The Impact of Domain-Specific Languages for Assembling Web Applications

Martin Nussbaumer, Patrick Freudenstein, and Martin Gaedke

Abstract - **Developing distributed Web-based solutions is not only a difficult task from the technological perspective. Communication problems concerning hypermedia and Web aspects between the developers and the business are an additional major roadblock to a project's success. In order to clarify these communication issues, we report from our experiences gained in a large-scale Enterprise Application Integration project. We address this problem area by applying Domain-Specific Languages and a supporting technical framework. Our overall vision is to enable domain experts to directly contribute to the development effort by autonomously specifying parts of the solution. A set of DSLs and notations derived from our experiences covering central concerns of Web applications is presented: navigation and structuring of application domains, data interaction using Web services, and Web-based process guidance. Web applications can thus be built in an evolutionary manner by composing building blocks whose behavior is configured with DSL programs. Our approach emphasizes conceiving communication with stakeholders and Web application development in a holistic way.**

Index Terms - **Conceptual Modeling, Domain-Specific Languages, Enterprise Application Integration, Reuse, XML and Web Services**

I. Introduction

In software development projects, the specification of requirements and aspects of the envisioned solution is a time-consuming task suffering from communication problems between the developers and the business [1]. Evaluations on reasons of project failures like the Standish Group's CHAOS Reports [2] show that factors like user involvement and clear business objectives, relying on efficient communication between the different stakeholder groups, are crucial for a project's success.

Beyond written natural language, further and much more sophisticated approaches towards a systematic and formal description of aspects of a solution have emerged. Especially for the development of Web applications, a variety of research activities in the fields of requirements engineering and management, conceptual modeling as well as domain engineering have evolved [3-6].

Manuscript received April 10, 2006.

Martin Nussbaumer, Patrick Freudenstein, and Martin Gaedke are with the University of Karlsruhe, Institute of Telematics, IT-Management and Web Engineering Research Group, Engesser Str. 4, D-76128 Karlsruhe, Germany, e-mail: {nussbaumer, freudenstein, gaedke}@tm.uni-karlsruhe.de, phone: +49(721) 608 – 8073, fax: +49(721) 388 097.

Beyond that, *Domain-Specific Languages (DSLs)* recently gained increasing attention. Deursen, Klint and Visser define them in [7] as programming languages or executable specification languages that offer, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain. Due to their limited scope and their level of abstraction tailored to the problem domain, DSLs are easy to understand and use, especially for domain experts and non-programmers. By the use of various graphical notations and accompanying editors, each of them being as intuitive as possible for a particular stakeholder group, the usability of a DSL can be further improved [1]. Like programs developed with general purpose languages, e.g. Java or C#, DSL programs can also be transformed into executable code using a dedicated DSL compiler. As a result, domain experts themselves can contribute directly to the development effort by validating, modifying and even creating parts of the solution on the basis of DSLs. The advantages of using DSLs do not only affect domain experts and non-programmers; they also comprise factors like increased productivity, reliability and maintainability [8] as well as efficient reuse [9].

In section 2, we report from our experiences gained in a large-scale university-wide Enterprise Application Integration (EAI) project. We address the problems found in the field of communication and collaboration with stakeholders and outline our solution. Section 3 introduces our evolutionary DSL framework approach and section 4 gives an overview of the underlying technical platform. Afterwards, in section 5, we present a selection of core DSLs that have been used efficiently in several scenarios. Finally, the exemplary evaluation of our approach is conducted and the conclusions and future work are given.

II. An Experience Report

We have been collaborating in the university-wide EAI project "Karlsruhe's Integrated Information Management (KIM)" [10] for several years now and have experienced various problem areas.

A. *Initial Situation*

We found technical problems due to the heterogeneity of the systems to be integrated. As a result of the decentralized organizational structure of a university, a huge diversity of distributed, autonomous IT systems has evolved over the decades. From the technical perspective, carrying out system integration in such a heterogeneous environment is a very difficult task. Another issue that comes along with the heterogeneity and decentralization is the lack of reuse capabilities due to technical limitations in the past. When developing applications in the KIM project, we often find existing software artifacts that would perfectly fit in a solution but cannot be reused efficiently.

On the other hand side, social problems make up the second major problem area. This includes known issues like fears of organizational change as well as problems in the fields of communication between the different project participants and stakeholder groups. Compared to traditional software development projects, the latter is being aggravated by the great diversity of involved stakeholders from completely different faculties and departments with entirely different skills and knowledge. Consequently, each group uses its own "language" when talking about aspects of the solution to be built. This becomes obvious especially in the phases related to requirements management and conceptual design when understanding the various languages of the particular stakeholders is decisive and misunderstandings must be avoided as far as possible. Agreeing on and learning a common language as a potential solution to this, has turned out to be not feasible because of the stakeholders' usually very limited availability. For example, when talking about business processes with stakeholders from all over the university, some of them prefer Petri nets as a means of communication as they play a major role in their research context. Others, for example, favor the Business Process Modeling Notation (BPMN) for the same reason. And people with a background in humanities and social sciences often like a notation in natural language better.

Regarding the technical problem domain, many promising approaches and research activities already exist [11-13], just to mention a few. Thus, our main challenge remained to find a way enabling a common understanding between IT staff and stakeholders throughout the development process without neglecting the technical perspective.

B. *Towards Communication*

In our project, stakeholders belonged to diverse departments from all over the university, had completely different skills and academic backgrounds, were predominantly non-programmers, and, in the most cases, were rarely available. Furthermore, most of them had no experiences in employing particular languages for the specification of parts of a solution, often they had not even heard yet of the modeling languages the developer team would like to use.

Over the last years, a lot of languages and modeling approaches for the specification of the various dimensions of a distributed Web-based solution have been established. Most of them attempt to cover their problem domain as exhaustive as possible and therefore include concepts and notations for almost every aspect of the problem domain. This often leads to very expressive and powerful modeling languages, being a good means of conceptual and logical design within the developer team. However, regarding the communication and collaboration with stakeholders in our project, they were not appropriate. Our stakeholders are often non-programmers and in the most cases are hardly available for interview sessions. Hence, it would cost too much time and effort for them to learn the various languages and notations for the diverse problem domains.

With respect to the characteristics and challenges in EAI projects described before, an adequate language should be easy to learn, understand and use, both for developers and stakeholders. Thus, it should be as simple as possible and as extensive as necessary. Based on our experiences, simplicity is often the key factor to usability and effectiveness. We often succeeded with languages that covered only about 80% of a problem domain's complexity but in return enabled all stakeholders to understand and use them. Furthermore, including the high-level abstractions and concepts of a problem domain proved to make a language more productive. Thus, learning, applying and understanding such languages became even easier for domain experts. An additional improvement to the usability and intuitiveness of a language could often be achieved by providing several (graphical) notations for one language, each of them tailored to a particular stakeholder group. Consequently, our overall vision was to enable domain experts to directly contribute to the development effort by autonomously specifying parts of the solution.

C. *DSL – Shaping Communication*

We discovered a multitude of small, simple and highly focused languages for solving small and clearly identifiable aspects of a distributed Web-based solution being a more viable and efficient alternative. Taking all the characteristics of DSLs into account and comparing them to the challenges in the field of communications between the developers and the business we faced in the KIM project, they turned out to be an ideal building block for our solution. In response to the diversity of stakeholder groups in our project, by using DSLs, we are able to collaboratively specify solutions for particular problem domains employing abstractions and graphical notations as suitable as possible for the various groups.

To unfold the full power of DSL-based development, so-called Language-Oriented Programming (LOP) [14, 15], a technical framework is required. Developing distributed Web-based solutions by adopting multiple DSLs to specify their various aspects demands a technical platform for transforming DSL programs into executable code. As a foundation of our technical solution, we introduced the central concept of Solution Building Blocks (SBBs). SBBs are software components whose behavior can be configured through DSL programs. Web applications can thus be built in an evolutionary manner by composing and managing SBBs. We use the WebComposition Service Linking System (WSLS) [16] as platform for the SBBs, facilitating their systematic composition and configuration.

III. A DSL-BASED WEB ENGINEERING APPROACH

In our approach, we place emphasis on simplicity. This results in a multitude of DSLs for the various aspects of Web applications, each of them being suitable for a small, clear problem domain and providing abstractions and notations tailored to various stakeholder groups with different skills and preferences. In the following subsections, we first describe the different layers and their associated components making up a DSL. Following that, we present the cornerstones of our approach to the systematic evolution of the emerging variety of DSLs.

A. DSL – Reading the Fine Print

Fig. 1 shows the elements of our approach towards DSL-based Web Engineering which is based on the principles of evolution and reuse. We differentiate between two phases in the course of a continuous evolution: *Development for Reuse* comprises the design and development of a DSL and *Development with Reuse* covers the usage of a DSL for the specification and development of a part of a distributed Web-based solution.

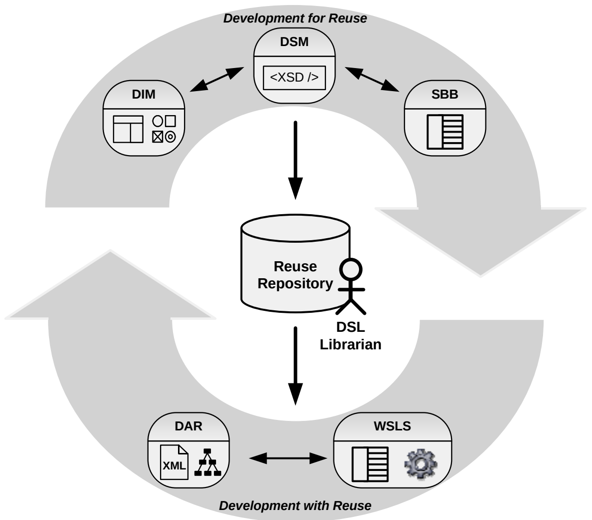


Fig. 1: Overview of our evolutionary and reuse-oriented approach towards DSL-based Web Engineering

In our approach, a DSL consists of three components. The *Domain-Specific Model (DSM)*, usually an XML Schema Document, represents the formal schema for all solutions that can be described with the DSL. Thus, the DSM has to be designed in accordance with the problem domain the DSL is intended for. Based on the DSM, a *Domain Interaction Model (DIM)* comprises a dedicated (graphical) notation being as intuitive as possible for a particular stakeholder group. The DIM is tightly coupled to the DSM; however, it needs not to cover all of its aspects. By using a DIM, stakeholders can employ the DSL, i.e. understand, validate and even create DSL programs, without being confronted with complicated source code. Instead, the DIM should provide concepts and notations derived from the problem domain and thereby should be easy to understand and use. In order to meet different requirements and characteristics of various stakeholder groups, a dedicated DIM for each group could be included in a DSL. A further enhancement to the usability and effectiveness of a DSL can be achieved by accompanying editing tools based on the notations specified in the DIMs.

Besides the design of a DSM and one or more DIMs, the development of a *Solution Building Block (SBB)*, being capable of executing the DSL programs, completes the "Development for Reuse" phase. An SBB can be seen as a software component whose behavior can be configured through a DSL program, usually in terms of an XML document. We use the *WebComposition Service Linking System (WSLS)* [16] as technical framework for the SBBs. WSLS aims at facilitating the systematic evolution of Web applications by reusing software artifacts and emphasizing the "configuration instead of programming" paradigm. In our approach, the WSLS framework allows for the systematic composition and configuration of SBBs with DSL programs.

During the "Development with Reuse" phase, the *Domain Abstract Representation (DAR)* is developed. The DAR represents the specification of a concrete solution within the DSL's problem domain. In other words, the DAR is a DSL program. Consequently, it is based on the DSM and modified by using one or more DIMs. As the DSM is usually specified as an XML Schema, the DAR is serialized and stored in an XML document based on the DSM. However, in contrast to today's integrated development environments (IDE), the editing process using DIM notations is not performed on this serialized form. Modifications are rather carried out directly on the abstract model itself. Thus, DSL programs can be edited in a more powerful way than it would be possible if interacting with the DAR's serialized form. After having developed a DAR, its XML representation is passed to the DSL's associated SBB, i.e. a component of the WSLS framework. The SBB in turn adapts its behavior according to the DAR and thereby executes it. Web application development can thus be performed in an evolutionary manner by composing SBBs and configuring them with DARs.

B. Systematic Evolution

Domain-Specific Languages are subject to continuous evolution. Their lifecycle starts with the identification of the need for a DSL for a particular problem domain, often based on experiences gained from the collaboration with stakeholders. This is followed by the specification of a new DSL, consisting of a Domain-Specific Model, one or more Domain Interaction Models and first steps towards a dedicated tool support for the DIM(s). Thereafter, the new DSL is employed in collaboration with different kinds of stakeholders. Thereby, new experiences are gained permanently, resulting in requests for change. This in turn results in the adaptation of the DSL's models and so on. Hence, the set of available DSLs for building Web applications underlies a continuous evolution through variation and selection: new DSLs are added, existing DSLs are improved, and DSLs that have turned out to be dispensable are removed.

Working in such an evolutionary environment requires the

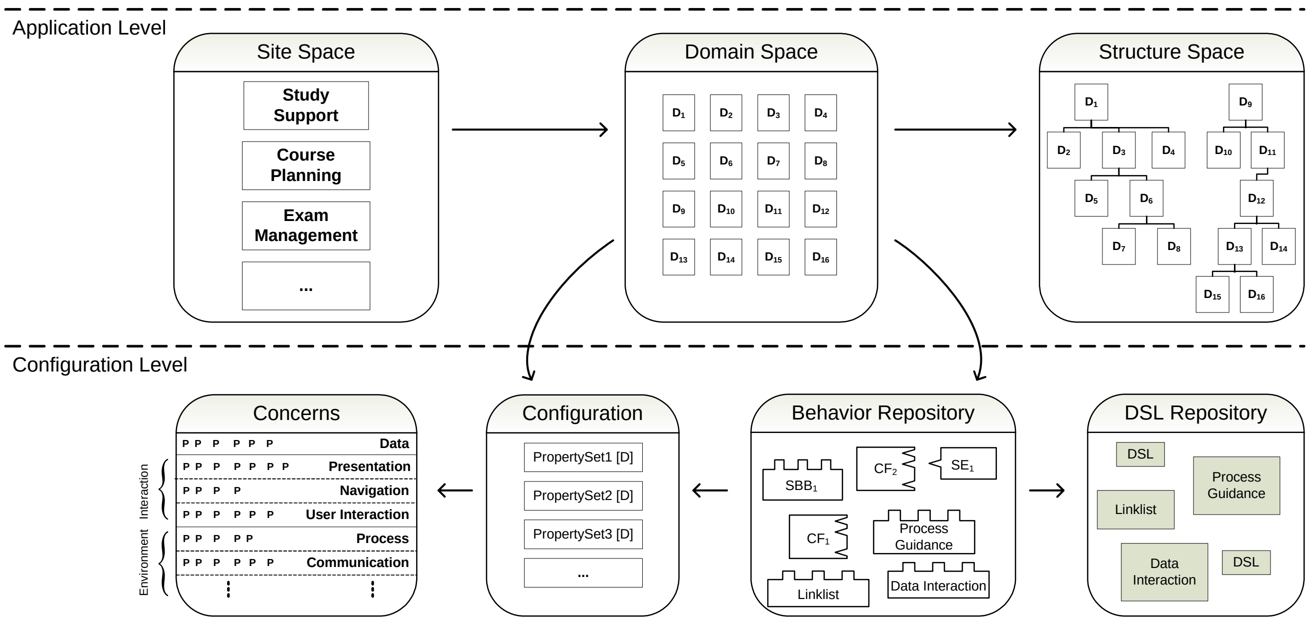


Fig. 2: A snapshot of the technical support system

ability to accept deficiencies in a positive way and address them as indicators for change. Principles from the field of total quality management like the Continuous Improvement Process (CIP) [17] would be unthinkable without a positive attitude to deficiencies and change. In addition, due to focusing on DSLs being tailored for small problem domains, an efficient management approach for the resulting multitude of DSLs is important.

Thus, a DSL framework encompassing all available DSLs and providing means for their systematic usage and management with a focus on evolution is needed. In the following, we outline two cornerstones of our approach to such a DSL framework: A DSL Repository for the systematic management of DSLs from the technical point of view and a team role called DSL Librarian being responsible for their efficient management and usage from the process perspective.

The DSL Repository is the central place for organizing, storing, managing and accessing DSLs and their components as well as associated metadata. Moreover, the repository must provide features for versioning to be able to cope with the continuous evolution of the stored components. In order to assure efficient storage and retrieval of DSLs, a sophisticated classification scheme supporting context-based searches is necessary. For example, it should be possible to find DSLs according to parameters like the problem domain, the application type, the kind of stakeholders etc.

The DSL Librarian accompanies the entire DSL lifecycle and promotes their usage. During the specification of new DSLs, she advocates the project team and is responsible for the avoidance of duplicate or badly reusable DSLs. Concerning the use of DSLs, she supports the project team in finding and using appropriate DSLs for the given problem domain and stakeholder group. Furthermore, she is responsible for the efficient maintenance of the repository which includes tasks like adding and removing components, updating the classification scheme as well as monitoring successful and unsuccessful searches and adoptions. With respect to the described tasks, the role of the DSL Librarian should be assigned to a team member who is very experienced in the field of DSL-based Web Engineering.

IV. Technical Platform

The technical support system used to execute a DSL according to section 3 is a framework called WebComposition Service Linking System (WSLS) [16]. The bird's eye view on the architecture of this framework is depicted in Fig. 2. Logically it is divided into two dimensions: the application level and the configuration level. Basically, WSLS is founded on top of the ideas introduced by the WebComposition approach [18]. Hence, WSLS supports the systematic development and evolution of Web applications by reusing existing software artifacts to reduce costs and increase their quality. Consequently, reusability is a guiding principle for the WSLS framework. A second central aspect is to establish the "configuration instead of programming" paradigm.

The continuous evolution of Web applications takes place in ever shorter periods and demands for more complexity at the same time. It is therefore urgent to provide approaches eliminating or at least reducing these problems, making the process of evolution faster and more flexible. As this is often aggravated by the wish of stakeholders to have short reaction cycles, like well known from agile methods like XP or SCRUM, exchanging the behavior by reconfiguring an application gets even more attractive. Furthermore, the possibility of rapid reconfiguration allows for "Quick Previews" of the application, which can help to increase customer satisfaction while reducing misunderstandings due to insufficient conceptual modeling.

Application Level: As shown in Fig. 2, the application level consists of three spaces: Site Space, Domain Space and

Structure Space. The Site Space manages all applications defined in WSLS. Each application contains one ore more domains defining the general application structure (Structure Space). These domains reside in the Domain Space which provides means for their management and configuration. In WSLS, applications are developed applying a reuse strategy often referred to as "development with reuse" or "consumer reuse" (composing and configuring reusable components), whereas domains themselves are developed conducting "development for reuse" or "producer reuse" respectively.

Configuration Level: In WSLS, behavior is subject to configuration. In the Behavior Repository three types of components reside: Solution Building Blocks (SBBs), Control Functions (CFs) and Service Elements (SEs). A Solution Building Block consists of a Control Function that combines additional involved components, so-called Service Elements, shaped to a dedicated concern like data, presentation or navigation. Thus, a CF acts as the design pattern mediator [19], whereas service elements realize the decorator and command patterns. For a further discussion on these components, see [16]. The behavior of a SBB can be influenced by configuring it with a DSL program, which corresponds to the Domain Abstract Representation (DAR). A SBB is consuming a DAR to execute it and provide a physical impact which is rendered by WSLS.

The *Separation of Concerns* paradigm is deeply anchored in the framework and therefore forms a fundamental pillar. Concerns in WSLS are coarsely classified in *Data*, *Interaction* and *Environment* and embodied as typed name-value pairs, so called *Properties*. Data affects the information space of an application and therefore comprises models, techniques and concepts. These reach from questions like what kind of data is found where and can be accessed how. An example could be a Web service to provide data, defined by the parts of a Web Service Description Language (WSDL) document: the concrete endpoint specifies the where, the types define what kind of XML-schemed business objects are provided and the abstract endpoint states how to access the data. An example of a data concern and its corresponding configuration in WSLS is given in section V.B. While Interaction covers aspects around presentation, navigation or user interaction, Environment comprises concerns like communication, process design or security, to mention just a few. Especially in EAI projects the Environment concerns become important, because the integration of existing (legacy) applications with the formalization of business processes in mind is a key factor.

V. DSL CATALOGUE

The need for simplicity bears a multiplicity of DSLs tailored to the different aspects of a distributed Web-based solution as well as special communication requirements for a specific group of stakeholders. In the following, we describe a selection of three DSLs out of our catalogue concerning three different, important dimensions of an EAI project: navigation and structuring of domains, data interaction using Web services and Web-based process guidance. The presentation of each DSL is divided up into a statement about the problem domain and the description of the Domain-Specific Model and a Domain Interaction Model.

A. Link List

Problem Domain: The concept of linking pieces of an application is inherently given to Web applications. This also applies to EAI projects where parts of an integrated system landscape are composed to new application domains through appropriate linking structures (e.g. menu or index). Thus, a DSL for the specification of linking structures is needed.

Domain-Specific Model: Fig. 3 illustrates the recursive definition of the Link List in a UML class diagram. It is capable of defining nested levels that can group aggregated links, each of them pointing to a part of the Web application. There exist two types of links: external links and domain links. External links are suitable to link to resources outside the Site Space. Domain links in turn refer to the internal site structure and point to resources within the Site Space.

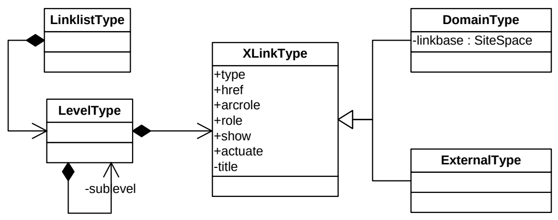


Fig. 3: UML class diagram that clarifies the relationship between nested levels and links.

The corresponding XML Schema Definition was designed according to the UML class diagram. The Link List schema is based on XLink [20], providing a standardized way for specifying links to resources in XML documents. A good overview about XLink and its benefits for navigation in hypermedia applications is given in [21]. The *linkListType* (1) aggregates one ore more levels, which in turn consist of a set of links and further sublevels. The cardinalities on the XML elements are omitted to achieve a better readability.

```
<xs:complexType name="linkListType">           (1)
 <xs:sequence>
  <xs:element name="level" type="tns:LevelType"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="levelType">              (2)
 <xs:sequence>
  <xs:choice minOccurs="0">
   <xs:element name="domain" type="domainType"/>
   <xs:element name="external"
    type="ExternalType" />
  </xs:choice>
  <xs:element name="sublevel" type=" LevelType" />
 </xs:sequence>
 <xs:attribute name="label" type="xs:string" />
</xs:complexType>
```

A link is realized as a *domainType* (3) or an *externalType* (4), which are equipped with XLink attributes to control the traversal and behavior of the linked domains. By specifying `xlink:show="embed"`, the Solution Building Block consuming the DSL program will embed the target domain and preserve the Link List, allowing for the realization of hypermedia design patterns (e.g. landmark or menu patterns). The statement `xlink:show="replace"` yields in a replacement of the Link List with the target domain. While such a behavior is feasible for internal domains, external resources should not be embedded, but rather be presented in a new context, which is expressed by `fixed="new"` (4).

```
<xs:complexType name="domainType">                    (3)
 <xs:attribute name="linkbase" type="xs:string" />
 <xs:attribute ref="xlink:title"/>
 <xs:attribute fixed="simple" ref="xlink:type" />
 <xs:attribute ref="xlink:href" />
 <xs:attribute ref="xlink:arcrole" />
 <xs:attribute ref="xlink:role" />
 <xs:attribute ref="xlink:show" />
 <xs:attribute fixed="onRequest"
  ref="xlink:actuate" />
</xs:complexType>
```

```
<xs:complexType name="externalType">                  (4)
 <xs:attribute ref="xlink:title"/>
 <xs:attribute fixed="simple" ref="xlink:type" />
 <xs:attribute ref="xlink:href" />
 <xs:attribute ref="xlink:arcrole" />
 <xs:attribute ref="xlink:role" />
 <xs:attribute fixed="new" ref="xlink:show" />
 <xs:attribute fixed="onRequest"
  ref="xlink:actuate" />
</xs:complexType>
```

Domain Interaction Model: As a first step towards a corresponding DIM, we designed a plug-in for Microsoft Visio based on the symbols shown in Fig. 4.

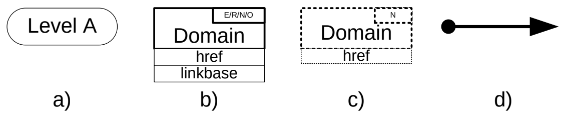


Fig. 4: (a) A level groups a set of links. (b) A box with a solid line symbols a domain, which resides inside the domain space. (c) A box with a dotted line represents an external link, which resides outside the application. (d) A connector realizes the aggregation of links and nesting of levels.

This paved the way for the collaborative specification of linking structures in our solutions with the involved stakeholders. The graphical notation uses levels to group a set of links (a). There are two types of links defined: linking to resources inside the application, called Domain Links (b) and linking to external resources that are not part of the solution (c). Furthermore, a link indicates the desired presentation of the target domain on traversal from the Link List according to the XLink specification: new, embed, replace and other. Section 6 will give an example of a Link List realized with the graphical notation introduced here.

B. Data Interaction

Problem Domain: The interaction with the information space is especially in an EAI project a key factor. For specifying the integration of and the interaction on various data sources from different applications of usually heterogeneous data types in a Web portal, a dedicated DSL is needed.

Domain-Specific Model: The Domain-Specific Model, which interacts with the information space of our solution, is based on the extensive use of Web services. Within the KIM project, we preferred a small set of standardized generic interfaces instead of a huge amount of specialized interfaces. This facilitates the realization of a multi-tier Web service-based SOA. Technically spoken, the Domain-Specific Model relies on an interface type called *CRUDS*, which stands short for Create, Read, Update, Delete and Search. This type of interface will typically not solve all problems, but helps to abstract and achieve standardization for most of the data centric tasks we are faced in KIM: finding, displaying and modifying data. Furthermore, such a generic interface allows for uniform access to Web services which also contributes to a high quality of the solutions being built [22].

Fig. 5 depicts the general structure of our DSM which is built upon the primitives of the CRUDS interface, focusing on the update perspective due to a better readability. The DSM includes a *primitivesType* (1) aggregating the distinct types “createType”, “readType”, “updateType”, “deleteType”, and “searchType” corresponding to CRUDS.

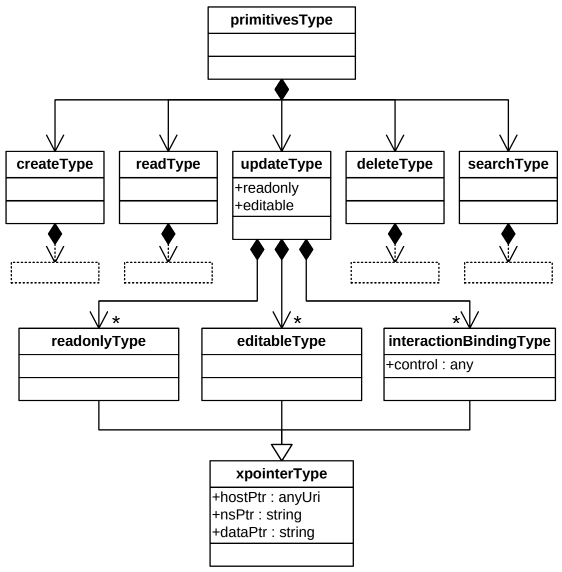


Fig. 5: UML class diagram visualizing the Data Interaction DSM which correlates to the primitives of the CRUDS interface and focuses on the “updateType”.

```
<xs:complexType name="primitvesType">          (1)
 <xs:element name="create" type="createType" />
 <xs:element name="read"   type="readType"   />
 <xs:element name="update" type="updateType" />
 <xs:element name="delete" type="deleteType" />
 <xs:element name="search" type="searchType" />
</xs:complexType>
```

Each primitive type allows for the specification of how to process a set of referenced data elements. These are referenced using expressions based on the XPointer framework [23] and executed on the WSDL type specification of the associated Web service. The construction of the XPointer expression (2) at runtime is realized by the substitution of defined attributes from the “xpointerType” (3).

```
$hostPtr?WSDL#xpointer(/definitions/types/     (2)
schema[targetNamespace="$nsPtr")xpointer($dataPtr)
```

```
<xs:complexType name="xpointerType">               (3)
 <xs:attribute name="hostPtr" type="xs:anyUri"/>
 <xs:attribute name="nsPtr"   type="xs:string"/>
 <xs:attribute name="dataPtr" type="xs:string"/>
</xs:complexType>
```

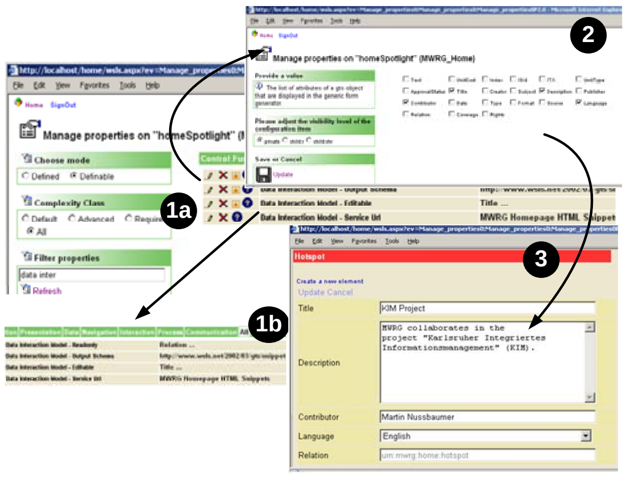
Due to space restrictions, the following detailed presentation of the Data Interaction DSM is confined to the update primitive. The *updateType* (4) extends the "xpointerType" and comprises additional types specifying which of the referenced data elements are read-only or editable. So far, "read-only" and "editable" were sufficient, but, as mentioned in section 3.2, our DSLs are subject to a systematic and continuous evolution.

```
<xs:complexType name="updateType">                 (4)
<xs:sequence>
 <xs:element name="readonly" type="readonlyType"/>
 <xs:element name="editable" type="editableType"/>
</xs:sequence>
</xs:complexType>
```

Beyond that, the "interactionBindingType" (5) accomplishes the mapping between elements of the referenced data object's schema and user interaction controls of a particular form model, e.g. XForms [24].

```
<xs:complexType name="interactionBindingType">(5)
 <xs:complexContent>
  <xs:extension base="xpointerType">
   <xs:sequence>
     <xs:element name="control" type="xs:any" />
    </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

Domain Interaction Model: Our technical platform WSLS, introduced in section 4, supports the definition of Property Sets that can be applied to configure a Solution Building Block. In an initial step, we developed a Property Set corresponding to the Data Interaction DSM to realize a Domain Interaction Model integrated in WSLS. The Property Set includes Properties for the Web service URL ("hostPtr"), the data schema of the considered type from the WSDL document ("nsPtr"), the sets of editable and read-only data elements as well as the interaction bindings. Fig. 6 depicts a typical scenario how the DIM can be used. The Property Editor (Fig. 6: 1a/1b) allows for manipulating Properties and thereby configuring the behavior of the Data Interaction SBB (Fig. 6: 1a). In Fig. 6: 2, the Property "editable" belonging to the "updateType" is edited. The displayed list of data elements is constructed based on the XML Schema Definition of a previously selected type ("nsPtr"). At runtime, the SBB can transform each checked data element into an XPointer expression according to the DSM as a triple consisting of the service url ("hostPtr"), the data schema ("nsPtr") and the data element ("dataPtr"). In this way, the SBB addresses the editable and read-only data elements and presents them to the user with the form controls specified via the "interaction bindings" Property (Fig. 6: 3).



Data Interaction Model - Readonly | Relation ...
Data Interaction Model - Output Schema | http://www.wsls.net/2002/03/gts/snippet
Data Interaction Model - Editable | Title ...
Data Interaction Model - Service Url | MWRG Homepage HTML Snippets

Filter properties: data inter

Title: KIM Project

Description: MWRG collaborates in the project "Karlsruher Integriertes Informationsmanagement" (KIM).

Contributor: Martin Nussbaumer

Language: English

Relation: urn:mwrg:home:hotspot

Fig. 6: A Domain Interaction Model for the Data Interaction DSL integrated in WSLS.

C. *Web-based Process Guidance*

Problem Domain: Software, especially in EAI projects, is increasingly being constructed by composing existing components on the basis of a loosely coupled framework. Thereby, a major focus lies on means for describing Web service choreography or orchestration respectively, whereas the aspect of user guidance in Web applications often is neglected. Hence, a DSL for specifying the user guidance process, i.e. the projection of the user interaction defined in a business process to the guidance through various components of a Web application, is needed.

Domain Specific Model: Heterogeneity of Business Process Modeling (BPM) techniques is a notorious problem for business process management. Although the discussion about the standardization lasts for more than ten years [25], the lack of a commonly accepted interchange format is still the main burden to business process management (see e.g. [26]). Based on our experience during the process analysis in the KIM project we found the common denominator in Finite State Machines. A Finite State Machine (FSM) describes the behavior of involved states, transitions and actions [27]. It represents a closed formal concept that allows us to describe workflows found in the course of our project. The Process Guidance DSM is equipped with XLink attributes to express the behavior, the traversal rules and its semantics. Furthermore, XLink encapsulates the linking structure from a business process and thereby leads to a modularized linkbase. Thus, advanced concepts like personalization or even business process federation can be achieved by reconfiguring the FSM by exchanging the linkbase.

In the following, the schema of the DSM is explained in more detail. A *workflowType* (1) describes the part of a business process which specifies a Web-based workflow. It consists of two or more *workstepTypes* that represent the states of the FSM and can be instances of e.g. a Data Interaction SBB or other available Solution Building Blocks. These workstepTypes (2) are realized as `xlink:type="locator"` which allows for reusing a domain by addressing it within the

Domain Space, specifying the `linkbase` and the `xlink:href` attributes.

```
<xs:complexType name="workflowType">          (1)
 <xs:attribute ref="xlink:type" fixed="extended"/>
 <xs:attribute ref="xlink:role" />
 <xs:sequence>
  <xs:element name="SBB" minOccurs="2
  type="workstepType" />
 </xs:sequence>
</xs:complexType>

<xs:complexType name="workstepType">            (2)
  <xs:attribute ref="xlink:type" fixed="locator"/>
  <xs:attribute ref="xlink:title" />
  <xs:attribute ref="xlink:role" />
  <xs:attribute ref="xlink:href"/>
  <xs:attribute name="linkbase" type="xs:string"/>
  <xs:attribute name="final" type="xs:boolean" />
</xs:complexType>
```

The *transitionType* (3) finally describes the transition formulized as an `xlink:type="arc"` that connects two states specified by `xlink:from` and `xlink:to` activated by a `token`. In addition to behavioral aspects regarding the activation of transitions, `xlink:actuate="onLoad"` can be used to indicate spontaneous state transitions allowing for executing a continuous chain of worksteps, also known as ε-transitions in non-deterministic state machines. In contrast, transitions that require a user input, are expressed with an `xlink:actuate="onRequest"`.

```
<xs:complexType name="transitionType">          (3)
  <xs:attribute name="token" type="xs:NMTOKEN" />
  <xs:attribute ref="xlink:type" fixed="arc" />
  <xs:attribute ref="xlink:from" />
  <xs:attribute ref="xlink:to" />
  <xs:attribute ref="xlink:arcrole" />
  <xs:attribute ref="xlink:actuate" />
</xs:complexType>
```

Domain Interaction Model: In the KIM project, all business processes were gathered by interviewing various stakeholders. The essence of these conversations was modeled with Petri nets, our Domain Interaction Model, using a tool called INCOME [28] as editor. Each business process is described in a comprehensive manner, which means that both, worksteps performed by users, often referred to as Human Workflows, as well as aspects concerning service automation like choreography or orchestration, are covered. Although the resulting Petri net models in INCOME are much more expressive than necessary to satisfy the DSM, using their XML representation and transforming them to a DAR according to our DSM has a couple of advantages: on the one hand side, reuse of already existing (partial) process information becomes possible, increasing the project team's productivity. On the other hand side, the modeling tool is well known by the domain experts who describe the processes, so there is no extra effort for them to learn a new one.

VI. DSL Applied – Study Assistance

In the following, we present our approach in practice in a typical scenario of the KIM project including linking structures, user workflows and data interaction. The KIM project deals with the university-wide optimization of business processes and the integration of supporting information systems. The realization is based on a multi-tier Web service approach and applies the service orientation paradigm, weaving together the existing systems employing state-of-the-art Web service technologies. The current, two-year project phase deals with the business processes involved in course planning, examination management and study support systems. Below, we are going to take a closer look at the study support system.

The study support system aims at assisting students with the planning, organization, monitoring and accomplishment of their studies. Therefore, it provides services like a personal information center, a personal student file, tools for evaluating the individual study progress and planning of next steps, features for course registration or a service for finding adequate learning partners. Following, we outline exemplarily the development of the course registration feature using the three DSLs presented in the preceding section. First, we show how to set up the study support system's menu using the Link List DSL. Afterwards, the user interaction process is realized employing the Web-based Process Guidance DSL. Finally, the single user interaction work steps are implemented by means of the Data Interaction DSL.

A. *The Application Structure*

In order to enable the various stakeholders to directly contribute to the solution, we customized Microsoft Visio with dedicated support for Link List diagrams according to the DSL introduced in section 5.

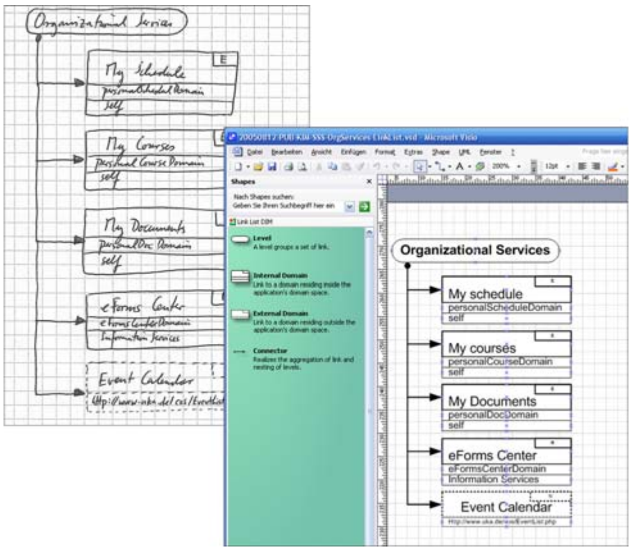


Fig. 7: Specifying the Link List DAR with pen and paper or in Microsoft Visio using a dedicated stencil.

Microsoft Visio is a graphical editor and allows for the placement of the predefined model elements and their annotation with additional attributes, according to the previously presented Domain-Specific Model (cf. Fig. 7). Based on the XML-export provided by Visio, we added an XSL(T)-based transformation engine to generate the corresponding Domain Abstract Representation XML from the diagram. This in turn can be used to configure the associated Solution Building Block which renders the menu at runtime (cf. Fig. 9, left hand).

B. Applying a Business Process

Having defined the general structure of our application, the sub-domains of the functional area “Organizational Services” must be filled with adequate behavior. Following, we take a closer look at the course registration feature “My Courses”. The associated user interaction workflow is depicted in Fig. 8 in form of a Petri net. The illustrated scenario represents the support for students when they customize their schedules by subscribing to courses. The Petri net business process acquired via the XML export of INCOME is transformed to the DSL’s corresponding DAR, as introduced in section V.C, using XSL(T). Finally, a dedicated Solution Building Block takes the DAR to realize the workflow. Therefore, a SBB is assigned to each state of the workflow, and user controls enabling users to activate the transitions between them are provided. The two states, “Subscribe to course” and “Review Schedule” are depicted in Fig. 9.

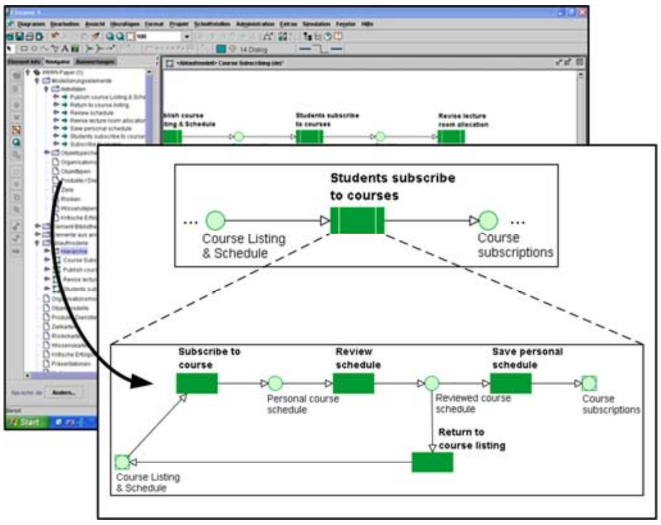
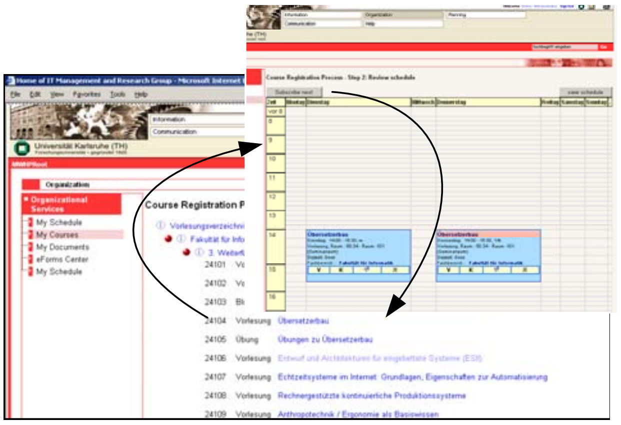


Fig. 8: Editing the Web-based User Guidance DAR in INCOME with Petri nets.

C. Integrating Web Services

The single worksteps of the process guidance shown above are Solution Building Blocks whose behavior can be configured by means of the Data Interaction DSL. In our case, “Subscribe to course” and “Review schedule”, represent a standardized communication with Web services according to section V.B. Fig. 9 shows the situation of the study support system, when a student has subscribed to a course and reviews the resulting schedule. The Web services facilitating this behavior are CRUDS-based and therefore allow the management of a student’s schedule by creating (C), updating (U) or deleting (D) registrations. Hence, the corresponding SBB is configured to offer appropriate user interaction controls and to communicate with these Web services by providing the DSM’s primitives within the WSLS Property Editor.



```
24101   Vo
24102   Vo
24103   Bl
24104   Vorlesung   Übersetzerbau
24105   Übung       Übungen zu Übersetzerbau
24106   Vorlesung   Entwurf und Architekturen für eingebettete Systeme (ESI)
24107   Vorlesung   Echtzeitsysteme im Internet: Grundlagen, Eigenschaften zur Automatisierung
24108   Vorlesung   Rechnergestützte kontinuierliche Produktionssysteme
24109   Vorlesung   Anthropotechnik / Ergonomie als Basiswissen
```

Fig. 9: The Course Registration in the Study Support Portal.

VII. SUMMARY & FUTURE WORK

In this paper, we presented our approach to DSL-based Web Engineering which emphasizes conceiving communication with stakeholders and Web application development in a holistic way. In our large-scale EAI project “Karlsruhe’s Integrated Information Management (KIM)”, we were faced with communication problems between the developers and the stakeholders from all over the university. We identified the great diversity and low availability of stakeholders as a major drawback making it infeasible to employ complex modeling languages for the specification of distributed Web-based solutions. We argued that, in such environments, simplicity is a key factor for an efficient usability of languages. The vision of our solution is to enable stakeholders to directly contribute to the development effort by understanding, validating and specifying parts of the application.

We proposed an approach consisting of a framework of Domain-Specific Languages (DSL) and an underlying technical platform. DSLs are small, simple and highly focused languages for solving clearly identifiable aspects of a distributed Web-based solution. They are easy to understand and learn and incorporate individual notations for various stakeholder groups. Furthermore, we outlined the cornerstones of our approach to the systematic management and evolution of the emerging multitude of DSLs – a DSL Repository and a DSL Librarian. Following, we presented the underlying technical framework being mainly responsible for the transformation of DSL programs into executable parts of a Web application. In this regard, we introduced Solution Building Blocks (SBBs) as software components whose behavior can be configured by DSL programs. The WebComposition Service Linking System (WSLS) served as technical platform facilitating the systematic composition and configuration of SBBs. Thus, Web applications can be built in an evolutionary manner by composing SBBs and configuring them with DSL programs which in turn are specified using dedicated editors.

Subsequently, we presented a selection of three core DSLs concerning three important dimensions in an EAI project: navigation and structuring of domains, data interaction using Web services, and Web-based process guidance. For each DSL we described its target problem domain and introduced the associated Domain-Specific Model and an adequate Domain Interaction Model. Finally, we showed a practical realization of our approach in the KIM project. We explained how the course registration feature in a study support portal was built employing the previously introduced DSLs on the basis of our technical framework.

For the future, we are planning to extend our approach in several directions. First of all, we will continuously evaluate our existing DSLs in practice, leading to an evolutionary improvement and development of new DSLs. Beyond that, we will pursue the idea of performing validation and optimization at domain level. Especially for DSLs concerning the dimensions navigation, presentation and user interaction this seems to be promising. For example, rule sets assuring the accessibility of a Web application's presentation aspects could be included in the DSM or DIM and enforced by accompanying editors. A further advancement would be an ontology-based DSL repository facilitating their systematic storage, management and retrieval. Finally, an integrated development environment for our approach would be desirable. Therefore, we are going to evaluate the DSL Tools delivered with Microsoft Visual Studio 2005.

References

[1] Fowler, M., Language Workbenches: The Killer-App for Domain Specific Languages? - 2005): http://www.martinfowler.com/articles/languageWorkbench.html.

[2] The Standish Group International, Extreme Chaos - Research Report (2001): http://www.standishgroup.com.

[3] Ceri, S., Fraternali, P., and Bongio, A. Web Modeling Language (WebML): A Modeling Language for Designing Web Sites. in 9th International World Wide Web Conference (WWW). 2000. Amsterdam, Nethderlands.

[4] Mcclure, C.L., Software reuse techniques : adding reuse to the system development process. 1997, Upper Saddle River, N.J.: Prentice Hall. xxiv, 350.

[5] Schwabe, D., Rossi, G., and Barbosa, S. Systematic Hypermedia Design with OOHDM. in ACM International Conference on Hypertext' 96. 1996. Washington, USA.

[6] Wiegers, K.E., Software Requirements. Second ed. 2003: Microsoft Press. 430 pages.

[7] Deursen, A.V., Klint, P., and Visser, J., Domain-Specific Languages: An Annotated Bibliography. ACM SIGPLAN Notices, 2000. 35(6): p. 26-36.

[8] Kieburtz, R.B., et al. A software engineering experiment in software component generation. in Proceedings of the 18th International Conference on Software Engineering. 1996: IEEE Computer Society Press.

[9] Batory, D., Lofaso, B., and Smaragdakis, Y. JTS: Tools for Implementing Domain-Specific Languages. in Proceedings of the Fifth International Conference on Software Reuse. 1998: IEEE.

[10] KIM Project Homepage - 2005), University of Karlsruhe: http://www.kim.uni-karlsruhe.de/ (02.11.2005).

[11] Chappell, D.A., Enterprise Service Bus. First ed. 2004: O'Reilly Media, Inc. 247.

[12] Zimmermann, O., Krogdahl, P., and Gee, C., Elements of Service-Oriented Analysis and Design - Web Site (2004): http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/ (30.07.2005).

[13] Meinecke, J., Gaedke, M., and Nussbaumer, M. A Web Engineering Approach to Model the Architecture of Inter-Organizational Applications. in Conference on Component-Oriented Enterprise Applications (COEA 2005). 2005. Erfurt, Germany: Gesellschaft für Informatik.

[14] Dmitriev, S., Language Oriented Programming: The Next Programming Paradigm - 2005), JetBrains: http://www.onboard.jetbrains.com/is1/articles/04/10/lop/.

[15] Ward, M., Language Oriented Programming, in Software - Concepts and Tools. 1994. p. 147-161.

[16] Gaedke, M., Nussbaumer, M., and Meinecke, J., WSLS: An Agile System Facilitating the Production of Service-Oriented Web Applications, in Engineering Advanced Web Applications, S.C. M. Maristella, Editor. 2005, Rinton Press. p. 26-37.

[17] Dale, B.G., Managing Quality. 2003: Blackwell Publishing.

[18] 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.

[19] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., Design patterns: elements of reusable object-oriented software. Addison-Wesley professional computing series. 1995, Reading, Mass.: Addison-Wesley. xv, 395.

[20] Derose, S., Maler, E., and Orchard, D., XML Linking Language (XLink) Version 1.0. 2000, World Wide Web Consortium.

[21] Christensen, B.G., Hansen, F.A., and Bouvin, N.O. Xspect: Bridging Open Hypermedia and XLink. in 12th Intl. World Wide Web Conference. 2003. Budapest, Hungary.

[22] Ibm, Elements of Service-Oriented Analysis and Design - 2005), IBM Homepage: http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/.

[23] Grosso, P., Maler, E., Marsh, J., and Walsh, N., XPointer Framework - W3C Recommendation (2003), W3C Consortium: http://www.w3.org/TR/xptr-framework/.

[24] Dubinko, M., Leigh L. Klotz, J., Merrick, R., and Raman, T.V., XForms 1.0 - W3C Recommendation (2003), W3C: http://www.w3.org/TR/2003/REC-xforms-20031014/.

[25] Hollingsworth, D., Workflow Management Coalition: The Workflow Reference Model. 1995, Winchester.

[26] Group, D., BMP 2003- Market Milestone Report - 2003), Delphi Group: http://www.bpminstitute.org/whitepapers/whitepaper/article/delphi-group-2003-bpm-market-milestone-report.html.

[27] Wikipedia, Finite State Machine - Definition (2005): http://en.wikipedia.org/wiki/Finite_state_machine.

[28] Lausen, G., Nemeth, T., Oberweis, A., Schönthaler, F., and W., S. The INCOME approach for conceptual modelling and prototyping of information systems. in CASE - The 1st Nordic Conference on Advanced Systems Engineering. 1987. Stockholm, Sweden.