Security and Licensing for Components of Web-based Information Systems

Martin Gaedke¹, Johannes Meinecke¹, Martin Nussbaumer¹

Abstract

Modern information systems are increasingly built on Web-based and component-based platforms. This raises the need for a Web-based 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 [5, 20]. 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) [7] 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 [19], 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.

¹IT-Management and Web Engineering Research Group, Institute of Telematics, University of Karlsruhe, 76128 Karlsruhe, Germany
This paper describes our work on the deployment and exchange of portal components as a support for Web Engineering. We begin with an investigation of existing approaches. This is followed by a discussion of open issues like security implications and the need for licensing mechanisms. We present a solution to these problems and the Pretoria system as an exemplary implementation.

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 the usage of software development environments like the Sun ONE Application Framework or Microsoft Visual Studio .NET. With the help of 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. One of the main disadvantages of this approach lies in 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. Also, the exchange of components between developers lies outside the scope of these systems.

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 [13] and PHP-Nuke [17] 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, but also support the import of independently developed software artifacts. The process of deploying the new components is in many cases relatively complicated, as it is not integrated into a Web interface. 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.

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 [12]. In this case the functionality being subject to composition is exported into Web services (so called Portlets). These generate the markup code for their part of the page and send it to the requesting portal, which then aggregates the individual parts to a Web page. With the absence of any code to be transported, the component installation process is made obsolete. As another result of this architecture, the data used by the component is kept along with the code outside the control of the portal operating party. In the context of our work, this can pose a serious problem concerning trust and security, which needs to be considered as a crucial aspect of information systems [15]. 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 due to restrictions by law.

Another aspect of component-based Web applications is covered by repositories that support component propagation by 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 [6]. Therefore, most systems focus on the process of searching and retrieving components. The CBWE repository introduced in [8], for example, supports searching by tagging its components with extendable metadata used for categorization and representation. The procurement of a component respectively the technical process of retrieving and installing a component into a portal is, however, not supported by the system. 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

Among the important characteristics mentioned by established definitions of the term *software component* as given in [19] is their possible third-party origin. Applied to our scenario, 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, we developed a set of concepts integrated within the Pretoria approach. We combined the component deployment with the exchange, emphasized an appropriate treatment of foreign code and envisioned a flexible licensing mechanism.
3.1 Component Exchange

For the purpose of this work, we use the term *exchange* 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 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.

![Diagram of component exchange process]

**Fig. 1.** Deployment, procurement and (re)use of components within the Pretoria approach

**Fig. 1** 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, we integrated a *Universal Description, Discovery and Integration* service (UDDI) [11]. 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 (service providers) and their public components (services). 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 [14], 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) [18] are used to enable and secure the component request and transport via a Web service. The WS-Attachment protocol [3] 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 [16], security
mechanisms are integrated into the inter-portal communication. For example, digital message signatures help to identify the parties using this public system interface.

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.

![Diagram](image-url)  
**Fig. 2. Security-relevant steps at installation time and at runtime**

Fig. 2 lists a number of steps to be performed during installation and at runtime that help to establish a robust code environment. As shown, code producers sign their components digitally before they are released. As a result, unnoticed manipulations of the binary code, for example during transport, 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. This includes a review of the metadata statements about the behavior of the component at runtime. 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 checks during installation and the access control at runtime are controlled by a flexible security policy, which is calculated as demonstrated in Fig. 3. In order to determine the set of permissions available to a certain component, a group of security domains is evaluated [1]. Each domain specifies a tree of rules consisting of a condition concerning the origin of the component combined with a set of permissions. The tree is traversed starting at the root. If a condition evaluates to true (e.g. if a component was signed with a certain key), the corresponding rights are collected and the procedure continues with the subnodes. If not, the whole sub-
tree is ignored. For every domain, the union of collected permission sets is determined. But only the intersection of all domains is finally issued to the component, allowing the definition of several policies with varying restrictiveness.

![Sample Security Policy Calculation](image)

**Fig. 3. 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. We present a flexible strategy open to different licensing models and providing protection against unauthorized use.

![License-Issuing Cycle](image)

**Fig. 4. License-Issuing Cycle**

*Fig. 4* 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 compo-
ponent 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). This mechanism is based on the Passive Requestor Profile [2] and the WS-Trust [9] specification. The idea of Security Tokens as items making security-relevant statements is applied to licenses as documents making statements about usage rights. The internal structure of the license is unknown to the installation routine, e.g. allowing support of X.509 [10] or XrML [4]. The Passive Requestor Profile ensures a secure and standardized form of communication between the license service and the Web information system.

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.

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. We chose the Microsoft .NET Framework 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-based access security and the already existing support for WSE and UDDI.

Fig. 5. Architecture of the Pretoria implementation

Fig. 5 depicts the general architecture of the cooperating system elements and the involved user roles. The Web portal was built using the WSLS framework, but the Pretoria system is designed to be adaptable to any portal working with ASP.NET components. 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). Additionally, there is the role of the system administrator, which is responsible for specifying the configuration. Among other decisions, he defines the security policy to be applied to the installed components. 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. 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.

The Pretoria system is available for download at http://mw.tm.uni-karlsruhe.de/projects/pretoria.

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, we discuss a component for managing and querying a lecture directory via a Web interface. 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. 6:

![Sequence Diagram]

**Fig. 6. 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 (1). 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 digi-
tal payment transaction to buy the full usage rights (2). 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 (3). Fig. 7 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 (4). After reviewing the published information, he selects the component for installation (5). During the licensing process he chooses to request only a trial license first (6). This allows him to use the component for four weeks, after which he buys a full license.

Fig. 7. The Publication of a component with Pretoria and its appearance at the UDDI server

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. We looked at a number of existing approaches 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, we designed a concept for supporting the deployment, administration and exchange of portal components 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.

We 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


