Aspects of Service-Oriented Component Procurement in Web-Based Information Systems

Martin Gaedke, Johannes Meinecke, Martin Nussbaumer

IT-Management and Web Engineering Research Group
{gaedke,meinecke,nussbaume}@tm.uni-karlsruhe.de
Fax: +49 (721) 388097
University of Karlsruhe, Institute of Telematics
Engesserstr. 4, 76128 Karlsruhe, Germany

Keywords: Web Engineering, Components, Web Services, Foreign Code, Trust

Abstract. Modern information systems are increasingly built on Web-based and component-based platforms. This raises the need for a service-oriented infrastructure to simplify the management and procurement of corresponding components. Special focus lies on the deployment and distribution of such software artifacts within the context of the World Wide Web to promote their reuse and therefore to save development costs. At the same time, the integrity of the overall system must not be neglected. The use of components from third-party vendors poses a potential security threat requiring additional care. Furthermore remains the issue of usage rights for the components and the data they provide. Flexible mechanisms can offer a huge range of different licensing models to be enforced on the runtime process. This paper presents an approach to deal with these challenges together with an implementation of a software system supporting component-based Web portals.

1 Introduction

Since the beginning, the World Wide Web has dramatically changed the way we access information. Its growing audience has been supplied with a growing pool of data from a wide range of different topics. Being itself a huge information system, it has become the technological fundament for software architectures denoted with the term Web-based information systems. These can take the form of Web portals that act as information access gateways for a global or limited audience. Technically, such portals are Web applications with the primary functionality of searching and displaying data, possibly originating from databases or Web services.

The development of Web applications is a relatively new and in many cases not a very systematically conducted process. Problems related to these issues are covered by the young discipline Web Engineering (Deshpande, et al., 2002; Weippl, et al., 2002). A principle especially relevant to Web-based information systems due to the recurring functionality of providing information access is the use of components. As a cost-saving alternative to individual implementations of different portals, these can also be constructed from pre-built software artifacts with the help of Component-Based Web Engineering (CBWE). Portal frameworks like the Web Composition Service Linking System (WSLS) (Gaedke and Nussbaumer, 2004) offer the necessary platform for configuring a portal’s Web pages and the visual components they contain without the need for programming a single line of code.
Common definitions of components, as given in (Szyperski, 1997), characterize them especially as units of deployment. Therefore, they need to be supported as such by a suitable software system. In other words, they should be installed as a unit (not as individual files) within a controlled and well-defined process that simplifies the overall task and makes use of modern Web technologies.

This paper describes an approach to the deployment and exchange of portal components as a support for Web Engineering. An investigation of existing approaches is followed by a discussion of open issues like security implications and the need for licensing mechanisms. As a solution to these problems, the Pretoria system as well as an exemplary implementation is presented.

2 Support for Components in Web Engineering

2.1 Existing Approaches

As stated within the last section, the development of Web-based information systems requires a systematic approach provided by Web Engineering and benefits especially from composition. This is not an entirely new idea and is already realized by a range of existing systems. It can be distinguished between several different approaches concerning the way composition is supported.

One possible solution involves tool support at development time. Platforms like Java or .NET offer dedicated programming models for components in Web applications. The implementation is supported by corresponding development environments (e.g. Sun ONE Application Framework, Microsoft Visual Studio .NET). With the help of their graphical tools, Web pages can be constructed from components simply by placing them via Drag&Drop and adjusting dialog boxes. The individual parts can range from simple improved Web page controls to complete applications. Environments allow the integration of third party components into their tools by offering installation or registration mechanisms. Even though it is not imperative to program any source code, the resulting applications still have to be compiled and deployed on a Web server. This approach causes a number of disadvantages. The perhaps most severe one is the loss of the compositional character at runtime: After compilation, the parts that were independent at design time cannot be distinguished anymore, nor is it possible to add or adjust components without rebuilding the system. Hence, the solutions lack flexibility with respect to the evolution and the possible requirement changes they may become subject to. Also, the exchange of components between developers lies outside the scope of these systems. Although developers can share each other’s solution parts, the act of copying and transporting the files which make up the components has to be conducted manually, e.g. via email, FTP, HTTP etc. The involved effort poses an unnecessary burden and hinders component propagation. Moreover, due to the absence of security procedures, there is no guarantee that the incorporated components internally behave as they are supposed to. This represents a risk for the component-consuming parties and in some cases may cause them to rather develop their own software.

More advanced solutions regarding the flexibility at runtime can be achieved by component-oriented and component-based portal frameworks. Examples for implementations in this field are Microsoft SharePoint Services (Microsoft, 2004) and PHP-Nuke (PHPNuke, 2004) among others – like in many Content Management System (CMS), administrators of corresponding portals are offered a Web interface allowing them to edit their pages. Modifications are not restricted to text content and comprise the insertion and adjustment of components acting as small Web applications. These systems come with a set of supplied components serving a general purpose, like e.g. event calendars or news boards. Additionally, they also support the integration of independently developed software artifacts. The process of deploying these new components is in many cases relatively complicated. Rather then simply uploading the necessary files via a Web interface, site administra-
tors need to login at operation system level and perform a number of manual tasks. This complexity stands in the way of a widespread use (and therefore reuse) of foreign components and can still be subject to improvement by automation. Some systems offer rudimentary support for the protection of authoring rights. During the installation process of SharePoint Services components for example, the acceptance of an End User License Agreement (EULA) is enforced. Support does however not extend over processes that involve the producer directly in terms of user registration or component procurement.

As an alternative to the software component approach, portals can also be extended by using Web Services. This idea is pursued for example by the Web Services for Remote Portlets (WSRP) specification (Kropp, 2004). In this case the functionality being subject to composition is exported into Web services (so called Portlets). Unlike conventional Web services, Portlets do not only provide XML data, but also the markup code necessary for visualization. Fig. 1 outlines the way a WSRP page is generated. A central portal server processes incoming page requests by delegating a part of the received input to remote Portlet servers. The hosted Portlets generate and return the output necessary for the parts of the page they are responsible for. After having aggregated the results from all services, the central server assembles the page and adjusts the design using a standardized Cascading Style Sheet (CSS) concept.

![Fig. 1: Distributed rendering of a Web page with WSRP](image)

The Portlets are usually controlled by a third party and can be integrated into multiple portals at the same time. Due to the absence of any code to be transported, the component installation process is made obsolete. Instead, the Web services to be used only need to be registered. The protection of copyrights is simplified by the fact that the provider of the functionality keeps control of who is accessing the service. As another result of this architecture, the data used by the component is kept along with the code outside the sphere of influence of the portal operating party. In the context of the described work, this can pose a serious problem concerning trust and security, which needs to be considered as a crucial aspect of information systems (Mouratidis, Giorgini and Manson, 2003). The providers of a Web information system supplying personal information concerning health, for example, will not want to rely on a technology that forces them to let external parties manage their private data. In some cases, this is even impossible because of restrictions by law.

Another aspect of component-based Web applications is covered by repositories that support component propagation and publication. These repositories serve as centralized or distributed storage places for a pool of readily programmed software. Aggregating a large number of components increases the range of possible applications but also raises the problem of finding suitable solutions out of a broad supply (Frakes and Nejmeh, 1987). Therefore, most systems focus on the process of searching and retrieving components. The CBWE repository introduced in (Gaedeke and Rehse, 2000), for example, supports searching by tagging its components with extendable metadata used for categorization and representation. Other systems, like the Agora Repository (Seacord, Hissam and Wallnauin, 1998), work as specialized forms of search engines that provide access to a large
pool of available components (e.g. Java Applets) from the Web. The procurement of a component respectively the technical process of retrieving and installing a component into a portal is, however, usually outside the scope of support. An integrated solution could combine the exchange with the deployment and thus simplify the overall process.

2.2 A selection of unsolved Problems of the Component Context

As the examination of the state of the art revealed, common component-based systems in the Web fail to sufficiently support the process of managing the software artifacts as units of deployment. This concerns particularly the exchange of the components between the Web information systems where they are used. A seamless and easy to use process is however fundamental for achieving a high degree of reuse. In order to enable potent solutions, open and distributed repositories can provide a broad supply of prefabricated building blocks. If components are transported within system support, and not e.g. manually via email, it is also easier to guarantee communication security. Involved issues in this context include the prevention of manipulation and theft, as well as the assured identification of the exchanging partners.

Among the important characteristics mentioned by established definitions of the term software component as given in (Szyperski, 1997) is their possible third-party origin. Applied to the scenario at hand, this means that the component producer and the party running the information system do not necessarily coincide. Furthermore, the software is usually distributed in binary form without source code. On the one hand, the components need to access the servers resources and data stored e.g. in file systems or databases. On the other hand, the portal owning party has no means of monitoring or controlling these activities in traditional systems. In order to counteract the potential threat of unwanted use of resources like the retrieval of confidential data, a robust environment with corresponding security mechanisms could greatly improve this situation. Discovering threats early, ideally during the installation process, results in stable portals with fewer problems at runtime. Another requirement would be a flexible control of access constraints taking into account the origin of the executing code. The degree of trust towards the code producers may vary and should reflect in the set of permissions granted to the code units. For example, portal providers might decide to use their own components for sensitive tasks like user data management that require exclusive access to certain files. Portals like SharePoint Services provide limited protection against untrusted code but do not offer any form of check during component installation.

The distributed deployment of foreign components does not only require trust towards the code supplying party but also towards the code demanding party. With the existence of systems supporting component propagation, developers might be interested in keeping track of the places where their property is in use. They might also want to charge a financial contribution from the portals. As customary with other software products, the installation of components could be combined with a registering and license-issuing process. Also, to make sure that these mechanisms cannot be circumvented, some form of copy protection could be provided. In addition to that, the licensing could be used to control the access to external data sources, possibly limiting their total number or enforcing a pay-per-access policy.

3 Pretoria Approach: A disciplined Process towards Component Procurement

As an answer to the presented problems inadequately solved by the state of the art, a set of concepts was developed and integrated within the Pretoria approach. The component deployment was combined with the exchange, with special emphasis on an appropriate treatment of foreign code and envisioned a flexible licensing mechanism.
3.1 Component Exchange

For the purpose of this work, the term exchange is used to describe the complete process necessary for a component installed on one Web portal to be transported and deployed on another. To support the overall process, a distributed and scalable approach was chosen. The subjects of exchange are not stored at a central location, but stationed at the individual portal servers. A registry serves as a directory containing the component’s properties and locations. Hence, it was possible to apply the same infrastructure as used for Web service discovery and integration also to the deployment of components.

Fig. 2 contains the steps necessary for a component exchange between two Web information systems. At first, an administrator decides to make a certain component available, as it serves a sufficiently general purpose and could easily be integrated into other information systems. She publishes an entry at a central registry (1). For that purpose, a Universal Description, Discovery and Integration service (UDDI) (Januszewski and Mooney, 2002) was integrated. UDDI is traditionally used for publishing Web services but is intended to be applied for more general purposes. In this case, it contains data about all participating systems and their public components. Other administrators can use the searching capabilities of UDDI software (2) for locating the required components referenced within the registry. They retrieve the published metadata (3) and may select individual components for installation. With the information contained in the UDDI entry, the supporting software contacts a Web service using SOAP (Mitro, 2002), which belongs to the providing Web information system, in order to request the component (4). This service disposes about the information which components are intended for public distribution. In case of a valid request, the component is issued as a result to the caller (5) where it can instantly be installed into the information system (6). The process is mostly transparent to the administrator, she does not have to deal with any technical details concerning the acquiring and handling of a component.

Additionally to SOAP, some of its extensions from the Web Service Enhancements (WSE) (Powell, 2003) are used to enable and secure the component request and transport via a Web service. The WS-Attachment protocol (Barton, Thatte and Nielsen, 2000) for example allows attaching large binary data sets, in this case the component files, to the XML-encoded SOAP message. With the
help of WS-Security (Nadalin, et al., 2003), security mechanisms are integrated into the inter-portal communication. Digital message signatures help to identify the parties using this public system interface and ensure that the components are not modified during transport. Also, encryption can be applied in scenarios where the exchanged software is not meant to be publicly available and must be protected from potential theft.

Fig. 3 depicts the data model of the information stored at the central registry as a UML class diagram. The model was designed to map with the metamodel prescribed by the hosting UDDI service. Its central entity is the Component, which corresponds to a UDDI service. To provide for a better overview, components can be tagged with Categories realized with the help of UDDI categorization schemes or UDDI tModels. Every component can be assigned to a Portal entry stored as a UDDI service provider. Among other attributes, this entry also contains the address of the portal’s retrieval service, which can be used together with the component identifier to locate and acquire a component. Portals that can take part in the exchange process are associated with a special service provider, the Portal Group.

![UML class diagram](image)

**Fig. 3: Logical data model used at the component registry**

### 3.2 Protection against Foreign Code

As mentioned above, the threat by foreign code in portal components is still a largely unsolved problem and requires special support within corresponding architectures. Fig. 4 lists a number of steps to be performed during installation and at runtime that help to establish a robust code environment. As shown, producers are to include code signatures into their components before they are released. This means that the component files themselves contain checksums that are digitally signed with private keys. As a result, manipulations of the binary code, for example during transport, can be noticed by anyone knowing the matching public key and one potential threat can be ruled out. The signature also serves as a characteristic that can be used to uniquely identify the code producer and allocate execution rights to the component accordingly. The installation process represents the system entrance of the partially trusted code and contains a number of security checks. The new files are analyzed and rejected if conflicts are discovered. Subjects to be checked include the set of referenced software modules, which may provide access to security-relevant functionality, or the namespaces used by the code artifacts. Additionally, metadata statements about the behavior of the component at runtime are reviewed. Programmers can issue such statements to indicate for example, which resources their code requires at runtime. This alone fails to sufficiently protect the server, as there is no guarantee that the authors actually declare all accesses correctly. Therefore, the execution of the installed code is subject to additional security control at runtime. In case of any violation of security rules, execution is interrupted and an exception is raised. To support the runtime environment of the portal in this eventuality, functionality to associate the exceptions with the causing components is supplied. This allows the environment to react with further actions like deactivating the faulty element or notifying an administrator.
Both the preliminary check during installation and the access control at runtime are controlled by a flexible security policy, which is calculated as demonstrated in Fig. 5. To determine the set of permissions available to a certain component, a group of security domains is evaluated (Abadi and Fournet, 2003). Domains provide rules for hierarchical scopes, e.g. for a company, for a division within that company and finally for a single server. For each one, a tree of rules is specified, where individual rules consist of a condition combined with a set of permissions. Conditions usually concern the origin of the component, as for example the public key used to sign it or the location on the server where it has been installed. The permissions are related to the resources which a component is allowed to access, like e.g. write access to a certain directory or the privilege to use a network connection. The tree is traversed starting at the root. If a condition evaluates to true, the corresponding rights are collected and the procedure continues with the subnodes. If not, the whole subtree is ignored. This recursive traversing enables the security check to distinguish between special cases formulated in varying levels of detail. For every domain, the union of collected permission sets is determined. In order to guarantee that a component complies with all relevant scopes, only the intersection of all domains is issued, resulting in the final permission set that provides the base for runtime and installation time control.

Fig. 4. Security-relevant steps at installation time and at runtime

Fig. 5. Sample security policy calculation
3.3 Integrated Component Licensing

With systems supporting a controlled portal component deployment, it is possible to combine the installation process with related functionality. Apart from the already introduced security check, an integrated licensing procedure must supply the solution to the problem concerning the (copy-) rights of the component producer. The Pretoria approach offers a flexible strategy open to different licensing models and providing protection against unauthorized use.

Fig. 6 depicts the flow of events leading to the issue, transport and acceptance of a license during the installation via a Web interface. The process is triggered by a user uploading a new component (1). Once it passed the security check, a defined interface is used to query the software whether it requires a license for proper use and where that license can be obtained (2). The installing user is then redirected to the supplied URL of the license service, and at the same time a license request is passed via HTTP (3). The service is actually a Web application, possibly belonging to the component producer, enabling the user to supply registration data about the planned deployment (4). This information can be fed into a registration database or included into a license. After completion, the license service redirects the browser back to the installing Web page while transporting the issued license (5). The internal structure of the license is unknown to the installation routine, e.g. allowing support of X.509 (Housley, et al., 2002) or XrML (Contentguard, 2002). After its transport, the document is passed on to the component, which contains code for validating its correctness (6). When this has been established, the installation process can continue and the license is stored for further use (7). At runtime, the component can demand to see it any time by using an appropriate interface (8). In case of circumvention of the licensing by copying component files, this runtime check could prevent the software usage. The license could also include descriptive data about the server, allowing the detection of replicated license files intended for other places of deployment. The approach is generally open to many different licensing models. Information systems could for example be extended with components that provide access to some data source via a Web service and that are controlled by a license to restrict the number of accesses.

![Fig. 6. License-issuing cycle](image)

The general communication mechanism of the licensing concept is based on the Passive Requestor Profile (Bajaj, et al., 2003) and the WS-Trust (Hondo, Melgar and Nadalin, 2003) specification. The central idea of WS-Trust is concerned with the exchange of items, called security tokens, making security-relevant statements. The tokens are issued by dedicated Web services, the security token services, and usually proof a certain identity or access privilege. The Passive Requestor Profile describes how one or more security token services can be used to realize a distributed authorization...
and authentication strategy for controlling accesses on resources that are requested via a Web browser. In the case of the Pretoria approach, this can be applied to the licensing process, where licenses are understood as special forms of security tokens that make statements about usage rights. Fig. 7 shows how the profile’s sign-on procedure, which usually redirects anonymous users to a login site for authentication, maps to a request for a license.

Fig. 7: License requesting mechanism in correspondence to the Passive Requestor Profile

By basing the license transport on an existing specification related to identity and access management, Pretoria also takes advantage of the security aspects of the involved communication protocol. Consequently, the messages exchanged during the browser redirections can be digitally signed and encrypted with WS-Security methods similar to the ones that are applied during the component exchange process. Hence, the participating information systems can e.g. be shielded from man in the middle attacks that could otherwise steal licenses within the relatively open and distributed architecture.

4 Supporting System for the Pretoria Approach

To realize the presented concepts, the Pretoria System has been implemented to support the deployment, administration and exchange of Web application components. The Microsoft .NET Framework was chosen as the technological platform providing the implementation model for the supporting software tool as well as for the supported components. The implementation takes advantage of the .NET support for component-based Web applications (ASP.NET Web controls), its code access security and the already existing support for Web service security. The Pretoria system is available for download at http://mw.tm.uni-karlsruhe.de/projects/pretoria. The following sections describe the general architecture as well as some selected implementation details concerning the realization of the security mechanisms.

4.1 Architectural Overview

Fig. 8 depicts the general architecture of the cooperating system elements and the involved user roles. The system was built to support Web information systems that are realized as portals based on Web application frameworks. The framework defines the type of components to be integrated as
well as how these can be instantiated at runtime to be placed and used on the portal’s Web pages. Although WSLS was chosen as the primary target framework, the implementation was designed for flexibility to support any portal working with ASP.NET components. The units of deployment usually consist of the dynamic link libraries containing the binary program code, template files containing the static part of the component’s markup code, and other auxiliary files, like e.g. images or XML data files. The generic approach taken even covers the case in which the installed files actually realize a Web service. Given the necessary permissions, the components can access various data sources, including their own files, local data sources of the portal (e.g. a database) or external data sources (e.g. Web services). The core of the implementation lies in the administration tool, a Web application supporting component administrators with their work (installing new components, managing existing ones and participating in the exchange process). The tool hides the technical details involved in the component handling process. Additionally, there is the role of the system administrator, who is responsible for specifying the XML-based configuration. Decisions involve the security policy to be applied to the components at installation time and at runtime, as well as the adoption of the tool to different portals and frameworks. The exchange of components with instances of administration tools on other portal servers is supported by the integration of a component registry (based on UDDI) as described earlier. For scalability, the registry can also be provided by a group of UDDI servers that synchronize with each other to give bottleneck-free access to large numbers of component entries. The license service represents the interests of the component producer within the architecture as a foreign Web application to be included into the installation process. As a proof-of-concept, a simple service was implemented which issues license documents that are bound to specific portal server IP addresses and that contain signatures created with X509 certificates.

Fig. 8. Architecture of the Pretoria implementation

4.2 Realization of message security

The distributed nature of the Pretoria architecture causes communication between the various system parts, which needs to be secured against potential threats. As mentioned above, the Pretoria approach relies on WS-Security as the security technology applied for the component exchange and the license issuing process. The implementation of the support system is based on the Web Service Enhancement add-on for the .NET Framework and prioritizes manageability of the security deci-
sions. Therefore, instead of hardwiring the mechanisms into the program code, these have been separated with the help of WS-Policy (Hondo, et al., 2003). As a result, the WS-Security requirements for the exchanged messages are specified in standardized XML-documents, where they can be changed later on at runtime with dedicated tool support. Corresponding documents at the sender’s and receiver’s side make sure that the requirements are enforced automatically during Web service calls. The listing in Fig. 9 shows an example taken from the configuration of the component exchange, demanding that the component-requesting party has to supply a signature to proof its identity. The necessary X509 certificates for creating and verifying the signatures are taken from the operation system’s certificate store, where they can be imported and managed with standard operation tools.

```xml
<?xml version="1.0" encoding="utf-8"?>
<policyDocument xmlns="http://schemas.microsoft.com/wse/2003/06/Policy">
  <mappings xmlns:wse="http://schemas.microsoft.com/wse/2003/06/Policy">
    <mapDefault policy="#policy-b895dfe4-a4b6-42ab-9b12-9c01b6e0019f" />
  </mappings>
  <policies xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
    <wsp:Policy wsu:Id="policy-b895dfe4-a4b6-42ab-9b12-9c01b6e0019f" xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy">
        <wsse:TokenInfo>
            <wsse:TokenType>wsse:X509v3</wsse:TokenType>
            </SecurityToken>
        </wsse:TokenInfo>
        <wsse:MessageParts Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
          <wsse:Body />
        </wsse:MessageParts>
      </wsse:Integrity>
    </wsp:Policy>
  </policies>
</policyDocument>
```

Fig. 9. WS-Policy document specifying the WS-Security restrictions to the component transport

### 4.3 Realization of resource access control

One major aspect of the Pretoria approach concerns the safe integration of binary code originating from a third-party. Conceptually, this means that not only the users of the software are subject to trust and control, but also the code modules themselves. Therefore, the implementation must account for that and protect the system’s resources at runtime. The situation is complicated by the fact that resources can either be accessed directly or indirectly by calling methods of intermediate modules. In order to provide runtime protection, the presented supporting system relies on the .NET code access security principle. The diagram in Fig. 10 depicts an example of an installed component using a resource via two other code modules. The .NET runtime environment, which loads the modules into memory, takes care of determining the corresponding permission sets by evaluating a policy that can be mapped to the rule trees in section 3.2. At design time, the rights that are necessary to use individual methods, classes or entire modules can be declared within the source code. Declarations can either state that only the direct caller must be authorized, or that the entire stack of code parts involved in the call possesses the necessary rights. When the code is executed, the .NET environment enforces the restrictions and throws an exception in case of any violation. A custom exception handler integrated in the portal software is used to react to the security problems that are not intercepted by the components themselves.
5 Distributed Deployment on Multiple Portals – A Concrete Example

The following paragraph demonstrates the capabilities of the Pretoria system by presenting a real world scenario taken from the Mobile University project at the University of Karlsruhe concerning the deployment of a component on several Web information systems realized as portals based on WSLS. As an example, a component for managing and querying a lecture directory via a Web interface is discussed. To provide the entries, an external Web service is used as the data source. Consider the following scenario, which is also depicted in the sequence-diagram as shown in Fig. 11:

Fig. 10: Example of a controlled indirect resource access

Fig. 11: Overview of Scenario
The component is first implemented as a standard ASP.NET control being part of a Web application, tested and exported into a deployment unit. During the first installation attempt by an administrator, a security problem is discovered concerning the usage of Web services. After talking to the developer, the administrator changes the security policy to grant Web access to all components signed with the developer's private key. This enables the installation routine to pass the security test and to redirect the administrator to the licensing service, where he registers and initiates a digital payment transaction to buy the full usage rights. From now on, the component is installed on his server, where it can be managed with the help of the Pretoria administration tool and used on the WSLS portal pages. After a few weeks of stable usage, the responsible administrator decides to publish the lecture directory component. Fig. 12 shows the corresponding functionality within the Pretoria administration tool as well as the resulting entry at the UDDI server. Later, another portal provider uses his tool to search for components belonging to the category teaching. After reviewing the published information, he selects the component for installation. During the licensing process he chooses to request only a trial license first. This allows him to use the component for four weeks, after which he buys a full license.

6 Conclusion and Future Work

This paper was concerned with Web portals and Web-based information systems built out of components to increase the reuse of code. A number of existing approaches were outlined with respect to deployment, procurement and reuse and pointed out the lack of security measurements against foreign code as well as the need for controlling the use of components with licenses. This in mind, A concept for supporting the deployment, administration and exchange of portal components was developed that integrates answers to the shortcomings of the state of the art. The solution makes use of modern Web service technologies. It has been implemented in conjunction with the WSLS framework and is in use within the context of the Mobile University project. The authors are now planning to develop new license mechanisms, e.g. based on Digital Rights Management (DRM). Furthermore, future implementations might cover other component models beyond the current one, possibly EJB or SharePoint Web Parts.
Acknowledgement

This material is based on research for the Mobile University (M-University) project supported by Microsoft Research.

References


