A Web Engineering Approach to Model the Architecture of Inter-Organizational Applications

Johannes Meinecke, Martin Gaedke, Martin Nussbaumer

University of Karlsruhe, Institute of Telematics,
IT-Management and Web Engineering Research Group,
Engesserstr. 4, 76128 Karlsruhe, Germany
{meinecke,gaedke,nussbaumer}@tm.uni-karlsruhe.de

Abstract. During recent years, the World Wide Web (WWW, Web) has increasingly been used as a platform for applications that link together processes both between and across organizations. Acting as distributed components, Web Services provide a standardized way of externalizing functionality on a global scale and as such enable accesses that transcend organizational boundaries to form federated applications. The design and evolution of these federated applications is now imposing new obligations for the disciplined engineering of composed Web solutions. To meet these obligations, we extend the WebComposition idea, which is an approach to apply component-based software development concepts on Web applications. This extension facilitates modeling the complex landscape of the components and services building the federated applications. In this context, we introduce the WebComposition Architecture Model that serves as a map to keep track of the interrelations between the federated partners in terms of the involved Web-technology. Among the modeled artifacts are Web services, Web Applications and organizational zones of control that are all subject to evolution in the sense of the WebComposition approach.

1 Introduction

The rise of network technologies has enabled many fields of applications and ways to support businesses. Initially applied to connect only distributed components within single companies, it is nowadays increasingly used to link together systems of multiple enterprises for the purpose of cooperation and federation. The resulting applications are therefore characterized by their component-oriented architecture and distributed nature.

This trend has also been influenced by recent advances in Web technologies, particularly in the field of Web services. In the context of enterprise integration, the service can take the role of the component in a distributed system by exposing functionality and data through defined interfaces and standardized Web protocols. The value they provide can then be used in applications from within as well as from outside the company. In order to facilitate their orchestration to higher-level services, solutions can rely on business-process engines that control the execution flow of multiple services based on formal descriptions of the process.

The task of making the envisioned federated scenarios work successfully raises a diversity of issues to be addressed. Beyond the many questions related to the integration of business-processes, technical solutions for the operation and evolution of the system landscape have to be found. Approaches like e.g. the UML Profile for Enterprise Distributed Object Computing (EDOC) [14] focus the modeling of collaboration architectures on various levels of granularity, but understand the component as a logical concept. While such viewpoints are certainly helpful for describing systems on an abstract level, they are less appropriate for dealing with the concrete technical details that are important for operating and evolving a solution. With the mentioned technologies involved, this problem is actually a matter of systematically building Web-based solutions, i.e. the discipline *Web Engineering* [4]. As a sub-discipline, Component-Based Web Engineering (CBWE) supports especially development processes that have to cope with evolving Web applications composed of re-usable parts (in this case: the Web services). By focusing the principle of *reuse*, new applications can be created from already existing components (i.e. development with reuse), that are constructed in a separate process (i.e. development for reuse). During the lifecycle of the federated systems, evolution leads to the creation of new (re-)usable services, possibly belonging to new partners that join the federation. To effectively support its lifecycle in a controlled and systematic manner, the composed system has to be specified with evolution in mind first. This relates e.g. to the parts of the composition, their places and the way they can be accessed (i.e. used). Furthermore, it has to be stated how the relationships between the cooperating partners (i.e. trust) at business level are reflected by the solution on a technical level. When such system descriptions are integrated into a dedicated model, a guideline for evolution can be accomplished.

In the following section, we first look at a number of related approaches to modeling distributed, federated systems. Section 3 adopts the view of the *WebComposition* approach [5] on the problem at hand. We then propose the WebComposition Architecture Model (WAM) as an answer to the demand for modeling federated service-based landscapes in section 4. Following that, Section 5 describes the application of the model in a real-world project. Finally, we conclude with a short summary in section 6.

2 Related Approaches

The general idea of supporting systems composed of services throughout their lifecycles can be related to a number of already existing efforts in different disciplines. In this section, we describe three approaches related to our work. The Dynamic System Initiative and the Data Center Markup Language follow similar ideas concerning system modeling, whereas the Enterprise Application Integration concept is more focused on federating software systems on various levels

2.1 Dynamic System Initiative

The Dynamic System Initiative (DSI) is a technological strategy devised by Microsoft that aims at an integrative support for the design, deployment and operation of dis-

tributed systems [12]. The focus is centered on software products based on the Windows platform and is not confined to Web applications. The initiative is driven by the idea of combining the two processes of building and operating IT solutions to emphasis the application life cycle as a whole. This is realized by the introduction of the System Description Model (SDM), which is used to mirror the solutions throughout their life cycles. At design time, the SDM aids the developers planning their products on different levels of abstraction. At deployment time, the model enables an automatic installation of a distributed system as one unit. At runtime, it serves as a source of information for maintenance tools.

Descriptions based on the SDM consist of layers that can be defined independently from each other by different authors. The number of used layers can vary between individual solutions. The subjects of the *application layer* include ASP.NET Web applications and services as well as databases. On the *application host layer*, the description refers e.g. to instances of database and Web servers. Below that, the *network topology and operation system layer* models the operation system, as far as its functionality and settings are affected, and the connections between the elements of the distributed system. Finally, the *hardware layer* describes computers and further devices with regard to the hardware requirements of the other system entities. The initiative also includes a product roadmap for supporting systems to be delivered. The overall approach is not platform-independent and does not specifically target scenarios of federated applications.

2.2 Data Center Markup Language

Similarly to DSI, the DCML open industry initiative (Data Center Markup Language) aims at modeling IT infrastructures of individual companies [13]. The major focus lies on the application of XML-based standards for specifying data center environments, dependencies between data center components and the policies governing management and construction of those environments. This allows e.g. for creating descriptions of abstract server prototypes that cover all relevant aspects for setting up a particular type of machine, including the operation system configuration, network parameters and hardware requirements. By instantiating such abstract descriptions, servers that have a special dedicated purpose can be (re-)produced with a minimal set of manual steps. The sum of the descriptions make up a blueprint of the entire data center that reflects the system as it currently exists (or as it is supposed to exist in the future) in order to keep up with its changing shape and composition.

Unlike the DSI, DCML offers a more platform-independent view: The modeling of the data centre environments is founded on the definition of an ontology describing the problem domain by using XML and OWL. Moreover, it offers a bottom-up perspective on a system, emphasizing the infrastructure rather then the hosted business applications.

2.3 EAI

Enterprise Application Integration (EAI) [9] can be understood as an effort that aims at connecting single information systems in order to enable them to exchange data and support a common process. Its primary concern lies in integrating systems within one enterprise, but it basically applies the same techniques that are also used to achieve business-to-business integration. Typical scenarios include especially the integration of complex systems like large Enterprise Resource Planning (ERP) applications, Customer Relationship Management (CRM) applications, or various legacy applications. There are many different approaches to reach the general objective of EAI. A possible distinction can be made between the levels at which the individual strategies try to bind together the separate applications.

As a relatively technical approach that does not directly involve the processes themselves, *information-oriented integration* connects information sources like databases to allow the information flow between different applications. Some solutions accomplish this by simply replicating the data. Others combine several databases to make them appear as one logical database. When the data sources are accessible via defined interfaces, integration can also be realized by connecting these with adapters.

A more profound approach in terms of business processes is taken by the *Service-oriented integration*. The general strategy is concerned with the reuse of methods that expose business logic within or between companies. This allows for building composite applications that aggregate the functionality of other remote applications to achieve solutions of a higher value. Technical realizations include mechanisms based on Web services or distributed objects standards, like e.g. CORBA or COM+.

The objective of *business-process-oriented integration* is to combine systems at a high level of abstraction by targeting the process itself. Technical integration methods are based on a specification of the company's relevant processes, which have to be defined in advance. The model serves as a description of how the involved sub processes and supporting systems are related to each other. With the help of middleware, this information is then used to glue together the software systems. The interaction can e.g. be controlled by event-driven mechanisms. When this strategy is used, sub systems trigger events that are then interpreted by the middleware and lead to the invocation of other system parts. Business-process-oriented integration is supported by standards like ebXML or BPEL4WS.

Unlike the other strategies, *portal-oriented integration* does not try to establish an exchange of information between system back-ends. Instead, the users are offered a single Web interface (a Web portal) that allows them to access multiple applications in a uniform way through a Web browser. The implementation of the portals is often realized with the help of application servers that also provide connectors to the back-end systems. As a higher form of integration, *portal federation* [8] strives for bringing together different autonomous portals, allowing the realization of new application scenarios in a distributed and self-organized manner.

3 Evolution Aspects of Component-Based Web Applications

One of the well known problems of Software Engineering for the World Wide Web lies in the fact that its resource-based (document) implementation model was never truly intended for the kind of complex applications that are in use today. During the design of Web-based applications the entities handled by the designers are often defined at a much higher resolution than possible in the actual code produced during the development process. Component Based Software Engineering (CBSE) [11] (i.e. the construction of software from existing components) has been around for about three decades. CBSE is said to allow the construction of more complex software at lower costs. It is supposed to lead to easier maintenance and evolution (i.e. a higher flexibility of a software product throughout its entire life cycle), as well as an overall increase of quality if performed systematically. The *WebComposition* approach adopts these concepts by providing dedicated models that allow creating Web-based applications from components. It bridges the gap between design and implementation by capturing whole design artifacts in components of arbitrary granularity. The resolution of a component is not preset, but can vary depending on the level of detail required by the design concept in question.

The requirements for a software system change as time goes by. It is obvious that many kinds of influences are responsible for this, e.g. new regulations, changes in corporate identity or an extension of functionality. Such maintenance tasks are difficult to handle, if we did not design the application with the possibility of future changes and extensions in mind. Therefore, the *WebComposition* approach focuses on the evolution of Web-based applications by reusing components. The process consists of three main-phases that are applied in an iterative way throughout the application lifecycle. They are derived from the common phases of software process models, but take the principles of the Web into account and address concepts of software reuse. The process model follows a spiral consisting of evolution analysis and planning, evolution design and the execution of evolution [7]. The first phase deals with common problems in strategic planning of the application's functionality respectively with Domain Engineering. *Domain Engineering* has been described as a process for creating a competence in application engineering for a family of similar systems [15]. The last two phases reflect the two different views towards reuse: consumer view (*development with reuse*) and producer view (*development for reuse*).

To support and enforce a disciplined and manageable evolution of a Web-based application in the future, it makes sense not to design the initial application on the basis of the concrete requirements identified at the start of the project. Instead the initial application should be regarded as an empty application that is suitable for accommodating functionality within a clearly defined evolution space. We denote this initial application with the term *evolutionbus,* which serves as the glue (or starting point) for all abstract application domains of a Web-based application (cf. Fig. 1).

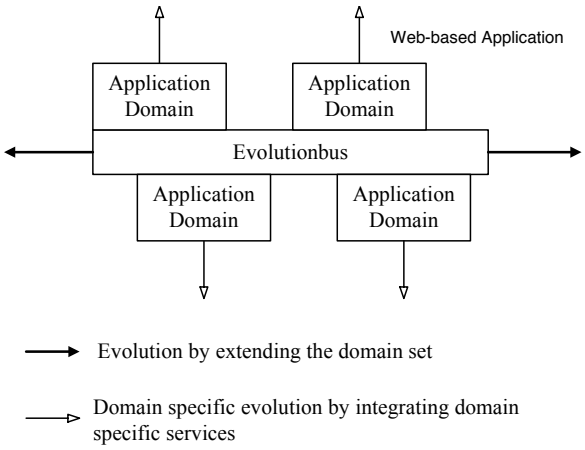

Fig. 1: The dimensions of a single Web-based application's evolution space

The evolutionbus enables the management and collaboration of domain-components, i.e. components that implement specific application domains such as Web-based procurement, reporting, or user driven data exchange. These domain-components (called *Services* within the WebComposition approach) may also be reused in future application domains. The evolution can take place in two clearly defined ways:

- *Domain specific evolution (vertical evolution)* – The extension of a domain through new services, e.g. by prototyping or referencing an existing service of a domain. Another possibility is that the domain itself changes or that it receives more functionality.
- *Evolution of the domain set (horizontal evolution)* – The evolution of an application is also possible by adding a new application domain that is created through composing of already existing services. This modification of the domain set extends the application functionality, as for example in the case of a shopping basket that is added to a Web-based product catalog.

Adopting the idea of linking together systems of multiple enterprises for the purpose of cooperation and federation (cf. section 1), it becomes obvious that a third form of evolution exists. This form focuses on reuse in the large by benefiting from already existing application domains in other evolutionbuses belonging to partner organizations, as depicted in Fig. 2.

- *Evolution with partner domains (federated evolution)* – The evolution of an application by adding existing autonomous, remote application domains or services, which remain under the control of partner organizations. This form of evolution extends the application functionality in a horizontal or vertical evolution manner without any local changes.

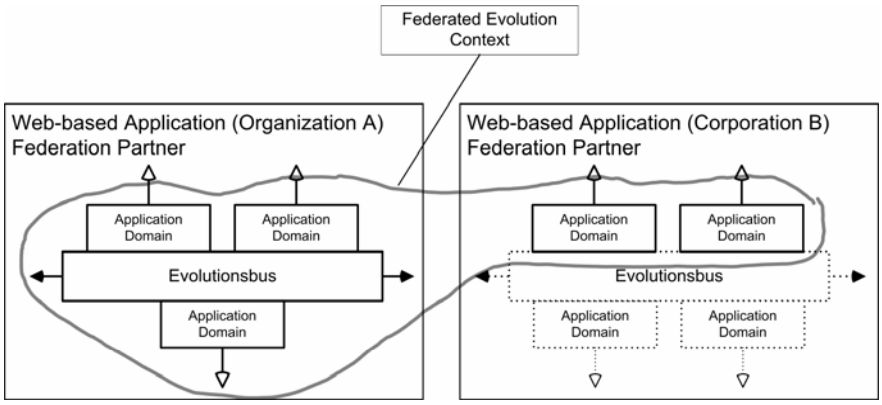

Fig. 2: Federated evolution across multiple Web-based applications (cooperating inter-organizational applications)

The evolution with partner domains adds additional levels of complexity to the already challenging task of modeling Web-based applications, as it has to cope with a lack of control over individual autonomous system parts. This demands for dedicated support that reduces the complexity by allowing the seamless specification of distributed and federated aspects. Therefore, in the next section, we introduce the WebComposition Architecture Model, which maps our abstract findings on evolution and federation to a concrete technological level.

4 WebComposition Architecture Model

Today, a common approach to composing enterprise applications from components distributed across organizational boundaries is to employ Web service technologies. By exposing data and functionality via Web service interfaces, companies can invoke each others services or create higher-level services to support entire business processes. However, in order to make such solutions operable, further considerations beyond merely providing and consuming services are necessary. First, the partners have to agree on the exact communication protocols to be used. Additional security protocols as well as firewall rules need to be established to enforce a safe invocation of services between different zones of control. The federated nature of such applications also raises identity management issues [17]. This has resulted in various specifications like e.g. WS-Federation [3], SAML [10] or the Liberty Alliance Project [1]. One of their key concepts is centered on the idea of distributing the access control process. This is achieved by the introduction of specialized Web services for authenticating anonymous requestors (*identity provider* or *IP*) and for authorizing the access to protected resources (*security token service* or *STS*). The full potential of this distributed approach is reached, when multiple services of different organizations are set up to trust each other. For example, the STS of a car manufacturing company can be configured to accept tokens from the IP of a part supplier. Hence, employees who have an account at the supplier's company can be authorized to view process-relevant infor-

mation at the manufacture's intranet portal. This form of federation avoids the need for multiple accounts for the same identity and thus lowers the cost of administration. Although the use of federation standards can solve many problems, it does not help to reduce the complexity of the overall solution without the proper modeling approaches.

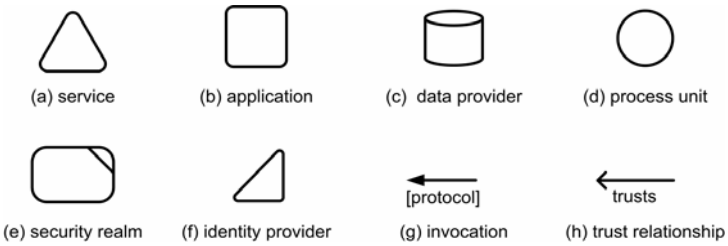

Fig. 3: Overview of the symbols for the most important modeling elements

In this context, we propose a model with a UML-like notation, the WebComposition Architecture Model (WAM), for capturing the most relevant aspects of the characterized systems. This covers in particular the participating services and applications, the access paths between them as well as the restrictions imposed on these accesses. Fig. 3 contains the symbols for the main modeling elements.

a. The *services* represent the system's distributed components that originate from the different involved organizations. They typically take the form of SOAP Web services that expose their functionality through a defined interface. It can be distinguished between atomic services, which provide basic, reusable operations, and composite services, which invoke other services to perform their work. One way to support such compositions is to rely on business process engines and use a formal process description to control the execution.
b. Users normally only interact with the overall system through the interfaces provided by *applications*. Within the model, this term relates to Web applications or portals that are accessed via standard Web protocols through a browser. In correspondence to the other model elements (e.g. the security realm), they can either be realized as internet or intranet applications, possibly requiring its users to authenticate.
c. In cases where it is useful to distinguish between the services and the underlying systems that serve as the actual data sources, this can be modeled with a separate *data provider*. This symbol is connected to the service or application with an undirected line. Data providers are not necessarily Web-capable themselves, as e.g. a database or a legacy system, for which the service acts as a wrapper.
d. Connected systems that perform functionality beyond data management are represented with a *process unit* symbol. This covers e.g. software that performs computations or triggers processes outside the modeled scope.
e. Services and applications can be enveloped by *security realms*, which are depicted as big rounded rectangles with their name in the top-right corner. These groupings reflect the organizational zones of control over networks, hardware and

software systems. Technically, their boundaries result e.g. from Web server configurations or firewall rules. When the systems employ the mentioned federation specifications (WS-Federation etc.), the realm also functions as a common identity and access management context. This means, there is exactly one designated security token service per realm that issues the tokens necessary for accessing the realm's resources. Moreover, this implies a common authorization system, like e.g. a set of roles and permissions that are understood among all protected resources. Security realms can also be nested, e.g. to support groups of Web services with a dedicated role system (compare *Realm C* in Fig. 4).

f. *Identity providers* serve as the places where the known users of the realm's applications as well as the applications and services themselves can be registered and have their accounts. As such, they can authenticate the members of the realm through login forms or Web service interfaces. The tokens they issue form the foundation for the security token service's authorization decisions.

g. The potential accesses on services and applications are marked with *invocation* links. Optional labels indicate the designated protocols, like e.g. HTTP, HTTPS, SOAP via HTTP, SOAP via SMTP etc. When two realm entities (i.e. services or applications) are linked, the intended meaning is that the service at the pointed end is called by the other entity. A link between an entity and a surrounding realm states that the entity is principally accessible from outside the realm (i.e. it is public). In the example in Fig. 4, service *WS2* is only available to the intranet (from applications residing in *Realm B*).

h. Separate realms that are to form a federation can establish *trust relationships*. Semantically, this means that the STS of the trusting realm accepts the tokens originating from the trusted realm. Hence, the identities of the foreign requestors can be mapped to tokens that are locally valid, enabling realm-crossing accesses in a controlled and manageable way.

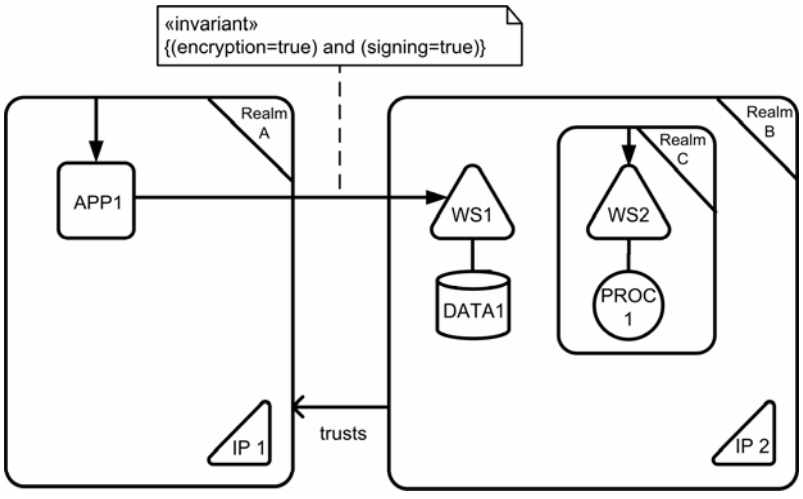

Fig. 4: Example showing the combined use of the introduced model elements

The introduction of the graphical symbolism can be understood as an approach to establish a good overview of the modeled scenarios. In our experience, real systems possess far more characteristics than can sensibly be visualized all at once. For instance, the protocols cannot always be referred to by a simple name like *HTTP*. Instead, even when relying on standards, there exist a huge number of possibilities concerning e.g. cryptographic operations or ways of requesting and passing on security tokens. We therefore suggest the use of the Object Constraint Language (OCL) [16] in correspondence to the way OCL supplements UML diagrams. This allows for refining the model's semantic with annotations like the one in Fig. 4. The added restrictions can be related to the attributes of a meta-class of the modeling element. In this case, the expression states that the messages exchanged during service invocation are signed and encrypted. Hence, the protocol labels can also be seen as abbreviations for a more precise, but at the same time more complex notation.

5 WAM Applied for Integrated Information Management

To demonstrate the practical relevance of the WebComposition Architecture Model, we outline its application in a real-world scenario in the context of an integration project conducted at the University of Karlsruhe [2]. The principal goal of the project is to integrate existing applications from different domains and business units based on a service-oriented architecture (SOA) at the University and to support its seamless evolution. By exposing the functionality of a diversity of distributed legacy systems as atomic Web services that can be composed to higher-level service domains, the resulting architecture brings together systems under the control of different university divisions based on heterogeneous platforms.

The following example involves the applications of two sub organizations: the central administration department and the Department of Economics and Business Engineering (DEBE). Both departments maintain stored data about students independently from one another. In the case of the administration, the functionality of interest is provided by HISLSF, an information system widely used at German universities. The DEBE, on the other hand, employs a Web-based system implemented with the scripting language PHP. A demand for integration arose from the process of controlling admission to computing facilities, which requires student data from both sub organizations. The scenario involves several aspects covered by WAM: the exchange of data with different protocols, non-public Web services and trust relationships between distinct realms of control. The realization of the scenario has been achieved with the help of a support system we implemented to provide for the necessary token-issuing services and communication infrastructure in correspondence to WAM [6]. Fig. 5 depicts an overview of the implemented scenario using the proposed notation.

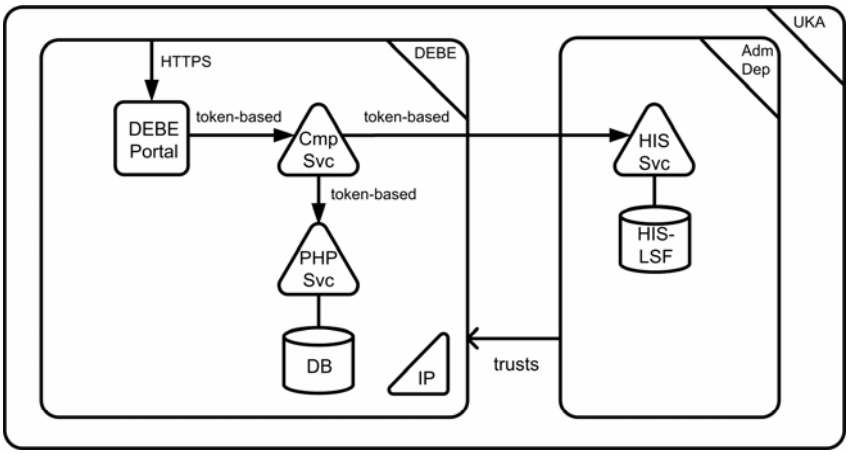

Fig. 5: A real-world integration scenario modeled with WAM

Representing the servers and applications that form autonomous segments of the university network (*UKA*) to be integrated, the model contains the two realms: *DEBE* and *Administration Department*. According to the WAM definition given in section 4, both realms are equipped with a security token service for access control. As the scenario involves just users of one organizational unit, there is only one identity provider (*IP*) required to authenticate these users. Similarly, the one-way trust relationship reflects the fact that inter-realm service accesses occur only in one direction. Applied to the example, this means that the administration department trusts the Department of Economics and Business Engineering to authenticate their own users and allows some of these to access (HISLSF) services.

In a typical use case, a computer facility administrator at the DEBE, who has been authenticated by the identity provider, accesses the *DEBE Portal* to request some student status information from a composite Web service (*Comp Svc*). The service itself calls atomic Web services of both realms (*HIS Svc* and *PHP Svc*) that provide the data from the underlying legacy systems: a database (*DB*) and the HISLSF-system (*HISLSF*). As can be seen in Fig. 5, the Web services are not publicly available: they can only be accessed from within the realm and with the help of valid security tokens issued by the realm's security token service. Technical details like token emission, transport and translation, as covered by the WS-Federation specification, are hidden by the model to offer a perspective focused on the actual trust structure and Web service dependencies.

In Fig. 6, the visible parts of the outlined use case as well as their connection to the model elements can be seen. The login screen is actually the front end of a separate Web application, the identity provider. The credentials entered by the accessing users are checked against the central LDAP directory of the department. Integrated into the portal, a page component provides the student data functionality by serving as the user interface for the composite Web service. Search queries are translated into SOAP calls that, among other parameters, contain the user's security token.

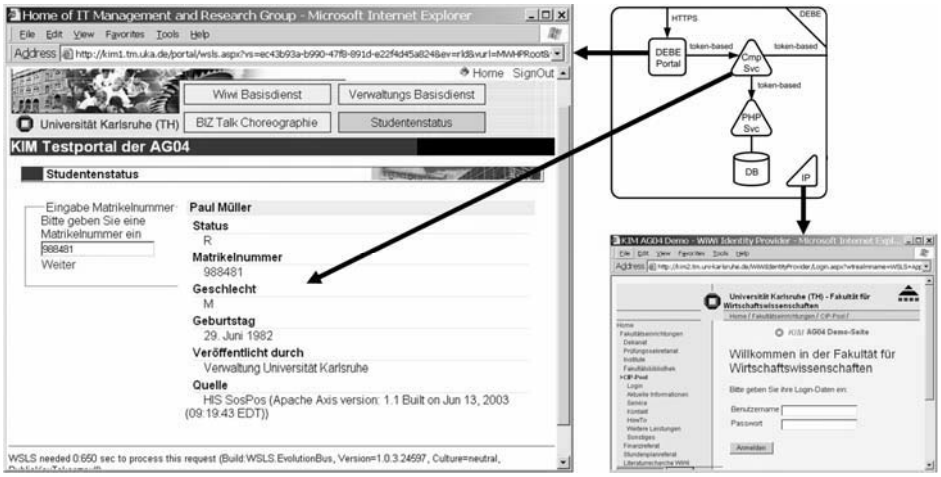

Fig. 6: Interrelation between the WAM model and the implemented system

6 Conclusion and Future Work

Inter-organizational Web-based applications evolve over time and as such need to be modeled in order to keep track of their current condition and to plan future changes in a systematic manner. Comparable attempts to model distributed systems exist, but do not specifically target cases where the modeled applications are controlled by several partners that take part in a federation. We discussed the problem from a evolution-oriented Web Engineering perspective and identified three types of evolution and their impact on the resulting development process. From these insights we concluded that federated evolution particularly requires modeling support. To complement the state of the art, we proposed the WebComposition Architecture Model (WAM) as a model with a UML-like notation that can be employed to describe landscapes of Web applications and Web services across multiple zones of control. As issues like e.g. the diversity of communication protocols can result in a complexity that exceeds the usefulness of graphical notations, we introduced OCL expressions as means to further specify system details. WAM has been successfully put to use in an integration project at the University of Karlsruhe. The federated, trust-oriented view turned out to be very appropriate, as requirements demanded that the involved sub-organizations would be able to stay in full control of the accesses on their applications.

As a next step, we are now working towards runtime support to achieve a tighter interaction between the model and the applications. By making the federation modeling information itself available via a Web service interface and linking it to the configuration of the participating services and applications, we hope to lay the foundation for tools that can be used to further facilitate federated evolution. This includes e.g. the process of joining the federation with new services or altering the structure of trust relationships between the security realms. Moreover, we are now formulating WAM in higher detail, possibly adding a meta model.

References

1. Liberty Alliance Specifications - Web site (2004), Liberty Alliance Group: http://www.projectliberty.org/resources/specifications.php (18.10.2004)
2. KIM Project Homepage, 2005, University of Karlsruhe: http://www.kim.uni-karlsruhe.de/ (24.04.2005)
3. Bajaj, S., et al.: Web Services Federation Language (WS-Federation) - IBM Developer Network (2003): http://www-106.ibm.com/developerworks/webservices/library/ws-fed/ (14.10.2004)
4. Deshpande, Y., et al.: Web Engineering. Journal of Web Engineering, 2002. 1(1): p. 3-17
5. Gaedke, M.: WebComposition: Ein Unterstützungssystem für das Web Engineering. GI Softwaretechnik-Trends, 1998. 18(3): p. 20-25
6. Gaedke, M., Meinecke, J., and Nussbaumer, M.: A Modeling Approach to Federated Identity and Access Management. in Poster Proceedings of the Fourteenth International World Wide Web Conference. 2005. Chiba, Japan: Association for Computing Machinery (ACM)
7. Gaedke, M. and Turowski, K.: Specification of Components Based on the WebComposition Component Model, in Data Warehousing and Web Engineering, S. Becker, Editor. 2002, IRM Press: Hershey, PA, USA. p. 275-284
8. Gootzit, D. and Phifer, G.: Gen-4 Portal Functionality: From Unification to Federation. 2003: Stamford, CT
9. Linthicum, D.S.: Next Generation Application Integration. Information Technology Series. 2004, Boston, MA: Addison-Wesley
10. Maler, E., Mishra, P., and Philpott, R.: Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1 - OASIS Standard (2003): http://www.oasis-open.org/specs/ (18.10.2004)
11. Mcclure, C.L.: Software reuse techniques : adding reuse to the system development process. 1997, Upper Saddle River, N.J.: Prentice Hall. xxiv, 350
12. Microsoft: Dynamic System Initiative Roadmap - Web Site (2003): http://www.microsoft.com/dsi (14.10.2004)
13. Oasis: Data Center Markup Language Framework Specification - OASIS Web site (2004): http://www.dcml.org/technical_info/ (03.11.2004)
14. OMG: UML Profile for enterprise distributed Object Computing (EDOC) specification - Web site (2004), Object Management Group: http://www.omg.org/technology/documents/formal/edoc.htm (20.06.2005)
15. SEI: Domain Engineering - Web-Page (1999), Software Engineering Institute, Carnegie Mellon University: http://www.sei.cmu.edu/domain-engineering/domain_engineering.html (01.10.1999)
16. Warmer, J. and Kleppe, A.: The Object Constraint Language, precise modeling with UML. 1999, Reading, Mass.: Addison-Wesley Pub Co
17. Witty, R.J. and Wagner, R.: The Growing Need for Identity and Access Management. 2003: Stamford, CT