Model-Driven Architecture for Web Applications

Mohamed Taleb[1], Ahmed Seffah[2] and Alain Abran[3]

[1] Human-Centered Software Engineering Group, Concordia University
Montreal, Quebec, Canada
Telephone: +1 514 848 2424 ext. 7165
mtaleb@encs.concordia.ca

[2] Human-Centered Software Engineering Group, Concordia University
Montreal, Quebec, Canada
Telephone: +1 514 848 2424 ext. 3024
seffah@encs.concordia.ca

[3] Software Engineering Department & Information Technology, École de Technologie Supérieure (ÉTS),
Montreal, Quebec, Canada
Telephone: +1 514 848 2424 ext. 3024
seffah@encs.concordia.ca

Abstract. A number of Web design problems continue to arise, such as: (1) decoupling the various aspects of Web applications (for example, business logic, the user interface, navigation and information architecture), and (2) isolating platform specifics from the concerns common to all Web applications. In the context of a proposal for a model-driven architecture for Web applications, this paper identifies an extensive list of models aimed at providing a pool of proven solutions to these problems. The models span several levels of abstraction such as business, task, dialog, presentation and layout. The proposed architecture show how several individual models can be combined at different levels of abstraction into heterogeneous structures which can be used as building blocks in the development of Web applications.

Deleted: ;

Deleted: will

Keywords: Models, Model-Driven architecture, Software engineering, Web applications, MDA, architecture

1. Introduction

The concept of the model has become a major paradigm in software engineering, and its use represents a significant projection?? in terms of level of abstraction, endurance??, generality and the capacity to evolve, among others. New issues are arising, however,

Formatted: Highlight

Formatted: Highlight

related to the rebuilding of software engineering activities around this new approach (model-driven architecture). These issues are associated with the more traditional issues based on the concepts of the model and meta-model themselves, as well as on their construction and validation (model architecture) (Is this what you mean? – ch).

The Internet and its languages offer major opportunities for developing new-generation architectures for Web software systems, the latest of which are highly interactive and platform-independent, and run on the client Web browser across a network. This paper is aimed at providing a pool of proven solutions to many recurring Web design problems. Examples of such problems include: (1) decoupling the various aspects of Web applications such as business logic, the user interface, navigation and information architecture; and (2) isolating platform-specific issues from the concerns common to all Web applications.

In this paper, software architecture is defined as in [1]: "*the structure of the subsystems and components of a software system and the relationships between them typically represented in different views to show the relevant functional and non functional properties*." This definition introduces both the main architectural elements (for instance, subsystems, components and connectors), and covers the ways in which to represent them, including both functional and non-functional requirements, by means of a set of views.

In order to address these problems, a pool of proven solutions is proposed here in the form of an architecture and the related models for a model-driven architecture for Web applications. These individual models can be combined at different levels of abstraction into heterogeneous structures which can then be used as building blocks in the development of these applications.

This paper is organized as follows: section 2 introduces related work on model-oriented architectures in general, such as the model driven architecture (MDA); section 3 primarily describes the MDA proposed here, as well as some models which we have identified and formalized; finally, section 4 presents a summary and directions for future work.

2. Related Work

The concept of the model plays a central role in the majority of scientific disciplines. In some of them, it is considered to represent the "confluence" of the sciences. Models have always been used in the computer sciences, but for a long time they were relegated to the background relative to the source code, which still plays a dominant role in industry. There is a growing tendency towards a reversal of these roles, however, and the concept of the model is moving from "contemplative" mode (interpreted by humans) to "productive" mode (interpreted by processors). Moreover, whereas the models have, until recently, been used only in the design phase, and lately in the development phase, they must from now on be embedded in the software itself to allow it to evolve dynamically. The use of systematic, dynamic and well-equipped?? models is leading to the consideration of model-driven architecture as a major paradigm in software engineering.

Formatted: Highlight

Deleted: of

2.1. MDA model

Models are commonly used to flexibly represent complex systems. They can be viewed at many levels of abstraction, and complementary model views can be combined to give a more intelligible and more accurate view of a system than a single model alone. Meservy and Fensternacher [2] claim that many software development experts have long advocated using models to help in understanding the problem that a system seeks to address; yet development teams commonly employ models only in the early stages of design. Often, once construction begins, teams leave models behind and never update them to reflect their changing designs of the software.

Deleted: modeling

Deleted: conceptions

Deleted: project

Most software developers would agree that modeling should play a role in every project [2]. However, there is no clear consensus on what that role should be, how developers should integrate modeling with other development activities, and who should participate in the modeling process [2].

In 2001, the Object Management Group introduced the MDA initiative as an approach to system specification and interoperability based on the use of formal models (i.e. definite and formalized models) [3; 4; 5; 6; 7]. The main idea behind MDA is to represent business logic in the form of abstract models. These models are then mapped (partly automatically) to different platforms according to a set of transformation rules. The models are usually described by UML in a formalized manner which can be used as input for tools to perform the transformation process.

The main benefit of MDA is that it clearly separates the fundamental logic behind a set of specifications from the characteristics of the particular middleware that implements it. In other words, the MDA approach distinguishes between the specifications of the operation of a system from the details of the way that the system uses the capabilities of its platform. This architectural separation of concerns constitutes the foundation of MDA, and was created with a view to achieving three main goals: portability, interoperability and reusability [8, 9, 10].

The MDA approach comprises three main steps:

- Specification of the system independently of the platform that supports it
- Specification of target platforms
- Transformation of system specifications into the specifications for a particular platform

In short, MDA makes a sharp distinction between models of:

- the business model (the Computation-Independent Model, or CIM), sometimes called a domain model,
- the business model in a specific technology context (PIM), and
- a model that is tied to the business and uses platform-specific code (PSM).

There are two others steps which can be integrated into MDA process development:

- **Capture of requirements in a CIM**. The Computation-Independent Model captures the domain without reference to a particular system implementation or technology; the CIM would remain the same even if the systems were implemented mechanically, rather than in computer software, for example;

- **Deployment of the system in a specific environment**. The issue here is the deployment of the system in several specific platforms and environments.

2.2. Compositional Structured Component Model

The authors have previously presented the CSCM model [11], which is designed to allow the construction of software components with variable lists of functionalities selected according to components' composition descriptor instances at runtime. The capability offered by CSCM components to select the required functionalities tackles the issue of an excessive number of unwanted functionalities. Furthermore, CSCM can significantly ease and simplify software maintenance, modification and reuse.

According the authors, the power of CSCM components [11] can be efficiently accessed in the development of software application families. Such families are most likely to reuse coarse- to large-grained software components across families of applications with different functional configuration and capabilities.

Comment [L1]: The relevance of this section is not clear to the readers. More context-rationale is required

3. The Proposed Architecture

3.1. Overview

To tackle some of the weaknesses identified in related work, set of concepts,? we propose a 5-tier architecture of model-driven generic classification schema for a Web software architecture.

3.2. Model taxonomy

A taxonomy of models is proposed below. As well, examples of models are presented to illustrate the need to combine several types of models to provide solutions to complex problems at the five architectural levels. This list is not exhaustive, as there is no doubt that more models are needed and that others have yet to be discovered.

Comment [L2]: In the sentence just before in 3.1, you talk about a 5-tier arcthitecure, and then you switch to a 'taxomony'. However, this is not really a 'taxomony'. Most probably you should drop taxonomy throughout the text, and stick with the 5-tier architecture. Also, in the abstract, you talk about an architecture, and not about a taxonomy.

Comment [L3]: This contradicts the Abstract which says that the list is exhaustive.!!

A number of Web models have already been suggested; for example, the OMG's Model-Driven Architecture [3, 4, 5, 6, 7], Si Alhir's Understanding the Model Driven Architecture (MDA) [8], Methods & Tools [8], Paternò's Model-Based Design and Evaluation of Interactive Applications [9], Vanderdonckt's Task Modelling in Multiple Contexts of Use [10], Msheik's Compositional Structured Component Model: Handling

Selective Functional Composition [11] and Puerta's Modeling Tasks with Mechanisms [12].

In our work, we investigate how these existing collections of models can be used as building blocks in the context of the proposed five-layer architecture. Which models solve which problems and at what level is the question we try to answer.

Comment [L4]: The structure of this paragraph is unusual. It is essential to the understanding of the paper? If not, it can be deleted.

An informal survey conducted in 2004 by the HSCE Research Group at Concordia University identified at least five types of Web model that can be used to create a model-oriented Web software architecture.

Comment [L5]: This is a weak rational – approach. I would suggest to delete this. It is not essential.

The following are examples of the models in our taxonomy:???

Formatted: Highlight

3.3. Domain model

The domain model, sometimes, called a business model, encapsulates the important entities of an application domain along with their attributes, methods and relationships [13]. Within the scope of user interface development, it defines the objects and functionalities accessed by the user via the interface. Such a model is generally developed using information collected at the business and functional requirements consideration stage. It defines the list of data and features or operations to be performed in a different manner, i.e. by different users on different platforms. The first model-based approaches used a domain model to drive the user interface at runtime. These domain models describe the application in general, and include some specific information for the user interface. For example, the domain model [13] includes:

- a class hierarchy of the objects in the application,
- the properties of the objects,
- the actions that can be performed on the objects,
- units of information (parameters) required by the actions, and
- pre- and post-conditions for the actions.

In their basic form, domain models should represent the important entities along with their attributes, methods and relationships. This kind of a domain model corresponds to the object model of recent object-oriented software development methodologiess.

Consequently, the way to integrate user interface and system development is with the simultaneous use of the data model. This is why recent model-based approaches include a domain model known from software engineering methodologies. Four other models are then derived from this model: the task, dialog, presentation and layout models.

Deleted: only real

Comment [L6]: In the subsequent section, you say the opposite: that the domain model is derived from the other ones!!!

Is 'derived' the correct term?

3.4. Task model

With the task model, the way in which activities can be performed to reach the user's goals for an interactive system [9] can be described. Designers can use them to develop integrated descriptions of the system from a functional and interactive point of view. They

Deleted: is

typically decompose, hierarchically, tasks and subtasks into atomic actions [10]. In addition, the relationships between tasks are described in correlation with the execution order or dependencies between peer tasks. Tasks may contain attributes about their? importance, duration of execution and frequency of use.

For our purposes, we must revisit the following definition:

Deleted: can rethink

> A *task* is a goal which, along with the ordered set of tasks and actions, would satisfy that goal in the appropriate context [13].

Deleted: .

This definition explains the intertwined nature of tasks and goals: actions are required to satisfy goals. Furthermore, the definition allows tasks to be decomposed into sub-tasks, and there exists some ordering among the sub-tasks and actions. In order to complete this definition we need to define also goal, action and artifact:

> A *goal* is an intention to maintain or change the state of an artifact (based on [13]).
>
> An *action* is any act that has the effect of maintaining or changing the state of an artifact (based on [13]).
>
> An *artifact* is an object that is essential to a task. Without this object, the task cannot be performed. Moreover, the state of this artifact is usually changed in the course of performance of the task. Artifacts are real things that exist in the context of task performance, e.g. the business. Artifacts are modeled as objects and represented in the business model. This implies a close relationship between the task model and the business model.

With these definitions, we can derive the information necessary to represent them in a task model. According to [13], one task description includes:

- a goal,
- a non-empty set of actions or other tasks that are necessary to achieve the goal,
- a plan of how to select actions or tasks, and
- a model of an artifact, which is influenced by the task.

Consequently, the development of the task model and the development of the domain model are interrelated. One of the goals of model-based approaches is to support user-centered interface design. Therefore, they must enable the user interface designer to create the various task models. One other model is then derived from this model: the domain model.

Formatted: Highlight

3.5. Dialog model

This model makes it possible to provide dialog styles to perform tasks and to provide proven techniques for conducting the dialog. The dialog model defines the navigational structure of the user interface. It is a more specific model and can be derived in large part from the more abstract task, user and business object models.

A dialog model is used to describe the human-computer conversation. It specifies when the end-user can invoke commands, functions and interaction media, when the end-user can select or specify inputs, and when the computer can query the end-user and present information [14]. In other words, the dialog model describes the sequencing of input tokens, output tokens and the way in which they are interleaved. It describes the syntactical structure of human-computer interaction. The input and output tokens are lexical elements. Therefore, this model specifies, in particular, the user commands, interaction techniques, interface responses and command sequences permitted by the interface during user sessions. Two other models are then derived from this model: the domain and task models.

Formatted: Highlight

3.2.6. Presentation model

The presentation model describes the visual appearance of the user interface [13]. This model exists at two levels of abstraction: abstract and concrete. In fact, it defines the appearance and the form of presentation of the application on the Web page. This model provides ways in which the contents or the related services can be visually organized onto working surfaces, the effective layout of multiple information spaces and the relationship between them. It defines the physical and logical layout suitable for specific Web pages, such as home pages, lists and tables.

A presentation model describes the constructs that can appear on an end-user's display, their layout characteristics and the visual dependencies among them. The displays of most applications consist of a static part and a dynamic part. The static part includes the presentation of the standard "widgets" like buttons, menus and list boxes. Typically, the static part remains fixed during the runtime of the interactive system, except for state changes like enable/disable, visible/invisible. The dynamic part displays application-dependent data which typically change during runtime (e.g. the application generates output information; the end-user constructs application-specific data).

The former provides an abstract view of a generic interface, which represents a corresponding task and dialog models. Three other models are then derived from this model: the domain, task and dialog models.

Formatted: Highlight

3.2.7. Layout model

The layout model is realized as a concrete instance of an interface. It consists of a series of user interface components which define the visual layout of the user interface and the detailed dialogs for a specific platform and its context of use. There may be many concrete instances of a layout model that can be derived from the presentation and dialog models.

The layout model makes it possible to provide conceptual models and architectures for organizing the underlying content across multiple pages, servers, databases and computers. This model is concerned with the "Look & Feel" of Web applications and with the construction of a general drawing area (e.g. Canvas widget), and all the output inside a canvas must be programmed using a general-purpose programming language and a low-level graphical library. Four other models are then derived from this model: the domain, task, dialog and presentation models.

Formatted: Highlight

4. Summary and Future Work

In this paper, we have identified and proposed five categories of models, and provided examples of them, for a model-driven architecture for Web applications to resolve many recurring Web design problems, examples of which include: (1) decoupling the various aspects of Web applications such business logic, user interface, navigation and information architecture; and (2) isolating platform-specific problems from the concerns common to all Web applications. Our discussion has focused on the way to specify a model-driven architecture using particular models such as the domain, task, dialog, presentation and layout models.

Future work will require the classification of each model and the illustration of each of them in UML class and sequence diagrams. Next, some transformation rules will have to be defined, as will some of the relationships between the models, so that they can be combined to define the new methodology for software development based on the resulting pattern categories that we have defined and formalized in our previous work and on the various models proposed and defined in this paper.

Comment [L7]: It will not be clear to the readers what patenrs has to do here

5. References

1. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: A System of Patterns: Pattern-Oriented Software Architecture. West Sussex, England, John Wiley & Sons (1996)
2. Thomas O. Meservy, Kurt D. Fensternacher.: 'Transforming Software Development: An MDA Road Map,' IEEE Computer, Vol. 38, No. 8, pp. 52-58 (2005)
3. An ORMSC White Paper, ormsc/05-04-01.: 'A Proposal for an MDA Foundation Model,' V00-02, OMG Group, [Online] available at: http://www.omg.org/docs/ormsc/05-04-01.pdf (2005)
4. Desmond Dsouza, Kinetium: 'Model-Driven Architecture and Integration Opportunities and Challenges,' OMG Group, [Online] available at: ftp://ftp.omg.org/pub/docs/ab/01-03-02.pdf (2001)
5. Richard Soley and the OMG Staff Strategy Group.: 'Model-Driven Architecture,' OMG Group, [Online] available at: ftp://ftp.omg.org/pub/docs/omg/00-11-05.pdf (2000)
6. Dr. Jishnu Mukerji.: Document number ormsc/2001-07-01, Architecture Board, ORMSC, 'Model Driven Architecture (MDA) – Technical Perspective,' OMG Group, [Online] available at: http://www.omg.org/docs/omg/01-07-01.pdf (2001)
7. J. Miller, J. Mukerji.: 'MDA Guide Version 1.0.1,' OMG doc.omg/2003-06-01, [Online] available at: http://www.omg.org/docs/omg/03-06-01.pdf (2003)

8. Sinan Si Alhir.: 'Understanding the Model Driven Architecture (MDA),' Methods & Tools, Vol. 11, No.3, pp. 17-24, [Online] available at: **http://www.methodsandtools.com/archive/archive.php?id=5** OR http://home.comcast.net/~salhir/UnderstandingTheMDA.PDF (2003)
9. F. Paternò.: '*Model-Based Design and Evaluation of Interactive Applications,'* Springer (2000)
10. Vanderdonckt, J., Q. Limbourg and N. Souchon.: 'Task Modelling in Multiple Contexts of Use,' In *Proceedings of DSV-IS 2002*, 2002, Rostock, Germany, pp. 77-95 (2002)
11 Hamdan Msheik, Alain Abran, Eric Lefebvre.: 'Compositional Structured Component Model: Handling Selective Functional Composition', IEEE 30th EUROMICRO Conference, pp. 74-81 (2004)
12. Puerta A.R., Tu S.W., Musen M.A.:'Modeling Tasks with Mechanisms.' International Journal of Intelligent Systems, Vol. 8 (1993)
13. Egbert Schlungbaum.: 'Model-based User Interface Software Tools Current state of Declarative Models,' Technical Report 96-30, Graphics, Visualization and Usability Center, Georgia Institute of Technology, CADUI'96 workshop in Namur, Belgium (1996)
14. Puerta A.R.: 'Model-Based Interface Development Environment', IEEE Software 14, 41-47 (1997)