Supporting Pattern-based Application Authoring for the Semantic Web

Fuchs Michael, Niederée Claudia, Hemmje Matthias

{fuchs,niederee,hemmje}@ipsi.fraunhofer.de

Fraunhofer IPSI, Dolivostrasse 15, 64293 Darmstadt

Abstract

The SWAN approach, presented in this paper, addresses the challenging task of developing Semantic Web Applications, i.e., Web applications that fully and effectively exploit and serve the Semantic Web as their operational environment. It is based on the support of two types of models, semantic domain models and conceptual user interface models, as well as on flexible model mappings. The SWAN framework, implementing the SWAN approach, contains tools for pattern-based Web applications authoring. SWAN user interface design patterns reflect frequent dialogue sequences and ease user interface construction as well as interpretation. Furthermore, each SWAN design pattern is associated with mapping constraints, which restrict mappings between user interface and domain model elements.

1. INTRODUCTION

The Semantic Web promises exciting new applications in the Web as well as improved task and cooperation support [1]. However, for Web applications to effectively operate in the Semantic Web environment they should serve as well as exploit the special characteristics of this new operational context. The Semantic Web vision, thus, imposes new challenges on the design and development of such Semantic Web applications: First of all, Web applications move from a purely human user community towards a mixed user community consisting of humans as well as software agents. Secondly, automatic interpretation of content, one of the main building blocks of the Semantic Web, is based on interlinking local domain models with globally defined interpretation schemes like vocabularies and ontologies.

Building upon innovative Web technologies and standards, our approach for Semantic Web Application development and iNtegration (SWAN approach) combines the following building blocks:

- Semantic domain models are semantically-enriched representations of the domain model underlying an application. They meet the requirement of interlinking local models with global schemata enabling interpretation in a global context;
- Conceptual UI models describing the interactions of users with the system on a conceptual level meet the requirement of representing interaction with humans as well as with software agents;
- UI design patterns describe frequent patterns of UI dialogues that represent larger sequences of interaction activities (cf. the shopping cart metaphor). The reuse of such pattern eases conceptual UI design and reduces the "mental effort" for users by relying on similar user interaction pattern.

In addition, flexible mappings between the different types of models are crucial building blocks of the model-based SWAN framework.

Design patterns (e.g., [2, 3]) are a medium of easing communication between designers, of establishing best practice in design, and of contributing to design quality. The type of design patterns applied for the SWAN approach are macro UI design patterns comparable to the application framework pattern described in [4]. The SWAN approach augments UI design pattern with domain model constraints. These constraints restrict the possible mappings between the conceptual UI model and the underlying (semantic) domain model and can be used to guide the definition of such mappings in building Semantic Web Applications. The SWAN approach is implemented in the SWAN framework relying on the form-based UI paradigm and following a meta-design approach [6], i.e., the SWAN system contains a suite of system authoring tools that are used in its own design and customization. This paper focuses on discussing the SWAN approach and the role of the Macro UI Design Pattern in this approach. It also gives a short overview over the SWAN framework and its realization. A catalogue of concrete design patterns will be presented in a future publication.

The rest of the paper is structured as follows: Section 2 discusses related work. Section 3 and 4 present the proposed approach, where section 3 describes the overall SWAN approach and the Semantic Web Application models, whereas section 4 discusses the role of UI design pattern in the SWAN framework. In section 5 we give an overview of the framework. The paper concludes with a summary and some plans for future work.

2. RELATED WORK

The approach discussed in this paper combines a design pattern approach with a model-based Semantic Web application approach. The related works for the two areas are introduced and discussed separately in this section. A complementing notion of Semantic Web Applications is used in [7]. Whereas we mean Web application effectively operating in the Semantic Web, in [7] applications that deal with RDF-based content in a user-friendly way are named Semantic Web Applications.

Especially, the programming language *Adenine* supports the RDF model as a built-in data type.

Several design pattern approaches and collections have been proposed within the software design and software engineering area since Christopher Alexander's architecture design pattern [2]. In [3], for example, a collection of successful solutions to recurring problems in object-oriented design has been published (see also critical discussion in [8]). With the explicit aim to include the user into the design process, Jennifer Tidwell [5] wrote a pattern language for HCI design which consists of about 50 design patterns for designing interactive systems. This collection of patterns is widely regarded as currently the most ambitious attempt at a UI pattern language. Other collections of UI design patterns can be found in [9, 10]. Furthermore [4] describes design patterns for Web application frameworks, which consists of a WebML [11] diagram to express the structure and navigation of site views and a structural schema diagram to specify the domain model view. In section 4, we use WebML as navigation description part of our pattern.

The pattern presented in this paper build upon HCI design pattern as they are proposed e.g. in [5]. However, they are not restricted to the HCI design process, but also support parts of the software design process. More precisely, they focus on the bridge between the HCI design and its mapping to the underling domain model.

The way we use our design pattern is most similar to that proposed in [12, 13, 14]. While we aim to have different levels of complexity within each pattern to fill the gap between user requirement and formal specification, [12] proposed an ontological mapping approach as a discipline of software architecture in addition to UI software design and software engineering. In this case a software architect acts as a mediator between designer and engineer in building up an ontological dictionary and also a workflow map of its recurring standard processes. [13], and [14] are proposing different patterns for each phase of the software design and development process addressing different groups of experts. In contrast to this we are more interested on how design pattern can be used to bridge the gap between two design tasks.

The SWAN framework introduced in this paper will follow the model-based software development approach (see e.g. [15, 16, 17, 18]). Mobi-D [15] is an interactive environment where declarative models can be connected. It distinguishes abstract and concrete models and supports mappings between them. In Mecano [16], a model-based UI development environment is introduced, which provides a tool for creating domain models. Based on this model, high-level dialogs (e.g. workflow and navigation structure of windows) as well as low-level dialogs (one step within the high-level dialog e.g. form input field, button) can be generated and later customized. However in contrast to our approach the domain model has to be constructed manually and is not directly coupled to existing application data. SUIMS [17], another model-based approach, is based on the so-called ART Schemata where a UI model is composed from objects, actions, parameter, attributes and their types, pre-conditions, and post-conditions as well as the corresponding relations between them. These schemata can be instantiated with parameters resulting in the creation of different types of UIs. To build up the schemata, a highly skilled programmer is needed, in contrast to our approach that includes a user-friendly system authoring tool suite.

Our conceptual UI model is based on the XForms standard [19]. Other formats for conceptual UI models have been proposed like e.g. the Ozone UI ontology in the Haystack framework [7]. This UI model is based on RDF easing the coupling with the domain model, but lacking the UI specific support of XForms (events, predefined controls, validation of input, etc.).

3. SEMANTIC WEB APPLICATIONS

Web Applications in the Semantic Web. The standard architecture for Web applications is based on some persistent data storage (e.g. a RDBMS). A vendor-independent protocol like JDBC makes the data accessible for the application layer. In the presentation layer, finally, the UI usually consists of a set of HTML-encoded pages. Technologies like JSP or ASP are used to dynamically create web pages including application data.

In the context of the Semantic Web the "users" of the interface are no longer restricted to humans, but also include software agents. If relying only on an HTML-encoded web page, a software agent cannot extract sufficient information

- to know **how** to enter into a dialog (i.e. an interaction) with the system which is in the case of an HTML-encoding mainly due to the lack of logical structure and due to the relaxed syntax rules.
- to infer **what** the (input) fields mean and which restrictions are available for qualifying the allowed input (constraints, types).

Humans use their cognitive abilities as well as their experience with similar types of Web applications for the interpretation of such web application UIs.

Semantic Web Application Models. This problem of "how" and "what" can be overcome by introducing two new types of models: on the one hand, *conceptual UI models* systematically describe possible interactions with the system on a conceptual level and, on the other hand, *semantic domain-models* describe application domains and relate model components to community-accepted ontologies and vocabularies. We are assuming form-based UIs and corresponding conceptual models that support the dialog and interaction with the user or agent in a well-structured and well-understood interaction paradigm.

Semantic Domain Model While all Web applications embody some type of a domain model that is underlying the application this model is usually made explicit in a systematic way at design time only, e.g., when creating a

UML class diagram. When the system is implemented the domain model is represented in a partly fragmented and partly duplicated way within the code of the application layer (e.g. Java classes) and the schema of the underlying relational database. Furthermore, the model is a local domain model relying on a conceptualization specific for the implementation of this application.

In the Semantic Web context the domain model has to be made explicit and it has to be set into relationship to a global ontology that is also accessible by agents that want to interact with the system. The RDF (Resource Description Framework) family (RDF Schema, OWL) supplies models and languages for developing domain models in a global context. RDF provides a data model for the description of resources in the Web. The associated schema language, RDF Schema, can be used for the definition of domain models and vocabularies. Modeling capabilities as they are required in SWAN approach are improved when RDF Schema is combined with the Web Ontology Language (OWL).

Conceptual UI Model The SWAN approach introduces an additional layer between the application layer and the actual UI that manages a conceptual model of the UI. This model describes the UI of the system on a conceptual level independent of the respective UI agent.

Conceptual UI models can be automatically transformed into UIs for specific types of UI agents (like e.g. for a Web browser, a PDA, or a mobile phone) by adequate processors introducing more flexibility with respect to the type of client that the UI is finally displayed on and avoiding the bias towards one type of UI paradigm. Moreover, by communicating the conceptual structure of the interface to, e.g., software agents in the Semantic Web can use it to learn how to use the interface, e.g., where are fields to fill and elements to select from a list. However, this requires a shared understanding and a corresponding conceptualization of UI functionality. XForms [19], the currently emerging standard for the next generation form-based UIs of Web applications, defines a model (XML-based syntax) for the description of UIs on a conceptual level which satisfies the aforementioned requirements. It also supports the definition of mappings between elements of the domain model and the conceptual UI. In addition, other standards of the XML family like XML, XML Schema, SOAP, XPath are, of course, of importance in the SWAN approach. They are e.g. used for information exchange, representation of intermediate information formats, and for component interaction following the Web service paradigm.

Interaction Design Pattern

The UI building blocks provided by a conceptual UI model language like XForms are on a rather elementary level. The aim of such languages is to provide flexible general purpose support for the construction of conceptual UI models. In practice, however, there are often interaction patterns that come up again and again. Such patterns are captured by introducing support for *interaction design patterns* into the SWAN framework. This aspect is discussed in more detail in section 4.

4. INTERACTION DESIGN PATTERNS IN SEMANTIC WEB APPLICATIONS

Interaction Design Pattern Benefits. UI design, as all other design, is positioned somewhere between creativity and relying upon approved methods and pattern. A prominent example for repetitive pattern in UI design from the e-Business domain is the shopping cart metaphor that encapsulates a certain type of UI and an underlying interaction pattern. A systematic way to reflect such repetitive pattern in the design process is the introduction of so-called design pattern that encode best practice in frequent design situations. The following advantages of UI design pattern are exploited in the SWAN approach:

- *Design Efficiency and Quality:* Making use of design pattern instead of starting each time from scratch clearly contributes to design efficiency. Complex UI solutions can be built up more systematically. Since the design patterns are based on approved "good" design their use can also contribute to design quality.
- *Improved Communication*: The proposed SWAN interaction design pattern define a vocabulary for UI designers to efficiently discuss about designs. Pattern Catalogues enriched with concrete UI examples can be used to effectively discuss UI designs with the user.

Furthermore, there are two additional benefits of supporting UI design pattern within the framework.

- *Mapping Constraints:* Constraints for the mappings between the conceptual UI and the domain can be identified based on an analysis of the respective UI design pattern; constraints hold for all UIs based on the respective design pattern and can, for example, be exploited for intelligent mapping support.
- *Shared Interaction Semantic:* A design pattern also supply an associated interaction pattern for the user of the respective UI. All UIs based on the same design pattern share the same interaction semantics. If there is an agreed upon set of design pattern the agent has to learn the pattern only once and can apply it to all UIs based on the same pattern. This holds true for humans as well as for software as users of the interface.

Design Pattern Example

We distinguish application design pattern reflecting a business process sequence in a specific domain and general design pattern that can be reused in different domains. This section introduces a simple general design pattern example that is used to illustrate our approach.

Our design pattern are composed from design pattern building blocks that themselves can be other design pattern or actual UI elements as they are e.g offered by XForms. In addition, we provide the context a design pattern can be used in as well as the constraints for mapping UIs based on the respective design pattern on an underlying domain model. In more detail, a UI design

pattern *DP* is described along the following n dimensions (following the description format proposed in [14]):

Name: Name of the design pattern *DP*. In the Semantic Web Context of the SWAN approach this should preferably be a URI, e.g. based on the URI of the pattern collection *DP* is part of.

Classification*:* Design patterns can be classified according to different criteria like purpose, interaction. type, and granularity easing the selection of adequate pattern. We will initially rely on existing classification schemes, refining them as our collection grows.

Problem: This dimension describes the UI design situations, in which *DP* can be applied and/or the UI design challenges that can be solved by applying DP.

Pattern Context: There are two types of requirements imposed by *DP* to its context: The *Pattern context* identifies design patterns providing a context for *DP*, i.e. design patterns that can contain or typically contain DP, or conditions for such containment (cf. context description in [14]).The second type of context, the ***Mapping requirements*** refer to constraints that a mapping between a domain model and a UI based on *DP* has to fulfill due to DP's characteristics.

Examples: One or more concrete UIs that are based on design pattern *DP*, illustrate the use of the design pattern and ease design pattern selection;

Description: This dimension describes the structure of *DP* and the interaction pattern it supports. Especially, the role of components of the *DP* is discussed.

Navigation Diagram: This dimension describes the navigation pattern and the dataflow within DP. The visual language WebML [11, 20] is used for this purpose.

The Design Pattern SelectionListToInteractiveDetail

***Name*:** SelectionListToInteractiveDetail

Classification: MultiPageDialog, FromRoughToDetail,…

Description: The Pattern consists of two components, where *Overview* acts like a Menu and *InteractionDetail* is the associated action. The action represents an interaction of a UI for modifying a certain Domain Object.

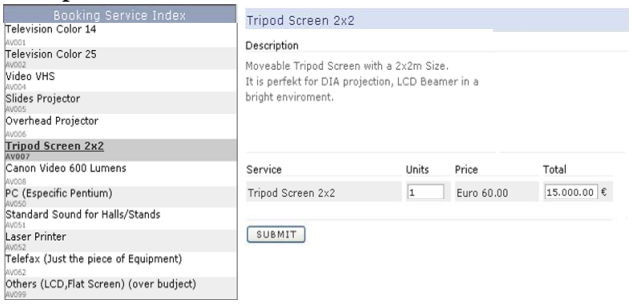
Problem: Interaction with one Item of a Collection

Pattern Context: BrowseAndInteractSystem

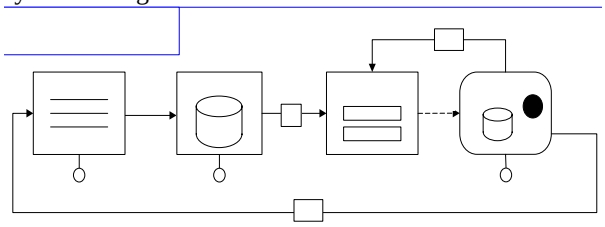
Mapping Requirements:

$$constraint_{SelectionListToInteractiveDetail}(Group,\ List,\ Collection)\ :=$$
$$\exists o_i\ \exists ref_j \in Collection\ :\ detail(ref_j) = o_i$$
$$\wedge\ constraint_{InteractiveDetail}(Group,\ o_i))$$
$$\wedge\ constraint_{SelectionList}(Group,\ List,\ Collection)$$

Example:



Navigation Diagram*:* Within the following WebML diagram, the unit *Selection List* passes on click the *Object ID* to the loader unit *Object* which automatically forwards it (with activity *A*) to the form unit *Interactive Detail*. After sending the modified *Object* to the data modifying unit *Update* there are two navigation possibilities. Either the transaction was *OK* and system will navigate to *Selection List* or it was *KO* (not successful) and the system navigates back to the form unit *Interactive Detail*



Identification of Mapping Constraints

We will illustrate the identification of mapping constraints by looking into concrete examples. We start with simple elements and proceed step-wise with more complex structures. In each step the structure from the previous step is used as a building block. This gives us the opportunity to also illustrate the propagation of mapping constraints from component to composite.

Before we start with the discussion we have to introduce some notations: For the domain model *D* we assume an OWL/RDF like structuring, where the model consists of a set of classes (*Classes)* and properties (*Properties*) that are used to model relationships between concepts as well as attributes of a class. We assume that there is a function *domain(p)* that, when applied to a property returns the domain of the property (cf. domain property of RDF Schema). For the conceptual UI model we assume that there is a set of control elements and a set of constructors. For each type of control element *CE* we assume a predicate *CE(e)* that is fulfilled for each control element *e* that belongs to the control element class *CE*.

We distinguish elements of the UI modeling language that can be set in relationship with the domain model, which we call controls, and elements that are independent of the domain model like labels. The following discussion is restricted to situations, where controls are mapped to domain model elements, since we are interested in the mappings and the associated constraints.

We start our discussion with the output control element, i.e. an element that is used to display information of the domain model. An output control object *e* is mapped to a domain object property *p*. Of course the value for an output control can also be composed from different properties, e.g. by string concatenation. However, we assume here that a derived property is created in this case before the mapping. The predicate *comp* means that the two elements are compliant with each other. For the case of the outputControl this means that the type of the element allows it to be displayed in the control element.

$$constraint_{OutputControl}(e_i,\ p_j)\ :=\ map(e_i,\ p_j) \Rightarrow comp(e_i,\ p_j)$$

An input control *e*, as it is implemented by input fields of a form, enables the user to input a character string. Again the control is associated with a domain model property. In addition, the mapping must enable the propagation of the user input into the properties of the domain model.

$$constraint_{InputControl}(e_i, p_j) := map(e_i, p_j) \Rightarrow (comp(e_i, p_j) \wedge writeable(p_j))$$

In the constraint we capture this by using a predicate *writable(p)* that is used to express that it is possible to change the value of the property p. This implies different things: we need a write function or a path expression that enables us to change the property value and it has to be allowed to change the property value. Examples of properties, whose values cannot be changed (directly), are derived properties (see above) or read-only properties.

Based on these two simple controls we can already create a first simple design pattern, which we call *InteractiveDetail*. This pattern is used to display or input values for the properties of one object. This pattern can for example be found in forms for entering your personal data like name, address, email, etc. The pattern is composed from a *GroupConstructor* that contains a set of *InputControl* or *OutputControl* elements or a mix of both plus typically some action element to complete/submit the interaction. For this pattern the following mapping constraint can be identified:

$$\begin{aligned} &constraint_{InteractiveDetail}(Group, o) := \\ &\forall e_i \in Group\ \exists p_j \in Properties\ :\ domain(p_j) = o \\ &\wedge\ (OutputControl(e_i) \Rightarrow constraint_{OutputControl}(e_i, p_j)) \\ &\wedge\ (InputControl(e_i) \Rightarrow constraint_{InputControl}(e_i, p_j)) \end{aligned}$$

The constraint states that all associated properties p_j ($1 \leq j \leq n$) have to belong to the same domain object o and that the constraints of the simple controls ($constraint_{OutputControl}$ or $constraint_{InputControl}$) have to be satisfied.

The pattern *InteractiveDetail* can be used in larger design patterns e.g. by combining it with an object selection list as it is done in the pattern *SelectionListToInteractiveDetail*, which has been described above.

Assuming that we built a mapping constraint for *SelectionList* (similar to *InteractiveDetail*) and furthermore there is a function *detail(ref)* which returns the details of an object *o* if *ref* is a reference to *o*. Then it is possible to combine the mapping constraints:

$$\begin{aligned} &constraint_{SelectionListToInteractiveDetail}(Group, List, Collection) := \\ &\exists o_i\ \exists ref_j \in Collection\ :\ detail(ref_j) = o_i \\ &\wedge\ constraint_{InteractiveDetail}(Group, o_i)) \\ &\wedge\ constraint_{SelectionList}(Group, List, Collection) \end{aligned}$$

Thus constraintSelectionListToInteractiveDetail is satisfied, if there is a list of references (which satisfy $constraint_{SelectionList}$) to objects which satisfy $constraint_{InteractiveDetail}$.

5. The SWAN Framework

SWAN Framework and its Architecture. The SWAN framework is an implementation of the SWAN approach. Its architecture enriches the traditional three layered Web application architecture layers by components for the management of the Semantic Web Application Models. Furthermore, components for the pattern-based definition and management of the mappings between models are included (see Figure 1). The semantic domain model is part of the application layer whereas the UI model is managed as part of the UI layer.

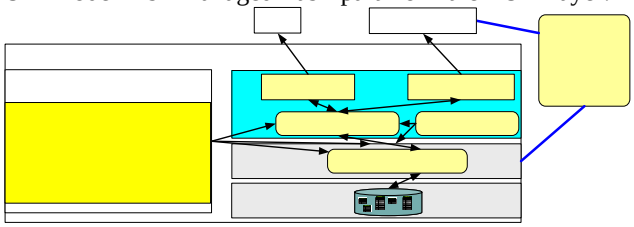


Figure 1: SWAN Framework Architecture

The mapping between the semantic domain model and the conceptual UI model propagates semantic information contained in the domain model into the UI. In defining such mappings, constraints coming from the underlying design pattern can be used to guide the process. Based on this propagation the semantic information can be made available to the agents of the Semantic Web.

The functionality of the framework is based on a flexible coupling process that implements the bi-directional mappings that are required to dynamically couple the different models of the SWAN architecture. This semi-automatic process is described in more detail in [21].

SWAN Authoring Tool Suite

The SWAN authoring tool suite completes the SWAN framework by providing system authoring tools for the definition of the models and mappings. It follows the idea of meta-design [6] enabling the design, customization, and evolution of an application solution without resorting to programming. In more detail, the tool suite consists of the following main components:

- ***Domain Object Mapping Manager (DOMM)***: This component extracts a default domain model from the database schema of an application and transforms it into an RDF-based representation. The rules governing this mapping are discussed in [21].
- ***VizCo***: The domain model can be restricted to relevant views to ease the task of defining mappings between the domain model and UI components. This is done with the authoring tool *VizCo*.
- ***Form Dialog Manager (FDM):*** The FDM is an authoring tool for the definition of conceptual user-interface models based on the XForms model and for the definition of mappings between elements of the conceptual UI and domain model views.

The SWAN authoring tool suite contains further tools, e.g. for the definition of multilingual and multi-role navigation structures based on taxonomies (see [22]). Currently an extension of the Form Dialog Manager is under development that enables the power user to select from a set of design pattern that is then used to provide an XForm template and to guide the mapping process.

6. Conclusions and Future Research

In this paper we presented the SWAN approach and framework for systematically developing Web Applications for the Semantic Web. The SWAN approach is based on the introduction of two types of Semantic Web Application models, conceptual UI models and semantic domain models, and the support of flexible mappings between these models. The approach is augmented by the use of UI design pattern that reflect frequent dialogue sequences and provide an additional higher-level layer for UI construction and interpretation.

Design pattern play an important role in guiding the UI design process as well as the definition mappings between the Semantic Web Application models, since SWAN design patterns are equipped with mapping constraints that restrict the set of possible UI-domain model mappings. Web applications built with the SWAN framework exhibit the semantic domain model as well as the underlying design pattern as part of their UI, thus, easing interpretation of and interaction with the UIs for human as well as for software agents.

A first prototype of the SWAN framework has been implemented and is currently evaluated in the context of an e-Business Web application. However, there are still several areas of future work. Currently, we are active in the definition of a language for the machine-readable representation of our UI design pattern, the definition and evaluation of a systematic catalogue of design patterns, the flexible integration of an extensible set of design patterns into the FDM tool and the implementation of an intelligent mapping definition support based on the mapping constraints identified for the design pattern. A further step for the exploitation of the design pattern in the Semantic Web context is the publication, and community-based evolution of a shared ontology for the description and classification of the design pattern.

References

[1] T. Berners-Lee, J. Handler, and O. Lassila. “The semantic web”. Scientific American, Special Issue on “Intelligent Systems/Tools In Training And Life-Long Learning“, 2001.

[2] C. Alexander. “The Timeless Way of Building”. New York: Oxford University Press, 1979.

[3] E. Gamma, R. Helm, R. Johnson and J. Vlissides.” Design Patterns”. Addison-Wesley, 1995.

[4] S. Ceri, P. Fraternali, M. Matera: "WebML Application Frameworks: a Conceptual Tool for Enhancing Design Reuse". WWW10 Workshop Web Engineering, Hong Kong, May 2001.

[5] Tidwell, J. (1999), “Common Ground: A Pattern Language for Human-Computer Interface Design”, Available at: www.mit.edu/~jtidwell

[6] G. Fischer and E. Scharff. „Meta-design: Design for designers”. In Proceedings of the DIS2000 Conference, 2000.

[7] Dennis Quan, David Huynh, and David R. Karger. “Haystack: A Platform for Authoring End User Semantic Web Applications”, in ISWC 2003.

[8] J. Borchers, “Interaction design patterns: twelve theses”, position paper CHI 2000, April, Hague, Netherlands: ACM Press.

[9] van Welie, Martijn , “Web Design patterns (2003)”, available a http://www.welie.com/patterns/

[10] Sari A. Laakso ,User Interface Design Patterns (2003), available at: www.cs.helsinki.fi/u/salaakso

[11] S. Ceri, P. Fraternali, A. Bongio: "Web Modeling Language (WebML): a Modeling Language for Designing Web Sites". WWW9 Conference, Amsterdam, May 2000.

[12] Peter J. Denning and Pamela A. Dargan, “A discipline of software architecture”, ACM Interactions Vol. 1, 55-65, ACM Press 1994.

[13] J. Finlay, E. Allgar, A. Dearden, B. McManus. “Pattern Languages in Participatory Design”, HCI 2002, London, UK. 2nd - 6th September 2002.

[14] Jan O. Borchers: “A Pattern Approach to Interaction Design”. 369-378: Proceedings of the Conference on Designing Interactive Systems: Processes, Practices, Methods, Techniques, Aug., 2000, NY ACM Press

[15] A. R. Puerta. “A model-based interface development environment”. In IEEE Software, volume 14 of 4, 1997.

[16] A. R. Puerta, Eriksson, Gennari, and Musen. “Model-based automated generation of user interfaces”. In Readings in Intelligent User Interfaces, San Francisco, 1998. ACM Press.

[17] J. Foley, C. Gibbs, W. Kim, and S. Kovacevic. “A knowledgebased user interface management system”. In Readings in Intelligent User Interfaces, San Francisco, 1998. ACM Press.

[18] J. Eisenstein, J. Vanderdonckt, and A. Puerta “Applying Model-Based Techniques to the Development of UIs for Mobile Computers” Readings in Intelligent User Interfaces. ACM Press, 2001.

[19] M. Dubinko, L. Klotz, R. Merrick, and T. V. Raman. “XForms 1.0”, W3C Recommendation 14 October 2003, http://www.w3.org/

[20] S. Ceri, P. Fraternali, M. Matera: "Conceptual Modeling of Data-Intensive Web Applications". IEEE Internet Computing, Vol. 6 , No. 4, 2002.

[21] Michael Fuchs, Claudia Niederée, Matthias Hemmje, Erich J. Neuhold “Supporting Model-based Construction of Semantic-enabled Web Applications”, In Proceedings of WISE 2003. IEEE Computer Society. Roma, Italy, December 2003.

[22] C. Niederée, C. Muscogiuri, and M. Hemmje. “Taxonomies in operation, design, and meta-design”. In Proceedings of DASWIS 2002. IEEE CS, 2002.