlined in the previous section our ePolicy framework employs a separate component for each role in the decision-making process. Each of the components are designed using Object-Oriented principles [5]. Techniques such as inheritance, abstraction and polymorphism are used to facilitate code reuse, ensuring minimal code development by system maintainers. Our framework is composed of one or more of the following components: a Request Handler (RH) component, which accepts user requests and delivers dynamic responses; a Policy Enforcement Point (PEP) component, responsible for invoking the ePolicy-guided evaluation process and enforcing the outcome; a Policy Evaluation Logic (PEL) component which carries out the evaluation process using the user input data, the ePolicies and the ePolicy operations; a Policy Repository (PR) component, which stores the meta-ePolicies, ePolicy-groups, ePolicies and ePolicy operations; a Policy Distribution Point (PDP) component which keeps track of distributed PR's and a Policy Maintenance Management (PMM) component which can be used to carry out maintenance on framework components, ePolicies and ePolicy operations.

A Policy Maintenance Manager (PMM) component carries out maintenance tasks on PR's, PEL's and PEP's in its jurisdiction. A PMM interacts exclusively with its own PR. If a PMM requires an update to a PR in a different jurisdiction it can contact the controlling PMM and register an update request. The PMM is specifically responsible for: handling all updates (creation, removal and modification) of ePolicies, meta-policies, ePolicy-groups and ePolicy operations; receiving requests from and sending requests to other PMM's for updates to PR's and finally, facilitating requests when appropriate and responding accordingly to a calling PMM. The general role of a PMM is shown in Figure. 1. The typical interaction in a decision-making process between a user, an RH, a PEP, a PEL and a PR is shown in Figure. 2 and is not elaborated further in this paper.

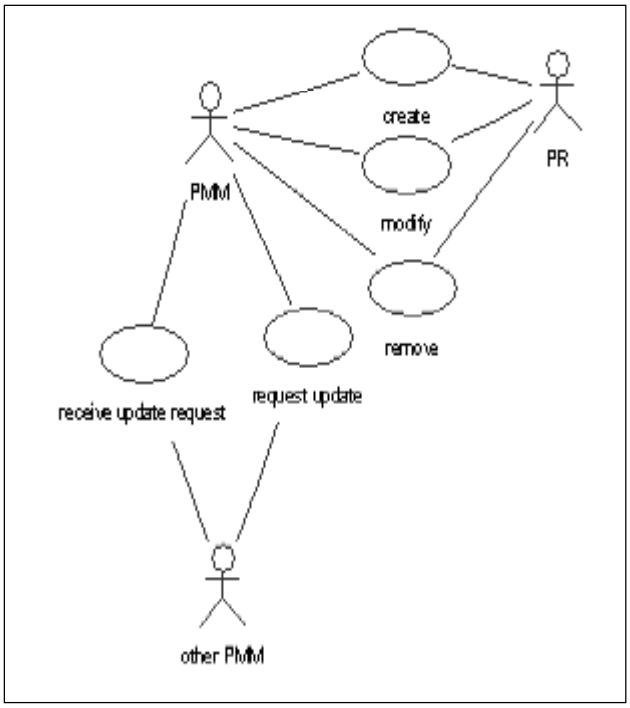

Figure 1. Role of a PMM

Our ePolicy framework will work in a centralized or distributed environment. In a distributed environment an RH can communicate with many PEP's, a PEP can communicate with many PEL's and a PEL can communicate with many PR's. A PMM will only ever communicate with PR's in its own jurisdiction directly, where necessary it will request changes on another PR via that PR's controlling PMM. We promote the design of a user friendly GUI-driven PMM to assist a maintainer with maintenance tasks. Meta-policies, ePolicy groups, ePolicies and ePolicy operations are never

permanently removed or overwritten during the development and maintenance process. They are time-stamped and stored in an archive repository. This facilitates version control and auditing of ePolicies and ePolicy operations.

When an ePolicy is archived, relevant ePolicy groups (or meta-ePolicies in smaller system) are updated to use the new updated ePolicy. A Policy Distribution (PD) component can be included in large or complex distributed systems that use many local and remote PR's. The role of a PD component is to keep track of PR locations. Each of the components is self-sufficient and as such can be maintained independently. Each component makes its behaviour available through a publicly accessible interface while the implementation of this behaviour is deployed separately. This means that clients do not need to be informed when the internal implementation of a component changes.

We have developed a supporting architecture for our ePolicy framework. Like ebXML [14], UBL [14] and other similar ebusiness-focused projects we make use of the Extensible Mark-Up Language (XML) [13] in our framework. Individual ePolicies, meta-ePolicies and ePolicy operations are represented using XML and their syntactical structure is represented using XML Schemas [13]. This is important from a maintenance perspective as it means that all applications only need to understand a single uniform format.

HTTPS is used to provide a secure channel for sending and receiving information between a users Internet browser and the RH. The RH delivers XHTML 1.0 compliant web content and presentation is provided through W3C validated Cascading Stylesheets (CSS) [13]. Session management and dynamic content creation are achieved using Java Servlet Technology [15]. The RH converts the user input information into an XML instance that complies with a Interaction strictly defined XML schema, the input data is then known as an eTransaction.

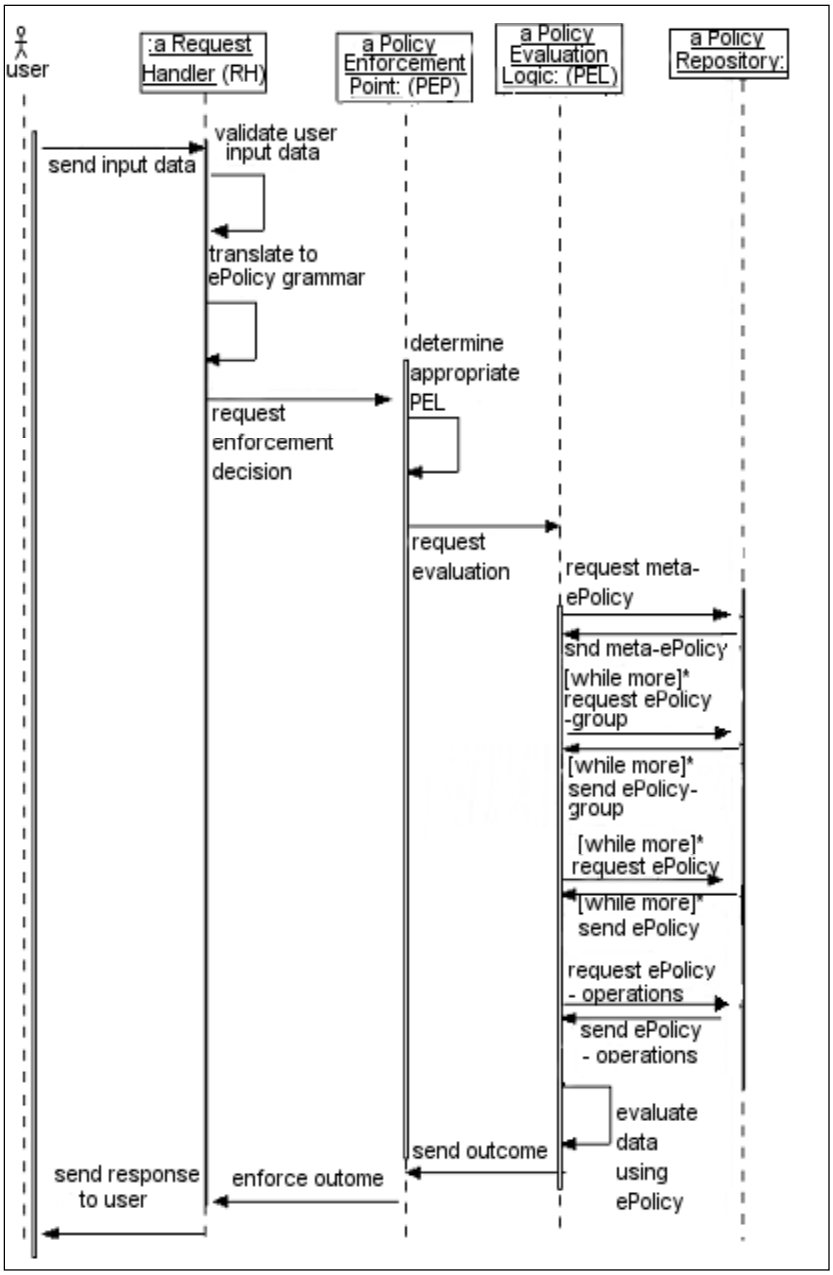

Figure 2. Policy Framework

We use XML as it is exportable, interchangeable and standard-based. The RH component is written in Java and uses Apache's Tomcat Servlet engine [16].

The PEP and the PEL are written in Java. Framework components use Java's Socket Technology to send and receive requests (except user requests, which use HTTPS. We have further defined the PEL into two separate components, a Policy Group Checker (PGC) and a Policy Checker (PC). The PGC interacts with the PR to get each of the ePolicies. It sends each of the ePolicies to the PC to be evaluated and the PC returns an outcome for each ePolicy to the PGC. The PGC uses the priority levels

associated with each of the ePolicies to reach an overall outcome. The PR uses the Castor Java-XML data-binding framework [17], to map the XML documents (eTransaction, ePolicies, meta-policies, ePolicy groups and ePolicy operations) to Java representations. XML documents are unmarshalled to Java objects and XML schemas are unmarshalled to Java classes. The Java objects are used in the evaluation process. When a new XML Schema is introduced our framework uses Castor to automatically generate Java classes to represent the elements in the schema. Each class generated includes suitable accessor and mutator methods and a default constructor. This can significantly reduce the amount of effort associated with introducing new data structures.

The PR uses Xindice [18] to store all the ePolicy and associated documents. Xindice is an open source project for storing XML documents. The PMM is a GUI-based component that interacts securely with the PR. The PR uses Java's Reflection package to dynamically determine the methods to invoke to send an ePolicy, ePolicy-group or meta-ePolicy. The GUI allows maintainers to easily retrieve and modify policies from the PR. As previously outlined, the original files are never overwritten or removed. When a maintainer needs to alter an ePolicy document, a copy of the document is made and dated and sent to an archive repository for version control and future auditing. In our prototype system we use the same Xindice server to store archived documents. When a maintainer submits files an updated document it is parsed to detect if it complies with its schema. If an error is found the maintainer is informed and requested to resubmit a correct document. This eliminates the possibility of an error or potential bug being passed to other components in the system.

We have identified the likely changes that would be required to web applications developed using our ePolicy framework to reflect changes in the application domain. We have determined that the types of change typical in our ePolicy framework are: a new ePolicy is introduced and the system must be updated to include this new ePolicy; an ePolicy operation associated with a particular ePolicy changes and the system must be updated to reflect the changed operation; a new ePolicy operation is introduced and the system must be updated to include this new operation; an ePolicy is moved to a Policy Repository in the same jurisdiction and an ePolicy is moved to a Policy Repository in a different jurisdiction.

When a new ePolicy is introduced into the framework the relevant meta-ePolicy must be updated to reference this change. No changes are required to any of the components (PEP, PEL, RH, PR, PMM). The PEL will automatically adapt to include the new ePolicy when the meta-ePolicy is next loaded. As ePolicies are highly organized structures and are very clearly defined, creating new ePolicies is a straightforward task and the level of human errors introduced by the maintainer should be reduced. As the PR will automatically check the structure of a recently added ePolicy the possibility of an error being passed to any other part of the system is eliminated. When an

operation associated with a particular ePolicy changes the ePolicy must be modified to refer to the new operation. This is straightforward as ePolicy operations conform to a well-defined systematic structure. No other component or ePolicy structure requires updating. The PEL will adapt to the new operation when the ePolicy using this operation is next evaluated.

Introducing a new ePolicy operation requires the most maintenance, in that a PEL using this operation must be modified, however, it will not require any new ePolicies to be created. A new ePolicy operation will have to be defined and any applicable PEL's will have to be updated to handle this new operation. As the PEL is designed using Object-Oriented techniques such as inheritance, the PEL can be easily extended to handle this new operation. As the PEL is a fully independent component no other component must be made aware of the changes. When an ePolicy is moved to a PR in the same jurisdiction the meta-Policy must be updated to reflect the new ePolicy location. In larger systems where a PD component is in place, it will need to be updated to reflect the new location. Finally, when an ePolicy is moved to a PR in a different jurisdiction the meta-ePolicy must be updated to reflect the new ePolicy location. In larger systems where a PD component is in place, it too will need to be updated to reflect the new location. The PMM will not require any change. The PMM only ever attempts to update ePolicies located in its own PR's. If it cannot find an ePolicy in one of it's local PR's it sends a request to the PMM at the location specified in the ePolicies meta-ePolicy and request's it to carry out an update.

Each of the above scenarios shows that our framework is sufficiently flexible to handle change. The ePolicy framework can be easily and efficiently used to develop and customize similar systems. The architecture is well defined and easy to deploy. The components can be independently developed and integrated. As the components are designed using functional decomposition techniques, the long-term maintenance effort required is reduced. ePolicies are stored in a centralized ePolicy store; therefore a developer does not need to be concerned with deploying multiple distributed policies. There is no need for a developer to understand many heterogeneous technologies or have any knowledge of proprietary languages in order to develop similar systems. As the PEL only uses a standardized interchangeable technology, such as XML, multiple communication formats do not need to be handled.

4. WORKED EXAMPLE

To provide an initial prototype for our framework we used a typical online application: an online wine distribution company operating in the USA. This scenario demonstrates different types of policies typical of this kind of trading: organizational policies, legal policies and social policies. For example, although various shipping methods may be available, the organizational ePolicy might only allow the use of insured courier; while legal policies might dictate that shipments can only be made to certain states or countries and a social ePolicy indicates that we should not allow a visibly intoxicated person to sign the delivery note. The online wine distribution company used in the prototype sells wine internationally and subsequently must take account of international policies.

Some examples of the ePolicies associated with the wine distribution company are outlined below.
A consumer can purchase a minimum of two cases and a maximum of eight cases of the same bottle of wine per transaction.
Under national and overseas law, wine cannot be sold to a minor. By placing an order on-line the purchaser is confirming that they are above the minimum age to purchase wine according to their national law.
The company distributes to the world with a single shipping method.

The company will not ship to 'bad' clients (clients with poor payment records), or clients with outstanding accounts. Clients have a 30-day credit period.
Shipping costs do not include any local duty or taxes that may be payable to import wine.
The company operates a replacement ePolicy based on credit per bottle returned against next order.

We have evaluated the maintenance effort required to change our prototype to reflect the five likely maintenance cases described previously and present a summary of each maintenance case next.

Case: A new ePolicy is introduced and the system must be updated to include this new ePolicy.
Description: To evaluate this case we introduced a new ePolicy that " no wine is shipped on Fridays as the company cannot control the storage of the wine over the weekend (shipping ePolicy)".
Discussion: A new ePolicy had to be created to represent this new policy instance. This is straightforward as an existing ePolicy (we only ship by an insured courier) has exactly the same structure and can be copied, edited and reused. As ePolicies are highly organized structures and are very clearly defined, a new ePolicy can easily be created and integrated even if it is not similar to an existing ePolicy. The meta-ePolicy for an order transaction must be edited to include this new ePolicy. This requires a new element to be added to the meta-ePolicy that outlines the location and priority associated with the new ePolicy however, this is easy as existing syntax can be re-used. No changes are required to any of the components (PEP, PEL, RH, PR, PMM). The PEL will automatically adapt to include the new ePolicy when the meta-ePolicy is next loaded. When the PEL requests the new ePolicy from the PR, the PR automatically unmarshalls the ePolicy to a Java object. This Java object will have the methods generated by Castor from its schema.

Case: An ePolicy operation associated with a particular ePolicy changes and the system must be updated to reflect the changed operation.
Description: To evaluate this case we changed the ePolicy "A consumer can purchase a minimum of two cases and a maximum of eight cases of the same bottle of wine per transaction" to "A consumer can purchase two and only two cases of the same bottle of wine per transaction".
Discussion: In this case the operation must change from an elementOf (a list) to equals and the ePolicy must be updated to reflect an equals operation. As the equals operation is clearly defined in a schema document this should be a straight forward. The PR will store the updated ePolicy and the old ePolicy will be archived. No other component or ePolicy structure requires updating. When the PEL requests the ePolicy from the PR, the PR will unmarshall the updated ePolicy to a Java object and this object will have the methods automatically generated by Castor that are associated with an equals operation. The PEL will automatically adapt to the new operation when the updated ePolicy is next evaluated.

Case: A new ePolicy operation is introduced and the system must be updated to include this new operation.
Description: To evaluate this case we introduced a new operation isValidDate into our system. This operation would be used to check if a date is within a valid range and could be used by an ePolicy that offered promotions or discounts for orders placed within certain dates.
Discussion: Introducing a new ePolicy operation requires the most maintenance effort. A new ePolicy operation will have to be defined to represent isValidDate and this operation will have to be added to the ePolicy operations structure. The PEL will have to be updated to handle this new

operation. As the PEL is designed using Object-Oriented techniques such as inheritance, the PEL can be easily extended to handle this new operation. In addition as the PEL is a fully independent component no other component must be made aware of the changes. When an ePolicy that uses this operation is next loaded the PEL will automatically handle the new operation and no further modifications will be required.

Case: An ePolicy is moved to a Policy Repository in the same jurisdiction.
Description: As our prototype only requires a single PR we hypothesis the maintenance effort involved in this task.
Discussion: The meta-ePolicy must be updated to reflect the new ePolicy location. No other changes are required.
Case: An ePolicy is moved to a Policy Repository in a different jurisdiction.
Description: As our prototype only requires a single PR we hypothesis the maintenance effort involved in this task.
Discussion: The meta-ePolicy must be updated to reflect the new ePolicy location. No other changes are required.

5. CONCLUSIONS

We presented a component-based framework that reduces the maintenance effort involved in evolutionary systems such as web applications. In our framework ePolicies and associated ePolicy operations are not embedded in the evaluation software but are stored centrally in an external store and are loaded as required. If an ePolicy changes, an update is only required once and at a centralized location. ePolicies are represented in a standard uniform format and as all components use this uniform format, maintainers do not need to understand or handle multiple data formats. We have shown how communication takes places between each of the components and have presented the technologies currently used to deploy this architecture. In addition we have provided a supporting example of an online trading store and have shown the minimum amount of maintenance effort to change the system to reflect changes in its domain.

6. REFERENCES

1. Comallen J. Modeling Web Application Architectures with UML. Communications of the ACM 1999; 42 (10):63–70.
2. Ginige A, Murugesan S. Web Engineering: An Introduction. IEEE Multimedia 2001; 8 (1):14–18.
3. Pressman R. What a Tangled Web We Weave. IEEE Software 2000; 17 (1):18– 21.
4. Bergin S, Keating J. A case study on the adaptive maintenance of an Internet application. Journal of Software Maintenance and Evolution 2003; 15 (4):245– 264.
5. Booch, G. Object-oriented Analysis and Design with Applications (OBT). Second Edition, Addison Wesley Professional: Boston, MA, 1994.
6. Zdun U. Reengineering to the Web: A Reference Architecture. Proceedings 6th European Conference on Software Maintenance and Reengineering, CSMR 2002. IEEE Computer Society: Los Alamitos CA, 2002; 164–173.
7. Qingshan Li, Jian Chen, Ping Chen. Developing an E-Commerce Application by using Content Component Model. Proceedings 36th International Conference on Technology of Object-Oriented Languages and Systems, TOOLS -Asia 2000. IEEE Computer Society: Los Alamitos CA, 2000; 275–284.

8. Capilla R, Duenas J.C. Light-weight Product-Lines for Evolution and Maintenance of Web Sites. Proceedings 37th European Conference on Software Maintenance and Reengineering, CSMR 2003. IEEE Computer Society: Los Alamitos CA, 2003; 53–62.
9. Warren P, Boldyre C, Munro M. The Evolution of Websites. Proceedings 7th International Conference on Program Comprehension, 1999. IEEE Computer Society: Los Alamitos CA, 1999; 178 –185.
10. Brereton P, Budgen D, Hamilton, G. Hypertext: The Next Maintenance Mountain. IEEE Computer 1998; 31 (12):49–55.
11. Gabriel, R. Patterns of Software -Tales from the Software Community. Oxford University Press Inc: New York, NY 1996.
12. Parsons D, Rashid A, Speck A, Telea A. A framework for Object Oriented frameworks design. Proceedings Technology of Object-Oriented Languages and Systems, TOOLS 1999. IEEE Computer Society: Los Alamitos CA, 1999; 141– 151.
13. World Wide Web Consortium: Extensible Hypertext Markup Language (XHTML), http://www.w3.org/MarkUp/; Extensible Markup Language (XML),http://www.w3.org/XML/Schema; Extensible Markup Language Schema (XML Schema), http://www.w3.org/XML/Schema; Cascading Style Sheets (CSS), http://www.w3.org/Style/CSS/.
14. Organization for the Advancement of Structured Information Standards: Extensible Hypertext Markup Language (XHTML), http://www.w3.org/MarkUp/; Universal Business Language(UBL),http://www.oasis-open.org/comittees/UBL; Electronic Business using Extensible Mark-up Language (ebXML), http://www.ebxml.org.
15. Sun MicroSystems: Java Servlet Technology, http://java.sun.com/products/servlet/.
16. The Apache Software Foundation. Apache Tomcat http://jakarta.apache.org/tomcat/.
17. ExoLab Group. Castor. http://www.castor.org/.
18. The Apache Software Foundation. Apache Xindice http://xml.apache.org/xindice/.