QUT Digital Repository:
http://eprints.qut.edu.au/

Karus, Siim and Dumas, Marlon (2007) Enforcing Policies and Guidelines in Web Portals: A Case Study. In *Proceedings WISE'2007 Workshops: Workshop on Governance, Risk and Compliance Management in Web Information Systems* 4832, pages pp. 154-165, Nancy, France.

© Copyright 2007 Springer
This is the author-version of the work. Conference proceedings published, by Springer Verlag, will be available via SpringerLink.
http://www.springer.de/comp/lncs/ Lecture Notes in Computer Science

Enforcing Policies and Guidelines in Web Portals: A Case Study

Siim Karus[1], Marlon Dumas[1,2]

[1] Institute of Computer Science, University of Tartu, Estonia
siim04@ut.ee
[2] Faculty of IT, Queensland University of Technology, Australia
m.dumas@qut.edu.au

Abstract. Customizability is generally considered a desirable feature of web portals. However, if left uncontrolled, customizability may come at the price of lack of uniformity or lack of maintainability. Indeed, as the portal content and services evolve, they can break assumptions made in the definition of customized views. Also, uncontrolled customization may lead to certain content considered important by the web portal owners (e.g. advertisements), to not be displayed to end users. Thus, web portal customization is hindered by the need to enforce customization policies and guidelines with minimal overhead. This paper presents a case study where a combination of techniques was employed to semi-automatically enforce policies and guidelines on community-built presentation components in a web portal. The study shows that a combination of automated verification and semantics extraction techniques can reduce the amount of manual checks required to enforce these policies and guidelines.

Keywords: web portal, customization, policy, guideline

1 Introduction

Continuing advances in web technology combined with trends such as Web 2.0 are generating higher expectations for user participation and customized user experiences on the Web [4]. These heightened expectations entail additional maintenance costs for community-oriented web sites, such as web portals. A natural way for web portal owners to balance higher expectations with the imperative of keeping a manageable cost base, is to “open up the box” by allowing the community to contribute content, services and presentation components into the portal. This way, the portal owner can focus on developing and maintaining the core of the portal instead of doing so for every service and presentation component offered by the portal. Also, increased openness and community participation has the potential of promoting fidelity, by motivating end users and partner sites to continue relying on the portal once they have invested efforts into customizing it or contributing to it. On the other hand, this increased openness needs to be accompanied by a sound governance framework as well as tool support to apply this framework in a scalable manner. Indeed, a manual

approach to reviewing and correcting user-contributed components would easily offset the benefits of accepting such contributions in the first place.

This paper considers the problem of allowing third parties to contribute presentation components to a web portal, while enabling portal administrators to enforce a set of policies and guidelines over these components in a scalable manner.[1] Specifically, the paper presents a case study where a team of web portal administrators needed to enforce a number of such policies and guidelines. Central to the approach adopted in this case study is a language, namely *xslt-req*, that allows portal administrators to capture the impact of policies and guidelines on the XML transformations that presentation components are allowed to perform. As a result, the portal administrators do not need to inspect and to fix every single submission in all its details; instead, most of the enforcement is done by a set of tools based on *xslt-req*. A key feature of *xslt-req* is that it builds on top of well-known web standards, specifically XML Schema and XSLT, thus lowering the barriers for its adoption.

The paper is structured as follows. Section 2 introduces the case study, including the policies and guidelines that needed to be enforced. Next, Section 3 discusses the techniques used to specify and to enforce these policies and guidelines. Related work is discussed in Section 4 while Section 5 draws conclusions.

2 Case Study: VabaVaraVeeb

VabaVaraVeeb (http://vabavara.net) is an Estonian portal for freeware.[2] A key feature of the portal is its high degree of customizability. Specifically, the portal allows third-parties to layer their own *presentation components* on top of the portal's services. Third parties may introduce custm-built presentation components for various features of the portal, such as the 'mailbox' feature, the 'user menu' feature or the 'statistics' feature. Once a third-party has registered a presentation component in the portal, they can re-direct users into the portal in such a way that users will consume VabaVaraVeeb's services through this presentation component. This allows third parties to loosely integrate services from VabaVaraVeeb with their own services at the presentation level, while enabling VabaVaraVeeb to retain some control over the delivery of its services. For example, a third-party web site that maintains a catalogue of security software, namely Securenet.ee, has added a presentation component on top of VabaVaraVeeb, to match its own presentation style. This way, users of Securenet are transferred to VabaVaraVeeb and then back to Securenet transparently, since the presentation style remains the same when moving across the two sites. The degree of customizability has been pushed to the level where the portal's services can be rendered not only through traditional HTML web pages, but also through alternative technologies such as XAML (eXtensible Application Markup Language) and XUL (XML User Interface Language). Presentation components are defined as XSL transformations [3] while the data delivered by the portal is represented in XML.

---

[1] In this paper, the term policy refers to a rule that must be followed and for which violation can be objectively defined, while the term guideline refers to a rule that should generally be followed, but for violations can not always be objectively asserted.

[2] The first author of this paper is one of the co-founders and administrators of this portal.

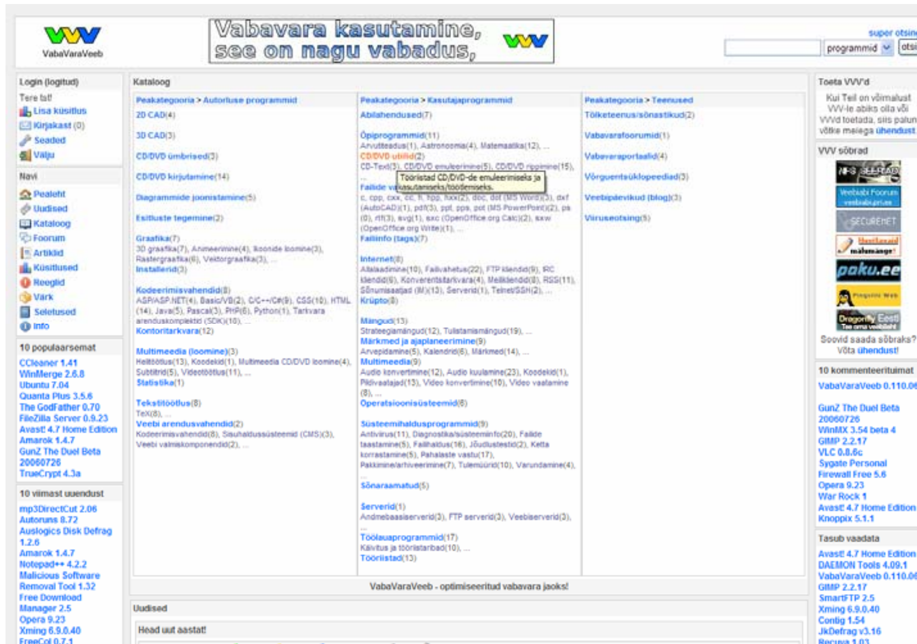

Figure 1 VabaVaraVeeb default interface.

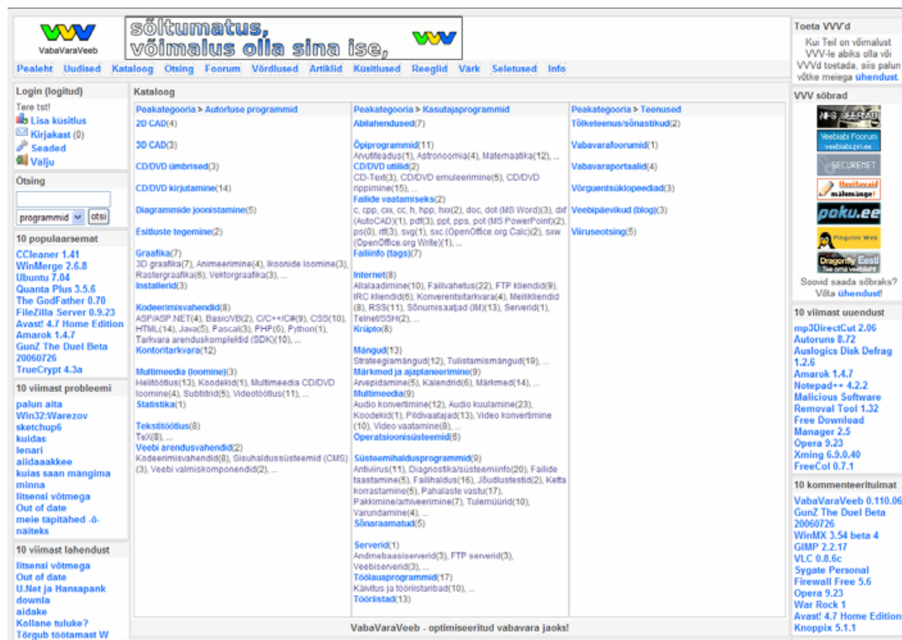

Figure 2 Modified VabaVaraVeeb interface with new main menu and other layout changes.

Figure 1 shows the default interface of VabaVaraVeeb while Figure 2 displays a modified interface. One can see that Figure 2 has a different main menu that appears at the top on the page (below the banner), and a repositioned right pane and search box. There are other less visible differences. For example, the presentation component corresponding to Figure 2 uses JavaScript to a larger extent than the default one.

Even though the portal gives significant freedom to third parties, all presentation components are required to conform to a set of *portal policies and guidelines* in order to achieve a certain level of uniformity and manageability. Originally, user-defined presentation components were manually verified for conformance against the portal policies and guidelines. However, as the portal grew, it became clear to the administrators that this manual approach would not scale up. Indeed, each contributed presentation component has to be checked against each policy and guideline, and each check is time-consuming. In addition, new versions of existing presentation components are submitted from times to times, and need to be verified again and again. This led to the need to automate the conformance verification of submitted presentation components against policies and guidelines.

The portal's policies and guidelines can be classified as follows:[3]

1. Certain content is mandatory and must always be presented to the users. Although the portal does not currently generate revenue through advertisement, it is foreseen this will happen sooner or later. Hence, it is important to ensure that paid advertisements and special announcements produced by the portal are always displayed, regardless of the presentation components in use. Some of this mandatory content must be presented to the users "as is", while other content may be presented in alternative ways. For example, promotional announcements may have different presentation requirements than other announcements.
2. Conversely, certain content must never be presented to the user. Presentation components are applied directly to the XML documents managed by the portal. Some information contained in these XML documents (e.g. user's access rights) may be sensitive or may only be needed by internal procedures. This information should therefore not be included in the generated pages.
3. Certain content must/may be delivered in certain output formats. For example, for banners we have both the URL of the banner and the URL of the advertised service. While rendering in graphical format, both URLs are marked compulsory, while in text mode the banner's URL is not marked as compulsory, however, the advertisement's alternate text is marked as compulsory.
4. Some content may need to be hidden or shown depending on the values of certain elements/attributes in the XML documents.
5. Styles should use existing or common controls when possible. This avoids duplicate code and contributes to forward compatibility [1], which is one of the design goals in VabaVaraVeeb.
6. Generic services must be preferred over internal components to expose similar aspects of objects or types of objects. Generic services are services used to perform common tasks like presenting simple dialogues or notifying about errors.
7. Complex services should be composed of individually addressable and "subscribable" services. Services built this way lower communication overhead

[3] For detailed specifications of these policies and guidelines, the reader is referred to [6].

and follow service-based approach [2] to enable forward compatibility. This service-based approach also allows portal owners to bill usage of every service separately (hence the requirement for "subscribability"). Users only need access to services and sub-services they are subscribed to.

8. The need for presentation components to access additional metadata to render a document should be minimized. On the other hand, the metadata already contained in the document should be used extensively. The names and values of XML elements often give valuable hints about their underlying semantics. For example, an element name with a suffix 's' in English, usually denotes multiple items, and this knowledge can be exploited to render the element's contents as a list. Also, in some cases, the XML document contains URLs and by inspecting such URLs, we can derive valuable metadata and use it for presentation purposes.

Rules 1 to 4 above correspond to policies while rules 5 to 8 represent guidelines. Guideline 7 does not relate to presentation components but to actual services provided by the portal. At present, third parties are not allowed to contribute such services, but it is foreseen that this will happen in the future, thus the guideline has been introduced and is being applied to all services internally developed by the VabaVaraVeeb team.

In principle, the conformance of a submitted presentation component against the first four policies can be automatically determined if the document structure is rigidly defined, i.e. not allowing unqualified nodes and nodes of type "any", and not allowing cyclic constructs in the document schema. Indeed, if the structure of the document is rigid, we can compute all possible source document classes that lead to different output document in terms of their structure and we can test the presentation component on sample documents representing each class. For each page generated by these tests, we can then automatically check if the policy is violated or not.

However, constantly evolving web portals can not rely on strict definitions of document structures as these definitions change too often and styles would need to be updated with every change. Due to this continuous evolution, document schemas must be designed in a forward-compatible manner by making them as loose as possible. This in turn makes the automatic enforcement of policies difficult. As explained below, we have found techniques to enforce these policies to some extent, but endless possible rulesets and document structures make it impossible to enforce in all cases.

Guideline 5 can be enforced by removing the users' ability to create custom basic controls. This may, however, result in lower performance of the solution as some simple tasks might have to be addressed using complex components. In some cases common controls are not present and have to be created beforehand.

Guideline 6 is difficult to enforce automatically as it requires detection of semantically similar code portions.

Guideline 7 is subjective as there is no metrics to decide whether or not a service should be divided into sub-services. It is still possible to use some metrics for approximation and compile-time warnings can be displayed at chosen value ranges.

By removing the ability for presentation components to access information in the portal – other than the presented document – we can easily enforce Guideline 8. However, if we enforced this guideline too strictly, we would lose forward compatibility. In Section 3.2, we discuss a technique to enforce this guideline while achieving forward compatibility, by following conventions in the naming of XML elements and exploiting these conventions to derive semantic information.

3 Defining and Enforcing Policies and Guidelines

To facilitate the enforcement of the policies and guidelines introduced above, several techniques are currently employed by VabaVaraVeeb's administration team. Central to these techniques is a language for capturing requirements over stylesheets, namely *xslt-req* [5]. Some of the guidelines are not crisply defined, so their enforcement can not be fully automated. Hence, other strategies are used to complement *xslt-req*. This section provides an overview of *xslt-req* and the techniques and strategies used for policy and guideline enforcement in VabaVaraVeeb.

3.1 The XSLT requirements definition language (xslt-req)

In the context of VabaVaraVeeb, restrictions over the transformations that presentations are allowed to perform are treated as a natural extension of restrictions over the structure of documents over which these transformations are applied. Accordingly, *xslt-req* was defined as an extension of XML Schema and the syntax of these extensions is similar to XSLT. In addition to providing an integrated framework for expressing document structure and allowed transformations, this design choice has the benefit that the portal developers and third-party contributors are familiar with XML Schema and XSLT, and it is thus straightforward for them to learn *xslt-req*.

The aim of *xslt-req* is to capture allowed and required data transformations. For each element, attribute or group in a schema, *xslt-req* provides extensions to specify:

1. whether the value of the element or attribute may be used in the output directly;
2. whether the value of the element or attribute may be used in the output indirectly;
3. whether the value of the element or attribute may be ignored; and
4. whether the values of the element's children may be ignored.

xslt-req also supports the specification of conditions that determine when should these rules be applied. These conditions are captured as XPath expressions over the source document. They may also depend on the requested output format. Additionally *xslt-req* can be used to limit the set of allowed output formats. Finally, *xslt-req* supports versioning and allows developers to explicitly designate the root element of the source document and to attach default policies to the document's nodes.

As mentioned earlier, *xslt-req* was designed to be easy to learn for developers familiar with XML Schema and XSLT. All *xslt-req* directives are in XML Schema *appinfo* sections. The similarity with XSLT can be seen in the following listing.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema  xmlns:xs="http://www.w3.org/2001/XMLSchema"
3 xmlns:xr="xslt-req" xr:schemaLocation="xslt-req.xsd">
4  <xs:annotation>
5    <xs:appinfo>
6      <!— Declaration of a new ruleset -->
7      <xr:xslt-requirements id="näide" version="1"
8           rootName="root" ignoreChildsDefault="false" />
9      <!— Output formats used in transformation rules-->
10      <xr:output-format name="xhtml" method="xml"
11           namespace="http://www.w3.org/1999/xhtml" />
12      <xr:output-format name="rss" method="xml"
13        namespace="http://backend.userland.com/rss2" />
```

```
14        <xr:output-format name="text" method="text" />
15      </xs:appinfo>
16    </xs:annotation>
17    <xs:element name="root">
18      <xs:complexType>
19        <xs:sequence>
20          <xs:element name="child">
21            <xs:annotation>
22              <xs:appinfo>
23                <xr:choose>
24                  <xr:when matchOutputFormat="text|rss">
25                    <!— Transformation rules -->
26                    <xr:transformation-rules
27               ignoreChilds="true" outputIgnore="true"
28             outputValue="false" outputCondition="false" />
29                  </xr:when>
30                  <xr:otherwise>
31                    <xr:transformation-rules
32             ignoreChilds="true" outputIgnore="false"
33             outputValue="true" outputCondition="false" />
34                  </ ...>
```

The *xslt-requirements* element in lines 7-8 of this listing specifies the version of the *xslt-req* ruleset, its identifier, and the name of the root element. XML Schema allows multiple root elements in a document. However, *xslt-req* requires that there is a single root element. This feature makes it easier to verify stylesheets. The *ignoreChildsDefault* boolean attribute specifies that unless elsewhere overridden, presentation component may not ignore child nodes of any element. Boolean attributes *outputValueDefault*, *outputConditionDefault*, *outputIgnoreDefault* specify other default values of *xslt-req* transformation rules, the meaning of which is explained below. By default, these attributes are false except for *outputValueDefault* which is true by default. Rules for unqualified elements and attributes can be specified in the *xslt-requirements* element as well. One can specify multiple *xslt-req* rulesets in one file by using different namespace prefixes.

Lines 10-14 specify the output formats affected by the ruleset. Output formats are given a name, which is used later to specify rules specific to that output format. The element for specifying output formats also includes an attribute called *method*, corresponding to the *method* attribute of XSLT's *output* element. Optionally, the URL to the document schema can be supplied with attribute *schemaLocation*.

Lines 23 to 34 illustrate the *choose-when-otherwise* construct. The syntax of this construct is defined in Figure 3. It is similar to XSLT's *choose-when-otherwise* construct. The main difference is that the element *when* in *xslt-req* contains an output format selector attribute, namely *matchOutputFormat*. Lines 24-29 formulate rules that apply only to "rss" or "test" output formats. Meanwhile, lines 30-34 formulate rules that must be followed in other cases. Element *when* also allows using attribute *test* as selector on the source document. This is similar to how XSLT elements *when* attribute *test* is used. It is possible to have multiple *when* elements (i.e. multiple selectors). It is also possible to use *choose* or *if* elements as children of *when* or *otherwise* elements (i.e. nested conditional statements are allowed).

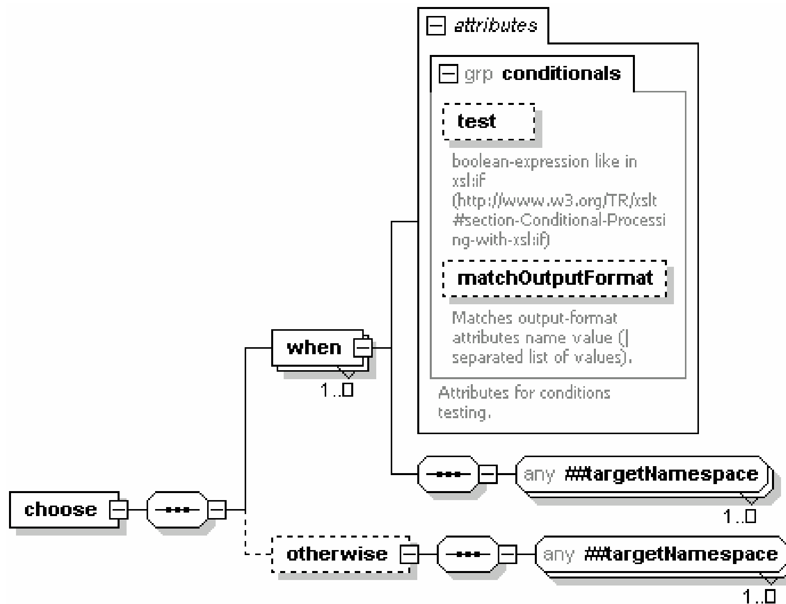

Figure 3 Syntax of *choose-when-otherwise* construct in xslt-req.

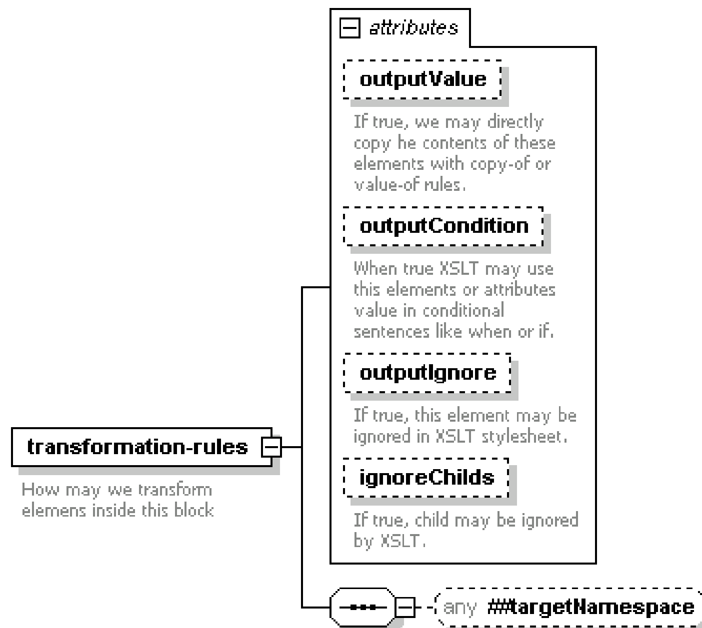

Figure 4 Syntax of transformation-rules in of xslt-req.

Lines 29-31 and lines 34-36 show examples of xslt-req transformation rules. The syntax of such rules is given in Figure 4. Attribute ignoreChilds states whether or not an element's children may be ignored. Attributes outputIgnore, outputValue and outputCondition state, respectively, whether or not the element or attribute may be ignored in the output, whether or not the nodes value may be included in the output (i.e. used in XSLT value-of or copy-of expressions) and whether or not the node's value may be used for formatting (i.e. used in XSLT when or if statements).

Even though the portal core is small and the structural descriptions and core services amount to less than 1% of the internal data structures, they have a significant impact on how presentations are built. This is the reason why about half the customizations made by users are to the presentation of the core services. Therefore, defining the internal data structures and using XML Schema with *xslt-req* to verify user stylesheets has significantly reduced the human workload of verifying stylesheets manually. This also allowed community-built presentation components to remain functional despite changes in the core services.

3.2 Exploiting implicit metadata

The workload for maintaining stylesheets was also lowered by using metadata implicitly contained in the source XML documents. In the early days of VabaVaraVeeb, the portal developers noted that the names of elements and attributes gave valuable hints for their presentation, and this was used to make the stylesheets simpler, forward-compatible and in line with the guidelines. In particular, the portal developers found they could use the suffixes of element names to detect lists of items. This enables the use of a single template for catalogue entries, search results, message lists and other list-like structures within the portal. When an element is identified to be a list, special attributes attached to this element are used to display information like the total number of items, the number of displayed items, the list name, page number (for multi-page lists) and buttons for navigating to previous and next page. The names of these special attributes tend to follow certain naming conventions. If some of the special attributes are missing, the features expressed by the missing attributes are not displayed. Using this simple detection made it possible to implement a presentation component for the search feature of one of the modules of the portal, and then reuse this presentation component for other modules. Another usage scenario for this technique is detecting modules themselves to display them in a special area of the page. Also, user input areas can be identified similarly for presentation purposes.

Initially, the portal administrators applied this technique to simplify their stylesheets and to increase reuse. After successful trials, third parties were also encouraged to use implicit metadata when contributing presentation components. This was achieved by emphasizing the transient nature of any internal structure and by supplying examples of detection rules.

This technique, in conjunction with the hard-coded requirement that all user-contributed stylesheets must have a fall-back to the default style, has made it possible to enforce policy number 8 and indirectly helped to enforce policies 5 and 6. This technique is, however, applicable only manually. It is interesting to note that users apply this technique naturally in about 80% of cases.

This technique is not error fail-proof. Rare oddities of general rules can be expressed by using templates with higher priority level than the priority level of the general templates thus ensuring that specialized templates are chosen over general ones. This kind of ‘overriding’ has been used in several cases to fine-tune the detection rules. For example the template for displaying lists of messages overrides the general template for rendering lists.

3.3 Additional techniques

Additional techniques included combining similar code snippets and updating the portal’s core to manage services in an addressable and subscribable way.

Similar code snippets were identified automatically using code profiling scripts that collect information on certain portions of code. The collected information included function parameters or code block input values, code block length in lines of code (with and without comments). This information was used to review code blocks that were similar according to the profiler script output. If these code blocks did turn out to have similar functionality, they were extracted and generalized functions were made to replace these code blocks. The profiler found only a small portion (less than 0.5% of lines of code) of the whole code-base to have similar code blocks. Of these, less than half were false positives. This technique is now used for all new services.

The new version of the portal’s core features improved support for tracking service access. This allows one service to be responsible for tracking service usage and therefore manage subscriptions to services. As the portal core manages the addressing of portal services, services only need to handle their internal addressing and subscription to their sub-services.

3.4 Outcomes

The guidelines put into effect have reduced the complexity of the presentation layer significantly. Management of all the presentation components (i.e. styles) has become a task requiring minimal manual intervention. And because the guidelines are designed to ensure forward compatibility, styles written once can be and are often used to display new features and services without modification. This has led to shorter development effort for adding new features into the portal, since these new features tend not to break the existing styles, and when they do, the source of the problem is easier to find. Furthermore, guideline 8 has reduced the number of lines of code in the presentation layer by almost half, lowering the human effort for verifying styles.

The impact of automatic tools used to verify the styles, based on *xslt-req*, is as high as that of using code examples and suggestions on how to follow the guidelines. Also, deviations from policies which are not automatically verified are often accompanied by deviations from policies that are automatically verified. This might be caused by the different levels of experience in programming among the contributors.

It is estimated that the cost of maintaining the portal has been reduced by more than 50% thanks to the contributions from third parties, the enforcement of the policies and guidelines, and the lowered costs of enforcement.

4 Related Work

There is extensive literature dealing with the enforcement of access control policies on XML content [7]. Policy definition languages proposed in this area allow one to attach access control policies to an XML document node and its descendants. In this sense, these policy definition languages share commonalities with xslt-req. However, they differ in several respects: First, access control policy languages focus on capturing under which conditions can a given XML node be read or updated by a user. Thus, they cover cases such as the one in the second policy outlined in Section 2. In contrast, they do not allow one to capture obligations such as “a given element must be displayed” or “an element must be displayed only in certain formats”, both of which are key features of xslt-req. Nevertheless, xslt-req may benefit from ideas in [7] and in similar work, to improve its ability to capture access control requirements.

There is also significant literature related to enforcing accessibility and usability guidelines on web sites. For example, Vanderdonckt & Beirekdar [8] propose the Guideline Definition Language (GDL) which supports the definition of rules composed of two parts. The *structural* part designates the HTML elements and attributes relevant to a guideline. This part is expressed in a language corresponding to a limited subset of XPath. The second part (the *evaluation logic*) is a boolean expression over the content extracted by the structural part. In contrast, xslt-req operates over XML documents representing the internal data managed by the portal, so that policies and guidelines are checked before the HTML code is generated.

The work presented in this paper is also related to the integration of services (possibly from multiple providers) at the presentation layer. This integration is supported by various portal frameworks based on standard specifications such as Java Portlets or WSRP [10]. More recently, Yu et al. [9] have proposed an event-based model for presentation components and a presentation integration “middleware” that enables the integration of services from multiple providers at the presentation layer without relying on specific platforms or APIs. However, these frameworks do not consider the enforcement of policies and guidelines as addressed in this paper.

5 Conclusion and Future Work

The paper discussed various techniques for enforcing guidelines in community-built web portals. Some of these techniques involve automatic verification of user-contributed components, others were merely suggestive, meaning that they give suggestions to the portal administrator regarding potential deadline violations, but still, the administrator has to manually verify the suggestion. The usefulness of automatic verification techniques was found to be at the same level as that of suggestive techniques. Therefore, these latter techniques should not be undervalued. Even though there are still verifications that require human intervention, much of the enforcement occurs before the components reach the portal administrators.

The automatic verification techniques can be costlier to the portals maintenance team as these techniques usually required human proofing or solving of the problems. Even if the output of automatic verification were presented directly to the authors of

presentation components, (s)he might not be able to understand and solve the problems reported without help from the portals developers.

Less than half of policy and guideline deviations detected by automatic verification techniques were false positives, verification against *xslt-req* does not lead to false positives due to its design. Automatic verification was used to detect less than 30% of all identified types of policy/guideline deviations. However, this 30% of types of deviations contained the most common deviations experienced.

The techniques presented in this paper have room for improvement along several directions. For example *xslt-req* could be extended to support the specification of rules based on patterns or XPath expressions. This way, *xslt-req* could be applied to more than just the static core structure. This would make it possible to allow all business layer services to have mandatory content or hidden content. In addition to extending *xslt-req*, the verification methods that use xslt-req need to be reviewed, as they currently assume that the document base structure is rigid.

As discussed in Section 3, templates that automatically extract semantics from XML element names and values have been successfully used to achieve forward-compatible stylesheets and to enhance reuse, despite the fact that these techniques are not fail-proof. Making these techniques more robust and studying their applicability in a wider setting is an avenue for future work.

References

1. C. Armbruster: *Design for Evolution*. White paper, 1999. Available at: http://chrisarmbruster.com/documents/D4E/witepapr.htm, 1999
2. K. Bennett, P. Layzell, D. Budgen, P. Brereton, L. Macaulay, M. Munro: Service-Based Software: The Future for Flexible Software. In *Proceedings of the 7th Asia-Pacific Software Engineering Conference (APSEC)*, Singapore, December 2000, pp. 214-221. IEEE Computer Society.
3. J. Clark (Editor): *XSL Transformations (XSLT)*, W3C Recommendation, 1999. http://www.w3.org/TR/xslt
4. M. Jazayeri: Some Trends in Web Application Development. In *Future of Software Engineering (FOSE'07)*, May 2007, pp. 199-213. IEEE Computer Society.
5. S. Karus: *Kasutajate poolt loodud XSL teisendustele esitavate nõuete spetsifitseerimine (Specifying Requirements for User-Created XSL Transformations)*. Bachelors Thesis, Faculty of Mathematics & Computer Science, University of Tartu, Estonia, 2005. http://math.ut.ee/~siim04/b2005/bak1.0_word2.doc (in Estonian).
6. S. Karus: *Forward Compatible Design of Web Services Presentation Layer*. Masters Thesis, Faculty of Mathematics & Computer Science, University of Tartu, Estonia, 2007. http://www.cyber.ee/dokumendid/Karus.pdf/
7. I. Fundulaki and M. Marx: Specifying Access Control Policies for XML Documents with XPath. In *Proceedings of the 9th ACM Symposium on Access Control Models and Technologies (SACMAT)*, Yorktown Heights, NY, USA, June 2004, pp. 61-69. ACM Press.
8. J. Vanderdonckt and A. Beirekdar: Automated Web Evaluation by Guideline Review. *Journal of Web Engineering* 4(2): 102-117, 2005.
9. J. Yu, B. Benatallah, R. Saint-Paul, F. Casati, F. Daniel, M. Matera: A Framework for Rapid Integration of Presentation Components. In *Proceedings of the 16th International World Wide Web Conference (WWW)*, Banff, Alberta, Canada, May 2007. ACM Press.
10. C. Wege: Portal Server Technology. *IEEE Internet Computing* 6(3): 73-77, 2002