Steering Model-Driven Development of Enterprise Information System through Responsibilities

Ming-Jen Huang and Takuya Katayama

School of Information Science, Japan Advanced Institute of Science and Technology
1-1 Asahidai, Nomi-shi, Ishikawa, Japan

Abstract. OMG proposes the MDA that promotes the ideas of modeling in UML and transforming UML models to code. But UML is not universal for every domain and the direct translation approach of the MDA is not adequate. In this paper, we introduce REST, an idea of using responsibilities as contextual information to instruct machines to generate software systems. First, we give an overview of RESTDA - a software development architecture for business based on the concept of REST. Then we describe a domain-specific language - Business Models. It helps developers to describe a business from a document-processing perspective. We also introduce a rule-based validation of consistency within Business Models. Finally, we describe the transformation mechanism of RESTDA. Our approach provides machines higher intelligence to generate source code for different contexts.

1 Introduction

Model Driven Architecture (MDA) proposed by OMG [1] is a software development approach that promotes defining platform-independent models in UML and having machines to transform them into technology-specific code [2]. Its concept is based on two assumptions. First, UML is precise and expressive enough to describe problems we are interested in. It is also universal enough to describe any domain of problems. Second, all problems defined by every kind of UML modeling constructs should imply identical contextual information. For the MDA, UML becomes the master key to open every door to any solution.

With regard to the first assumption, different domains have different requirements. Thus, a domain-specific language (DSL) that is customized for a specific domain is more realistic and more productive [3]. With regard to the second assumption, considering the following example: does the case of implementing UML models of a car having four wheels equal to the case of a teacher having four students? By UML, they may be drawn identically in class diagrams or even sequential diagrams. For an effective model transformation mechanism, we do not only have to give machines syntactic and semantic information, but also the capability of reacting according to different contexts. To that end, we propose a conceptual idea, *Responsibility-Steering Model Transformation* (REST) to augment existing model-driven approaches.

Huang M. and Katayama T. (2005).
Steering Model-Driven Development of Enterprise Information System through Responsibilities.
In *Proceedings of the Joint Workshop on Web Services and Model-Driven Enterprise Information Systems*, pages 165-170
DOI: 10.5220/0002566201650170

REST is a conceptual idea of model transformation that is inspired by Responsibility-Driven Design, which is proposed by Wirfs-Brock [4]. She promoted the idea of designing a software system from responsibilities and devising role objects to collaboratively work together to assume these responsibilities [5]. In REST, we consider responsibilities of an abstraction level are *realized* by responsibilities of a level beneath. And realization of all responsibilities of all levels, combining with domain-specific languages, instructs machines to generate detailed implementation of different technology-specific code. The advantages of our approach are: (1) Responsibilities provide extra contextual information of domains under consideration. The problems like the example of car and teacher can be avoided. (2) By defining model transformation in terms of responsibilities of different levels, any change of requirements can lead to easy and reliable modification to the target system. (3) By formalizing responsibilities, the correct transformation can be ensured.

The purpose of our work is to devise a development architecture and to apply the idea of REST to the development architecture to solve the problems of the MDA mentioned above. In this paper, we introduce the development architecture for business called *Responsibility-Steering Development Architecture* (RESTDA).

The remainder of this paper is organized as follows. Section 2 gives the overview of RESTDA, the details of the DSL - Business Models, and the description of the rule-based consistent validation of BM. Section 3 describes the details of REST and its implementation in RESTDA. Section 4 gives the conclusions and future works.

2 Responsibility-Steering Development Architecture

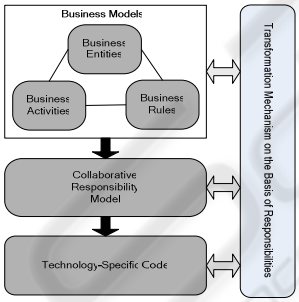

Fig. 1. Architecture of RESTDA

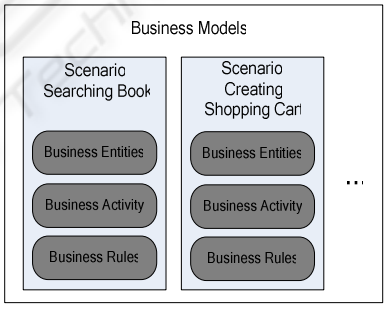

Fig. 2. Structure of Business Models

RESTDA is composed of a DSL to model concepts of business world and a model transformation mechanism between models and code. Fig. 1 shows the architecture of RESTDA. The DSL, *Business Models*, describes different business scenarios from the structural, behavioral, and constraint aspect. Definition of BM of a target system is transformed into a technology-neutral object model - *Collaborative Responsibility Model* (CRM) by machines with a business scenario as a unit. CRM does not contain details of technology-specific implementation but generalized software objects and responsibilities of these objects. By means of CRM, a system can be divided into many vertical-sliced parts, and each part can be transformed into different

technology-specific code that is most suitable to a situation. Instead of direct translation of meta-model to code, RESTDA applies the idea of REST, using syntactic, semantic, and contextual information, to instruct machines to transform BM to CRM and CRM to source code.

2.1 Business Models Description

BM defines the running of a business from three different views. *Business entities* describe the structural view. *Business activities* describe the behavioral view. *Business rules* describe constraints of business entities and business activities. Definition of BM of a target system has one or more scenarios which describe a possible situation of document processing (see Fig. 2).

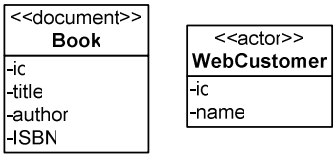

Fig. 3. A Sample of Business Entity Diagram

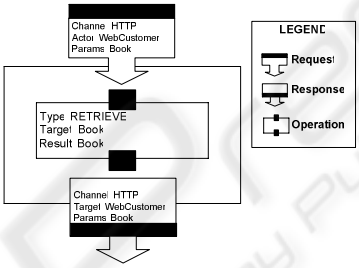

Fig. 4. A Sample of Business Activity Diagram

Business entities are roles that participate in a scenario. They are described in business entity diagram. A sample is shown in Fig. 3. Here, we borrow the drawing conventions from UML. There are two types of business entity, actor and document. Actor type entity represents human role in a scenario. In the diagram, it is displayed in stereotype `<<actor>>`. Document type entity is what is usually printed out as a formal or legal document in a business. In the diagram, it is displayed in stereotype `<<document>>`. Between business entities, they may have relations.

A business activity is a sequence of operations on which business documents are processed. A business activity has three parts, *request*, *operations*, and *response*. Request describes how the request is sent (Channel), who makes the request (Actor), and what information is carried by the request (Params). A single operation is an action operating on a document. It describes what type the operation is (Type), what document to operate on (Target), and what information to provide after completion of an operation (Result). There are four types of operation: CREATE, RETRIEVE, UPDATE, and DELETE. Operations can be linked sequentially to represent sequential operations. A business activity is described in a business activity diagram. As the exemplar Fig. 4 shown, the request is sent via `HTTP` and made by `WebCustomer`. `WebCustomer` should provide information of `Book` in the request. The business activity has a single operation to `RETRIEVE` information of `Book` and return resulting `Book`. The response is sent via `HTTP` to the `WebCustomer`. `WebCustomer` and `Book` are referred to the business entities of the scenario.

Business rules are constraints of business entities and business activities. For a business entity, business rules define the possible range of values of its properties. For a business activity, they define conditions of allowable activity requests or conditions of allowable operations, among other things.

2.2 Formalization and Implementation of Verification

The semantics of BM is formalized as predicates and implemented in a rule-based engine to verify validity of BM. These predicates are called *verification rules*. They are defined in terms of three basic constructs, *be* of business entity, *attr* of entity attribute, and *ba* of business activity. The types of business entity and activity are $DocumentType(be)$, $ActorType(be)$, and $BusinessActivityType(ba)$. Each construct has an identifier $ID(be)$, $ID(attr)$, and $ID(ba)$. Relations (*own* and *detailedBy*) between business entities are $Own(be_1,be_2)$ and $DetailedBy(be_1,be_2)$. Channel, actor, params of request are $RequestChannel(ba)$, $RequestActor(ba)$, and $RequestParams(ba)$. Type, target, and result of operation are $OperationType(n,ba)$, $OperationTarget(n,ba)$, and $OperationResult(n,ba)$ respectively (n denotes the sequence of operations). For example, $OperationType(1,ba)$ denotes the type of the first operation. Channel, actor, params of response are $ResponseChannel(ba)$, $ResponseTarget(ba)$, and $ResponseParams(ba)$ respectively.

There are five types of verification rules within BM. In this paper, we explain only the first type of verification rules - structural relation. In BM, business entities have two types of relation, own and detailedBy. For example, an actor type `Manager` owns a document type `MonthlySalesReport` and `MonthlySalesReport` is detailed by a document type `WeeklySalesReport`. Types of entity at two ends of a relation should be correct and they are represented as two rules:

1. Only an actor type entity can own a document type entity

$$\forall be_1,be_2\ Own(be_1,be_2) \Rightarrow ActorType(be_1) \wedge DocumentType(be_2)$$

2. Only a document type entity can be detailed by a document type entity:

$$\forall be_1,be_2\ DetailedBy(be_1,be_2) \Rightarrow DocumentType(be_1) \wedge DocumentType(be_2)$$

The verification rules are implemented in a rule-based engine, Jess [6]. Jess contains facts and rules. The collection of facts is information Jess knows. The collection of rules in Jess is a kind of actions that triggers under certain conditions [7]. Rules in Jess can be stated as "`if` P `then` A". P denotes a set of conditional facts. A denotes a set of actions. P is tested against all known facts. For example, if we know (1) a verification rule states that only an actor type entity can own a document type entity and (2) a fact states that `SalesStaff` (actor type) owns `PurchasingStaff` (actor type). If a Jess rule states "if (1) is not satisfied against all known facts, then displays a warning message." Since `PurchasingStaff` of (2) is not a document type, Jess would display a warning message.

3 Responsibility-Steering Model Transformation

REST is a conceptual idea of model transformation that uses responsibilities of different levels as contextual information to instruct machines to transform platform-independent models into technology-specific code. Real-world responsibilities of structural and behavioral constructs and constraints of a DSL are *realized* by generalized object responsibilities and the generalized object responsibilities are realized by responsibilities of technology-specific code, such as classes or components. In RESTDA, the realization of generalized object responsibilities is pre-defined. Developers only have to define (1) the responsibilities of BM and (2) how generalized object responsibilities realize these responsibilities for each scenario.

First, developers have to define real-world responsibilities from BM. A responsibility of any level always has a *holder* and a *receiver*. A holder represents a structural role which assumes the responsibility. A receiver represents a structural role that is affected by the consequence of the responsibility. Responsibilities of the same level are connected by holders and receivers. We use *Collaborative Responsibility Diagram* (CRD) to draw responsibilities, holders, and receiver as shown in Fig. 5. A collaborative responsibility diagram shows the structural and behavioral aspect of responsibility realization. To read the diagram, a rounded rectangle represents a responsibility and a rectangle represents a role. The left-hand role of a responsibility represents a holder and the right-hand role represents a receiver. A receiver of a responsibility could be a holder of another responsibility. The responsibilities are fulfilled from left to right one by one.

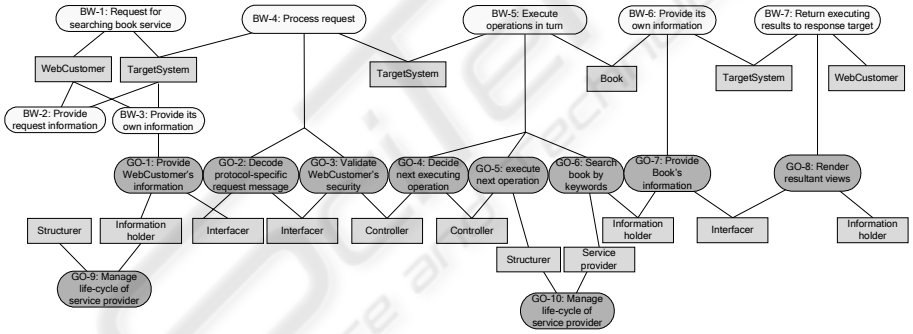

Fig. 5. A Sample of Collaborative Responsibility Diagram

Second, developers have to define how generalized object responsibilities realize the real-world responsibilities. It is a process of refinement by decomposing a real-world responsibility into smaller chunks. For example, the real-world responsibility "Process request" is realized by two generalized object responsibilities: "Decode protocol-specific message" and "Validate WebCustomer's security". Again, a holder and a receiver are assigned to a generalized object responsibility. They come from generalized objects. We borrow the concepts of role stereotypes from Responsibility-Driven Design. It defines six types of role: information holder, structurer, service provider, coordinator, controller, and interfacer [5]. A generalized software object represents a stereotype that assumes a set of generalized responsibilities. Developers have to contemplate types of responsibility and types of generalized object

simultaneously for each scenario. Responsibilities of generalized objects and their holders and receivers form CRM that are further transformed into Java code by Jess.

RESTDA predefines how a generalized object of CRM is transformed into one or more Java classes. The generation rules are also implemented in Jess in a code-template-generation fashion where the data for placeholders of code templates come from definition of CRM. These rules also define how different source code to generate for different responsibility definitions.

4 Conclusion and Future Work

In this paper, we introduced the software development architecture for business – RESTDA which is based on the idea of REST. The significance of the research is that domain experts can use BM to describe the running of a business without concerning any technology details. Instead of direct translation approach, the combination of syntactic, semantic, and contextual information of each level offers machines higher intelligence to generate software systems from platform-independent models.

With regard to future work, one is to formalize the concept of responsibilities Another is to use much expressive higher-order logic to quantify over predicates and to apply automatic theorem provers, such as HOL, to verify consistency of BM and responsibility realization [8,9].

Acknowledgments

This research is conducted as a program for the "21st Century COE Program" by Ministry of Education, Culture, Sports, Science and Technology.

References

1. MDA Guide Version 1.0.1. OMG. http://www.omg.org/docs/omg/03-06-01.pdf (2003)
2. Frankle, D.S.: Model Driven Architecture : Applying MDA to Enterprise Computing. Wiley, New York (2003)
3. Thomas, D.: MDA: Revenge of the Modelers or UML Utopia? IEEE Software, Vol. 21, No. 3, pp. 15 – 17 (2004)
4. Wirfs-Brock, R.: Object-Oriented Design: a Responsibility-Driven Approach. OOPSLA '89 Conference Proceedings, pp. 71 – 75 (1989)
5. Wirfs-Brock, R., McKean, A.: Object Design: Roles, Responsibilities, and Collaborations. Addison-Wesley, Boston (2003)
6. Jess v7.0a4. http://herzberg.ca.sandia.gov/jess/
7. Friedman-Hill, E.: Jess in Action. Manning: rule-based systems in Java. Manning, Greenwich, CT (2003)
8. Aoki, T., Katayama, T.: Unification and Consistency Verification of Object-Oriented Analysis Models. Asia-Pacific Software Engineering Conference, (1998)
9. Yatake, K., Aoki, T., Katayama, T.: Collaboration-Based Cerification of Object-Oriented Models in HOL. Verification and Validation of Enterprise Information Systems (2004)