RESOURCE MANAGEMENT IN WEB-BASED BUSINESS APPLICATIONS

MARTIN GAEDKE* AND KLAUS TUROWSKI+

*Telecooperation Office (TecO), University of Karlsruhe, Vincenz-Prießnitz Str. 1, 76131 Karlsruhe, Germany, phone: +49 (721) 69 02-79, FAX: -16, email: gaedke@teco.edu

+Otto-von-Guericke-University Magdeburg, Business Information Systems, P. O. Box 4120, 39016 Magdeburg, Germany, phone: +49 (391) 67 1-83 86, FAX: -12 16, email: turowski@iti.cs.uni-magdeburg.de

Abstract
The World Wide Web (WWW, Web) has become a new application platform offering most services needed for a distributed operating system. With this, an opportunity is created that allows fusing originally separated business application systems to support new, value-added business applications accessible via the Web. In this paper, we present an approach to facilitate collaboration of business application systems. In particular, we point out how new Web-based business applications are created by fusing services of existing application systems that may reside in heterogeneous environments. Furthermore, we introduce a means to support inter-application communication on a semantic level, thus providing a global information infrastructure for Web-based business applications that supports resource management.

Keywords: Information infrastructure; Web-based business applications; WCML; XML; EDI

1. INTRODUCTION

In the ensuing decade, the use of the Web has moved far beyond of its originally anticipated scope. Actually, we face its gradual change from a distributed system for knowledge-interchange towards a new application platform offering most services needed for a distributed operating system. This yields to a dramatic and rapid growth of the Web, more recently triggered by organizations offering Web-based products and services and thus asking for e-commerce applications. Especially business to consumer e-commerce expands with benefit from the distribution of Web browsers for universal, cross-platform access and offers opportunities for new products, e.g. the integration of a rapidly increasing number of Web-based services provided by different business units [5] to a single point of access. This often requires that services of different business application systems, e.g. systems for order management or
inventory management, are combined to a new product, which hides the respective services and application systems from the customer. The Web was developed to provide information on a simple distributed document-centric system and has evolved into a complex (application) system, that benefits from centralized maintenance and evolution as well as from effortless deployment of services at low cost. As the Web has become a runtime environment for small-scale to large-scale business applications, a more structured and disciplined approach for design, implementation, and maintenance respectively evolution of such Web applications is required. Specifically, as the life cycle of Web applications is highly influenced by arising technologies, user-behavior, and state-of-the-art trends in e-commerce.

Nevertheless, today's development of Web applications suffers from the underlying simple and coarse-grained implementation model leading to an ad-hoc application development practice [2]. This is mainly forced by the gap between fine-grained design models and the coarse-grained Web implementation model, which at least complicates the application of software engineering practice. Since the Web does not relate well to state-of-the-art software development models, it is rather difficult to define architectures and frameworks that ease the construction of Web-based applications or the fusion of several heterogeneous applications to a Web-based business application system.

Even if the gap between the models is bridged manually, design decisions will get lost during implementation. The reuse of Web-based business applications and resources yields therefore to a bigger complexity. Building Web-based application systems has to be supported on an architectural level that provides a properly granularity and preserves design decisions independent from a given implementation. To do so, we use the WebComposition Markup Language (WCML) based on the object-oriented WebComposition model that allows the reuse of code and design, and supports automatic code generation. WCML is an application of the XML (eXtensible Markup Language) and allows the definition of WCML components as fine-grained entities for modelling design and code artefacts.

As services of different business application systems are combined to new Web-based products, communication between (legacy) business application systems, that collaborate to make a given product available, has to be ensured on a basic level. For this reason, we propose the use of a Tuplespace as a layer of the architecture to provide inter-application systems interchange of business data by means of an EDI (Electronic Data Interchange) approach that is based on XML [1].

The next section introduces basic terms, shows how they are presented and provided in the Web, and discusses the proposed architecture for an efficient Web-based resource management in more detail. It contains the technical basement for modeling services in the Web, and a description of our approach to provide Web-based business applications. Furthermore, we explain how communication of different underlying business application systems is ensured. Finally, we will give conclusions and discuss further work.

2. BRIDGING THE GAP BETWEEN THE FINE-GRAINED DESIGN MODEL AND THE COARSE-GRAINED WEB IMPLEMENTATION MODEL

The document-centric Web implementation model prevents the possibility to model design elements, like user interface objects in interactive applications, dialogues part of workflows, or structural elements to keep care of the application's corporate look and feel. Fine-grained
model entities will get lost after mapping to coarse-grained entities of the implementation model, as they have to be glued together in one code base or even be cloned into several documents. Thus, changes in design and layout, or reuse of the entities for cost reduction and quality improvement makes maintenance respectively evolution of a Web application a difficult task to perform. Design methods and systems with support of mapping higher-level concepts and fine-grained entities to the Web, like OOHDM [10] or RMM [8] address these problems in different ways. The disciplined development of large-scale Web applications with heterogeneous legacy systems in the back-end and a possible partial distribution for maintenance and reuse of higher-level concepts is less looked at, even though this scenario is important in e-commerce applications.

The fine-grained entities of the design model often do not match very well with the granularity of the Web implementation model. Even if Web server technology is enriched, like the introduction of servlets or server-side scripting technologies, the platform-independent deployment entities (the Web documents) burden a disciplined reuse of existing code. In contrast the generating methods, like server-side script code, prevents to reuse by depending on a specific platform infrastructure.

Figure 1: Architecture and integration WCML components
The WebComposition approach introduces an object-oriented model that uses components as a uniform concept for modeling *code abstractions* of arbitrary target languages. As the model is object-oriented, its entities may be composed by reusing existing components. A component representing a Web page may be composed by referencing components that are responsible for the code delivery of a header, body, and footer. The WebComposition model is based on a *prototype-instance paradigm* [13], in contrast to a class-based object-oriented model that requires a class definition to instantiate objects. In the WebComposition model an object is an instance of a component, and a component can serve as prototype for other components. Another possibility to share the code of a component, is to allow multiple references on the same component. Sharing is fundamental for reuse and for maintainability.

The WebComposition approach aims at the target that reuse of design and code is successfully practiced in software development. There are many examples that show how reuse is responsible for quality improvement and cost reduction[4]. WebComposition is not intended and does not compete with other component technologies like Java-Beans or DCOM/COM, in contrast, any target language code fragment can be modeled; allowing the support of federating component models for heterogeneous target languages or systems. The WCML document type definition describes the markup notation for the components. Components described in WCML reside in a WCML document, which we refer to as a *virtual component store*, in conformity to the WebComposition system described in [7]. Like the delivery of HTML documents, the delivery of WCML components described using the XML standard is therefore easily possible by delivery of WCML documents. This implementation technology serves for the service-oriented Web application development that we describe in more detail in the following sections.

3. REUSE AND EVOLUTION IN WEB-BASED BUSINESS APPLICATION DEVELOPMENT

Web-based business applications typically serve customers with the ability to order products or retrieve information. In general, we can abstract from the different possibilities a customer has, by saying a Web-based business application provides a service or services to its users. Each product, news item etc. can be modeled as a service that is part of a company's business process. A service captures information about items that a company wants to provide, how to present the information, and what to do with received data. A user can finally access a service through defined Web pages and navigational concepts. Further, a service must be able to handle further processing steps in the back-end.

Using WCML, we propose the definition of a service as a set of components:

- *Service content*: A WCML component defined by a set of properties responsible for capturing the different object attributes.
- *Service layout and navigation*: A WCML component defining the layout respectively how the service-content should be displayed. This component may be composed of several other components, e.g. if the service content should be displayed with several Web-pages. The latter case would also allow to define complex navigational contexts.
- *Service processing*: A set of instruction or calls to initiate the processing by a legacy system. We propose a more general concept by definition of a WCML component capable of
generating an independent interchange format. This allows that a service can be (partially) processed by different systems in the back-end.

Service-components may be used as prototypes for additionally services and define a way for a seamless evolution of the overall application. E. g. a company selling a brand-new mobile phone may reuse the existing service of an older mobile phone by using the old service-component as a prototype. Merely the service-content must be overloaded with the data of the new phone. Further, the service can be provided in several layouts using different kinds of navigation and adapting to the needs of different user classes.

A service-oriented Web application development profits from this abstraction. As pages, navigation, and processing can be grouped easily by accessing the responsible service component, different services may be set up together, and therefore describe an application configuration. Different classes of users can be served with the services they really need without developing one service twice. The abstraction supports and enhances the use of corporate look and feel layout-components, or the reuse of special design artifacts and navigation structures.

The change of design of parts or of the complete Web application by redesigning components is therefore possible. New services based on a given service may be created through the implementation of a service-factory design pattern, as described by [6]. The use of this pattern makes a system independent of how its objects are created, composed, and represented. It is evident that service-factories are an important element for the evolution of a Web application by means of cost reduction and quality improvement.

Refering to the above example, we combine two services offered by different business application systems to produce a new product “registered mobile phone”. This product consist of the business services “sale and registration of a phone card” and “sale of a mobile phone”. Looking on the connected business process reveals that at least two companies, or two organizational units of the same company, are involved in creating this new product: One that produces and customizes the mobile phone and one that provides the registration. Furthermore, there is an additional information flow between the two companies, which is hidden from the customer. This information flow is related to the management of physical distribution of the sold goods, as phone card and mobile phone should be delivered simultaneously, and to the terms of payment, as the mobile phone is often (partly) paid by the phone company, but not by the customer. To automate this information interchange efficiently, communication on a semantic level between different (legacy) business application systems, that offer services to make a certain product available, has to be ensured.

Instead of providing specific connection software for any possible combination of business application systems that may have to interact with each other, we propose the use of a tuplespace, which provides a common (business) interchange format (cp. figure 1) and that is based on WCML components.

For the tuplespace approach to work, we need different parts:

- An ontology (a common (and standardized) term set that covers most business transactions and that is accepted by all participants),
- a special communication service offered by each business application system, that translates the application systems output to the common term set and vice versa, and
- the tuplespace itself, that preserves and propagates messages between the business application systems.

The question of the right ontology for business transactions leads directly to EDI. EDI presents a way to establish and improve communication between different business partners.
Implementing EDI leads to organizational informational surplus values, e.g. an improved organizational as well as operational structure, or time and cost savings.

```
<Message>
  <Type>Request for quote</Type>
  <Request-Date>22.1.98</Request-Date>
  ...
  <Product>
    <EAN>230239844531</EAN>
    <Description>Mobile phone</Description>
    <Color>blue</Color>
  </Product>
  ...
</Message>
```

Figure 2: Part of a message generated to request whether a certain mobile phone is available

The most important standard for EDI was established by the United Nations, known as UN/EDIFACT (Electronic Data Interchange for Administration, Commerce and Transport) [12]. It standardizes electronic exchange of structured information, e.g. orders or invoices, thus permitting a direct communication between different business application systems.

Taking the approach given in [11, pp. 6-9], we propose to use UN/EDIFACT segment names as a standardized term set, and to encapsulate business data, which have to be exchanged in XML tags that are named according to this term set. This does not implicate any restrictions, since all efforts mentioned above regarding standardization of uniform business scenarios and semantic rules provide backward compatibility to UN/EDIFACT.

By encapsulating business data in XML tags with a given standardized meaning, messages from other business application systems become understandable. Furthermore, only important parts of a message may be processed.

Figure 2 shows a typical example for an XML-based EDI message. With this example it is easy to understand how data encapsulated in XML tags get a meaning, and how only relevant data may be extracted by the receiver. If the receiver would need the article number to process the request for quote, it would be enough to access the data, which is encapsulated in the EAN tag. Another receiver might need the description and the color as well to process the request. In this case, resulting in a search for the DESCRIPTION and the COLOR tag. The underlying so-called matching process to find the relevant tags is done by using XML parsers, even with concern to nesting, as a COLOR tag within a PRODUCT tag might have a different meaning than a COLOR tag at a different position. XML parsers are (freely) provided from different sources, e.g. XML4J from IBM, XMLParser from Sun, or the MSXML Parser from Microsoft.

However, the tag names in figure 2 are not standardized, as they were arbitrarily chosen to be understood by human readers. In order to support an automated inter-machine communication we use UN/EDIFACT segment names as tag names as depicted in figure 3. The upper part of figure 3 shows a part of the UN/EDIFACT definition of a date. The segment names are printed bold. The lower part of the figure shows the corresponding XML message, which substitutes the REQUEST-DATE tag from figure 2. Its meaning is: The request for quote was issued at December 18, 1998.
DTM DATE/TIME/PERIOD
To specify date, time, or period.

2005 Date/time/period qualifier, M, an..3

137 Document/message date/time
Date/time when a document/message is issued.

2380 Date/time/period, C, an..35

2379 Date/.. format qualifier, C, an..3

102 CCYYMND
Calendar date: C = Century;
Y = Year; M = Month; D = Day.

<DTM DTM2005="137">
<DTM2380>19981218</DTM2380>
<DTM2379>102</DTM2379>
</DTM>

Figure 3: UN/EDIFACT segment names and a corresponding message

With the common term set defined, a special communication service is needed, which is offered by each business application system whose services are part of a certain product. The communication service translates the application systems output to the common term set and receives messages via the tuplespace that preserves messages regarding to the common term set. Thus, the communication service consists of two parts: an input service, and an output service. The output service translates the messages of a certain business application system to XML-based UN/EDIFACT, and has to be implemented for each application system individually. To implement the input service, that translates XML-based UN/EDIFACT messages to a format that can be understood by the respective application system, the XML parsers described above can be reused. Again, input and output service are "accessed" via WCML components that are capturing the code needed for initiating communication (figure 4), and may be implemented as, e. g. Java-based software agents as proposed in [3].

For implementation, we use a tuplespace approach [9], which serves as a distributed and shared data space. The data items that the tuplespace is capable to handle are tuples. Marking a tuple semantically can be done by using XML, and thus enhancing the tuplespace to deal with XML tuples, which are simply XML documents, especially XML/EDI data. The XML tuplespace provides the functionality to store structured data and to enable distributed processes, e. g. the mentioned software agents, to access and modify XML tuples.

Data conform with a DTD (Document Type Definition) is said to be valid XML. In this case, an XML parser could check incoming data against the rules defined in the DTD and check if the data was structured correctly. Data is known as well-formed XML if it is structured as defined by XML and sent without a DTD. The XML tuplespace supports the handling of both, valid and well-formed XML, giving the processes connected to the XML tuplespace the freedom to exchange and transfer any information.
The functions provided by the XML tuplespace are `in`, `out`, and `read`, and are provided by an XML tuplespace that is addressed by the given XML tuplespace URL, as described in the following:

- **`out(XML tuplespace URL, DTD, XML tuple)`**
  The function `out()` puts an XML document, specifically an XML/EDI document into the XML tuplespace without overwriting existing XML tuples. If the XML tuple is sent along with a DTD, the XML tuple is said to be a valid XML tuple, otherwise a well-formed XML tuple. Valid XML tuples ensure that the data is conform to a given grammar, provided by the DTD.

- **`in(XML tuplespace URL, DTD, XML tuple)`**
  The function `in()` returns an XML tuple from the XML tuplespace if a XML tuple matching to the DTD and the given XML tuple can be found. The returned XML tuple will still remain in the XML tuplespace.

- **`read(XML tuplespace URL, DTD, XML tuple)`**
  The function in `read()` returns a XML tuple from the XML tuplespace if a XML tuple matching to the DTD and the given XML tuple can be found, and deletes the XML tuple in the XML tuplespace.

To access an existing XML tuple the XML tuplespace provides three kinds of searches:

- **Actual**
  Actual XML tuples are all XML tuples that do not contain a placeholder tag, e. g. `<*>`. An actual XML Tuple is:
• Formal
Formal XML tuples are all XML tuples that contain a placeholder tag in a data section. The placeholder serves for supporting pattern matching algorithms, e.g. implemented in a database. Formal XML tuples are not valid XML. The following XML tuple may be used to search for the above actual XML tuple.

    <PRODUCT>
    <EAN>230239844531</EAN>
    <DESCRIPTION><*></DESCRIPTION>
    </PRODUCT>

• Semantic
Semantically, XML tuples are all XML tuples that contain a placeholder tag for semantic searches. Semantic XML tuples are not valid XML. E.g. the following XML tuple along with a given DTD could be used to find all XML tuples of the type defined by the DTD:

    <*)&>

As the client process accesses a tuplespace through HTTP (HyperText Transfer Protocol), standard authentication and cryptography mechanisms may be used to secure the business data interchange.

4. CONCLUSIONS AND FUTURE WORK

We have shown how services offered by different business application systems in heterogeneous environments may be combined to new (Web-based) products using WCML components, XML-based EDI, and a tuplespace. With this, an architecture is provided, that allows the use and definition of services, that abstracts from implementation details specific to respective heterogeneous environments. Our approach has been implemented in part as a prototype, and its use has been shown in practice [5]. Through the support of multiple target languages, different components (with same functionality) for different legacy systems may be embedded in one product, and reuse is supported on design level as well as on implementation level. As our approach is based on common standards, e.g. XML, Java, or UN/EDIFACT, positive effects as cost reduction and shorter times to market due to reuse as well as a better integration of legacy systems are increased, and diffusion is promoted, finally resulting in an improved resource management.

REFERENCES


119


[6] E. Gamma; R. Helm; R. Johnson; J. Vlissides. "Design Patterns: Elements of Reusable Object-Oriented Software". Reading, (1994)


