An XML Definition Language to Support Use Case-Based Requirements Engineering

M. Golbaz, A. Hasheminasab, and N. Daneshpour

***Abstract*—The focus of this paper is to introduce a theory based on a wide-spreading model of Use Cases (UCs). The pervasive benefits of UCs in the computer systems and software's developments have motivated the UCs formalization needing. Also, the definition of UCs through Use Case Description Markup Language (UCDML) is enabled the XML technology explotation in order to suggest the powerful ways for the creation, usage, distribution, and maintenance of UCs. Consequently, we present the UCDML formalization, an XML definition language, to support the UC-based requirements engineering.**

***Index Terms*— Requirements engineering, UCDML, UCs, XML technology.**

I. Introduction

Use Case (UC) analysis is one of the first and primary means of requirements gathering in the behavioral methodology, and UCs are standard techniques for collection of requirements in many modern software methodologies developments. According to the Unified Modeling Language (UML) semantics, a UC can be described in plain text, using operations, in activity diagrams, by a state-machine, or by other behavior description techniques, such as Pre- and Post-conditions. The interaction between the UC and the Actors can also be presented in collaboration diagrams [1]; however, no format is presented for any of these alternatives. Most users find it suitable to follow the Jacobson's original style [2] of natural language descriptions inserted in a table, but the problem is that the original proposal was imperfect. UCs have a formal and well-defined structure in the theory of software engineering, and are commonly expressed as hard-to-read text documents, containing a mix of natural language statements, semi-standard names and expressions, and raw cross references.

Although the XML technologies benefits based on the UCs and UCDML descriptions are very questionable, the followings are the reasons of UCDML software developers presented for the UCs descriptions [3]:

Manuscript received November 11, 2007.

M. Golbaz is with the Department of Computer Engineering, College of Electrical Engineering, Shahid Rajaee University, Tehran, Iran. (Phone: 0098-21-22970062; fax: 0098-21-22970033; e-mail: mgolbaz@sru.ac.ir).

A. Hasheminasab is with the Department of Computer Engineering, College of Electrical Engineering, Shahid Rajaee University, Tehran, Iran. (e-mail: alireza.hn@sru.ac.ir).

N. Daneshpour is with the Department of Computer Engineering, College of Electrical Engineering, Shahid Rajaee University, Tehran, Iran. (e-mail: daneshpour@aut.ac.ir).

- XML is exceedingly related to the Web. Its meaning is that by simply using XML-enabled web browsers and web servers, the UC descriptions may be simply published and edited in distributed systems. So, our approach in contradiction of most UC-manipulation tools is not platform-dependent.
- The UCs can be easily converted into different formats by using the XSLT technology. Many free software tools support XSLT and their output can have a generic structured format. UCs can be transformed to be compatible with the file formats of existing applications as well.
- It is without difficulty possible to produce a detailed textual documentation directly from the structural description of the UC by the XSLT technology. Thus, we have already created a first transformation style-sheet to present the UCDML descriptions in a standard tabular format which is frequently employed in business documents.

Accordingly, we present the UCDML formalization, an XML-based approach to the definition of UCs. This paper outlines a syntax and informal semantics for UC template, and we propose an XML definition language to support UC-based requirements engineering.

II. Related Works

In the last few years, lots of researches have been done in the field of UC-based requirements engineering, and a variety of techniques, models and notations have been developed quickly.

Du Bois *et al*. [4] present the specification language Albert II. This formal language is based on ontology of concepts used for capturing requirements of real-time, distributed systems. Being as expressive as pioneering RE languages [5], the language is reported to be natural: it offers a declarative description of mapping between the informal statements provided to the customers onto formal statements expressed in the language. Then, Cunning and Rosenblit [6] present a semi-formal method to structure the behavioral requirements for real-time embedded systems. This method is based on a set of forms called Structured Requirements Specifications (SRSs) that contain both informal text-based descriptions and formally defined language constructs.

Dulac *et al.* [7] confirm that the interactive use of computer-supported, visual representations of requirements helps create, navigate, review, and understand formal specifications. There are also research efforts in visual requirement representation that focus on different aspects of the requirements engineering practice. One such example [8] investigates multimedia technology for eliciting requirements of general software systems, promoting reusable components that include not only code and documents, but also voice narration, animation sequences, and message mechanisms.

UCs are often used to describe requirements for systems to be designed and implemented in the object-oriented paradigm [9]. However, there has been much debate as to where the UC is the most effective approach. Jacobson [2], later an important contributor to the Unified Modeling Language (UML) and Rational Unified Process (RUP), see the UC as useful for requirements, specification and design, then codified the visual modeling technique for specifying UCs. Originally he used the term Usage Case, but found that neither of these terms sounded natural in English, and eventually he settled on the term UC. Since he originated UC modeling, Cockburn [10] made contributions to the subject, arguably the most influential, comprehensive, and coherent next step in defining what UCs are (or should be) and how to write them.

Many other authors have suggested using guidelines for UC descriptions [10]–[13] and such guidance is often entirely plausible. Graham [14] suggests structure for task events, and Alexander and Stevens [15] suggest that requirements statements should also be similarly straightforward. Adolph *et al.* [16] present patterns for UC descriptions that provide a high level view of the UC structure, and some advice on how to construct individual sentences. Other authors have posited similar guidance on writing descriptions, such as recommending the use of strong verbs and nouns [17] and avoiding passive voice [18]. Others again suggest structural guidance. For example, Fowler and Scott [19] and the OMG [20] recommend writing primitive steps and numbering events.

The UML community takes a similar viewpoint, for example, Booch *et al.* [21], Jackson [22], [23], and Kovitz [24] see UCs as a means for describing a specification, because UCs deal with interactions between a user (actor) and the machine (system). However, Rosenberg [25], [26] has a slightly different take on the matter. He sees UCs as ways of describing “units of behavior”, requirements as describing the “laws that govern that behavior” and functions as “the individual actions that occur within that behavior” (p. 123). In the following we present an obvious statement of content or format of the UC text.

Works on natural language analysis of UCs include [27], [28]. They mainly focus on providing guidelines and restricted languages for writing UCs in natural language whereas avoiding ambiguities and errors. In [27] Rolland and Ben Achour present an approach for guiding UCs development. They propose linguistic patterns and structures for UC specification as well as an iterative process for writing UC specification as an unambiguous natural language text. A similar approach is discussed in [29] where a restricted language based on a set of guidelines is defined for UCs. Fantechi *et al.* [28] use linguistic techniques to analyze UCs expressed in natural language. They collect quality metrics and detect defects related to UCs inherent ambiguity. Martin Glinz [30] proposes a natural language based notation for UCs and a manual approach for state-charts synthesis.

In the Potts' *et al.* approach [31], [32] scenarios are in textual form following some tabular notations. The requirements engineering process is supported by a hyper-text tool in which scenarios and requirements are annotated with requirements discussions, rationales and change requests. Therefore, whereas inspecting a requirement or a scenario fragment, the user can retrieve, through hypertext links, the open questions, responses and arguments that have been posed on this element and the change requests referring to it as well.

Cockburn [33] offers a step-by-step description given in natural language. UCs written in Cockburn's style shows an apparent sequence of actions and separate normal cases from exceptional ones. Nevertheless, nonlinear flow of actions (alternatives, iterations) is not systematically treated and there is still no comprehensible separation between user actions and system responses. We split the notion of interaction given by Cockburn into a sequence of well distinguished atomic steps. The new atomic notion of interaction, gained by separating user actions from system responses, has been more suitable for consecutive automatic manipulation of UCs.

Duran *et al.* [34], [35] use XML to signify software requirements and XSLT to support the requirements verification in order to guarantee some quality properties. However, this approach does not focus on the formal structuring of requirements through XML. In fact, it is mostly oriented towards the implementation, and the authors propose the REM tool which is a visual environment where the user can write the specifications and automatically perform some verification on them.

A number of XML languages for UCs exist, notably OMG’s interchange language for all parts of UML [36]. In contrast, UCDML is not designed to match a particular notation but is designed to embody the aspects of UCs that people use and interpret consistently, with the goal of supporting automation based on this consistency. Nentwich *et al.*’s xlinkit is a general-purpose framework for expressing and checking consistency constraints in XML documents [37].

Thomas A. Alspaugh [38] describes an approach for identifying aspects of scenarios(and UCs containing them) that people use consistently, structuring them, and using this structure to support work with scenarios, and this approach is implemented by an XML language and a Java package for it. Also, Della Penna *et al*. [3] define a new XML-based language, called SDML (Scenario Description Markup Language), to support the Scenario-based requirements engineering. In the following we give a brief synthesis of some techniques for representing and using UCs which have been developed so far.

III. Use Case Model Terminology

The UML defines UCs drawing syntax as well as the associations of UCs and Actors, but it does not include a clear statement of content or format of the UC text. In this section, we describe the different parts of UCs which are consisting of following parts:

Identifier. Each UC should have an integer number in addition to a unique name suggesting its purpose. The name should express what happens when the UC is performed.
Type. Each UC has three types namely *Primary*, *Secondary*, and *Optional*.
Description. Each UC should have a Description describing the role, purpose, outcome, or a high-level description of the sequence of actions.
Overview. The Overview provides a quick summary of a UC processing documentation.
Level. The Level specifies the goal level of the UC distinguished by the *Summary*, *Sub-function*, or *User levels*.
Scope. This indicates the system or subsystem witch the UC refers to, and contains two *Strategic* and *System* types.
Actor. An Actor is a person or external entity that interacts with the system and performs UCs to accomplish tasks. Actors can be illustrated as an *External*, *Primary*, or a *Secondary*.
Stakeholders and Interests. Stakeholders and Interests are various entities may not directly interact with the UC system but they may have an interest in the outcome of the UC.
Status. This shows the UC work in progress, ready for review, passed review, or failed review status.
Categorization. Categorization provides a way of UCs breakings into the manageable collections.
Trigger. The Trigger describes the UC initiation causes event such as an Actor, another UC, or a temporal event.
Pre-conditions. Before the initiation of a UC, Pre-conditions define the expected state of the system.
Basic Course of Actions. This is the interaction between Actors and the System. Each step should have a *Step Number*, an *Actor Action* and a *System Response*.
Post-conditions. Post-conditions define the final state of the system after a successful completion of the Basic Course of Actions. For each Post-condition, the followings including *Success End Condition*, *Failure End Condition* and *Minimal Guarantee* should be documented.
Alternative Courses of Action. Alternative Courses of Action define additional paths which may be triggered the Basic Course of Actions; consequently, an *Alternative Name*, *Triggers*, *Course Steps* and *Post-conditions* should be documented.
Frequency of Use. Frequency of Use estimates the number of times the UC will be performed by the Actors per appropriate unit of time.
Assumptions. Each Assumption should be in a declarative manner evaluating to be true or false.
Associated UCs. Combination of *Extended* and *Included* UCs can create the Associated UCs.
Special Requirements. For the UCs needing to be addressed during design or implementation, additional requirements (e.g. nonfunctional) identified by the Special Requirements.
Issues. Issues are a list of action items related to the development of the UCs. There may also be some notes on possible implementation strategies or impact on other UCs.
General Notes. Additional information related to the UCs need to be captured whereas UC analysis is performing.

IV. THE UC DESCRIPTION MARKUP LANGUAGE

In this section, we formalize the structure of the UC model throughout a formal language called UCDML (Use Case Description Markup Language) whose syntax has been defined through an XML Schema. A UCDML document begins with the <UCDML> element that contains these elements:

Identifier. Each Identifier is defined by a distinct <identifier> element. As shown in Fig. 1, this element contains a <name> element giving the unique name of the UC, and a <number> element identifying the integer number of the UC.
Type. Each type is defined by a distinct <type> element. Fig. 2 illustrates the UC element type attributes (i.e. primary or secondary or optional).
Description. This is defined by a distinct <description> element giving a textual description of the UC (Fig. 2).
Description. This is defined by a distinct <description> element giving a textual description of the UC (Fig. 2).
Overview. Each overview is characterized by a distinct <overview> element providing an overview of the UC (Fig. 2).

```
<identifier>
<name>
<!--UC name--> </name>
<number>
<!--UC number--> </number>
</identifier>
```

Fig. 1. UCDML fragment for the UC Identifier.

```
<type type="primary, secondary or optional" />
<description> <!--UC description--></description>
<overview> <!--UC overview--></overview>
```

Fig. 2. UCDML fragment for the UC Type, Description and Overview.

```
<level type="summary, sub-function or user" />
<scope type="strategic or system" />
<actor type="external, internal, primary or secondary" id="..." >
<name>
<!--actor name--> </name>
<description>
<!--actor description--> </description>
</actor>
```

Fig. 3. UCDML fragment for the UC Level, Scope and Actor.

```
<stakeholder id="..." >
<name>
<!--stakeholder name--> </name>
<description>
<!--stakeholder description--> </description>
</stakeholder>
```

Fig. 4. UCDML fragment for the UC Stakeholder, Status and Categorization.

Levels. As it is seen in Fig. 3, each level is marked by a distinct <level> element specified the type attribute (i.e. summary, sub-function or user).

Scope. Fig. 3 shows the type attribute (i.e. strategic or system) defined by a distinct <scope>.

Actor. Each Actor is defined by a distinct <actor> element. This element has two attributes specifying the unique identifier of the Actor and the Actor type (external, internal, primary or secondary), respectively. As shown in Fig. 3, the <actor> element contains a <name> element giving the name of the Actor and a <description> element contains a textual description of the Actor.

Stakeholder. Each Stakeholder is defined by a distinct <stakeholder> element which has an attribute specifying the unique identifier of the Stakeholder. The <stakeholder> element in Fig. 4 includes a <name> element giving the name of the Stakeholder and a <description> element contains a textual description of the Stakeholder.

Status. Each Status (i.e. work in progress, ready for review, passed review or failed review) is defined by a distinct <statue> element (Fig. 4).

Categorization. Categorization is defined by a distinct <categorization> element (Fig. 4) which has a type attribute specifying the business importance, level of abstraction, UC type, functional area and etc.

Trigger. Definition of each Trigger is by a distinct <trigger> element distinguishing the Trigger type (i.e. Actor, another UC or a temporal event). According to the Fig. 5, the <trigger> element contains an <actor> as well as a <useCase> element including the unique identifier of the Actor, the Actor name, the unique identifier of the UC and the UC name, respectively. Also, it involves an <event> element with textual temporal event description.

Pre-conditions. Each one is defined by a distinct <precondition> element commenting on the Pre-conditions of a UC (Fig. 5).

Basic Course of Actions. Each Basic Course of Action is defined by a distinct <basicCourseofAction> element which has an id character specifying the unique identification. This element describes the control flow through a sequence of <step> element with an id attribute presenting the position in the control flow, and contains the <interaction> element pointing the basic step of the control flow as well. As it is behold in Fig. 6, each <interaction> has an <actor>element which shows the unique identifier of the Actor and the Actor name; moreover, it has an <action> element describes the happened Actor actions in each step, and a <systemResponse> element recognized the system name.

```
<trigger type="actor, another UC or a temporal event" >
<actor id="..." name="..." />
<use-case id="..." name="..." />
<event>
<!--event description--> </event>
</trigger>
<precondition>
<!--UC precondition--></precondition>
```

Fig. 5. UCDML fragment for the UC Trigger and Pre-condition.

```
<basicCourseofAction id="..." >
<step id="..." >
<interaction>
<actor id="..." name="..." />
<action>
<!--actorAction--> </action>
<systemResponse name="..." >
<!--systemResponse--> </systemResponse>
</interaction>
</step>
</basicCourseofAction>
```

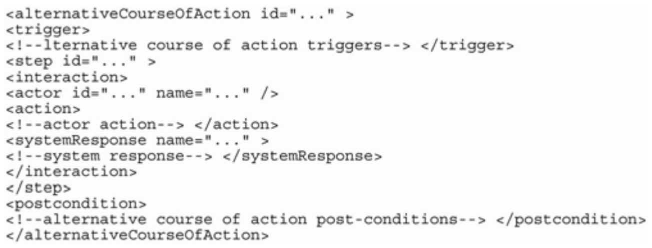
Fig. 6. UCDML fragment for the UC Basic Course of Action.

Post-conditions. Explanation of a Post-condition is by a <postcondition> element. This element is comprised of a <successEndCondition> element with the successful end condition description or a <failureEndCondition> element with the failure end condition description, and also a <minimalGuarantee> element with the guarantee or assurance of the UC depiction (Fig. 7).

Alternative Courses of Action. This is characterized in a <alternativeCourseOfAction> element which has two featuring the special identifier and the name of the Alternative Course of Action. This element has a <trigger> element describing the execution causes of the Alternative Course of Action, a <step> element describing the alternative control flow, and a <postcondition> element specifying The post-conditions of an Alternative Course of Action. Each <step> has an id attribute declaring the position in the alternative control flow and contains the <interaction> element through a basic step of the alternative control flow description. As revealed in Fig. 8, each <interaction> contains an <actor> element with unique identifier of the Actor and the Actor name attributes. In addition, it has an <action> element which describes the Actor actions in the step, and a <systemResponse> element with the name of the system feature.

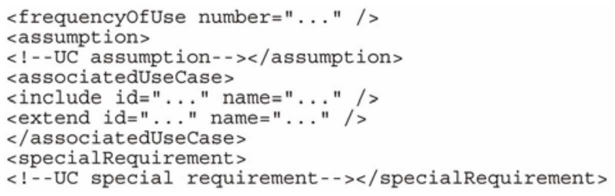
```
<postcondition>
<successEndCondition>
<!--UC successful end condition--> </successEndCondition>
<failureEndCondition>
<!--UC failure end condition--> </failureEndCondition>
<minimalGuarantee>
<!--UC minimal guarantee--> </minimalGuarantee>
</postcondition>
```

Fig. 7. UCDML fragment for the UC Post-condition.



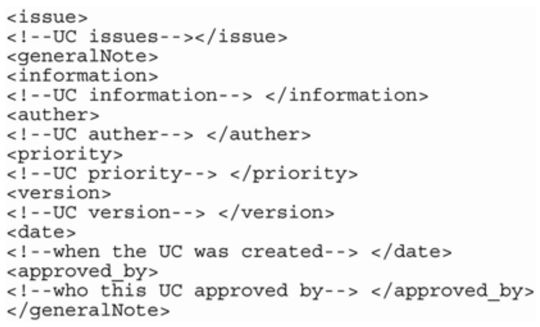
```
<alternativeCourseOfAction id="..." >
<trigger>
<!--lternative course of action triggers--> </trigger>
<step id="..." >
<interaction>
<actor id="..." name="..." />
<action>
<!--actor action--> </action>
<systemResponse name="..." >
<!--system response--> </systemResponse>
</interaction>
</step>
<postcondition>
<!--alternative course of action post-conditions--> </postcondition>
</alternativeCourseOfAction>
```

Fig. 8. UCDML fragment for the UC Alternative Course of Action.



```
<frequencyOfUse number="..." />
<assumption>
<!--UC assumption--></assumption>
<associatedUseCase>
<include id="..." name="..." />
<extend id="..." name="..." />
</associatedUseCase>
<specialRequirement>
<!--UC special requirement--></specialRequirement>
```

Fig. 9. UCDML fragment for the UC Frequency of Use, Assumption, Associated Use Case, Special Requirement.



```
<issue>
<!--UC issues--></issue>
<generalNote>
<information>
<!--UC information--> </information>
<auther>
<!--UC auther--> </auther>
<priority>
<!--UC priority--> </priority>
<version>
<!--UC version--> </version>
<date>
<!--when the UC was created--> </date>
<approved_by>
<!--who this UC approved by--> </approved_by>
</generalNote>
```

Fig. 10. UCDML fragment for the UC Issue, General Note.

Frequency of Use. A frequency of use is defined by a distinct <frequencyOfUse> element. As shown in Fig. 9, this element has a number attribute identifying the number of times the UC will be executed.

Assumptions. As it is demonstrated in Fig. 9, definition of each assumption is by a separate <assumption> element.

Associated Use Cases. Each Associated Use Case is defined by a different <associatedUseCase> element containing an <extend> and <include> elements which have two attributes specifying the unique identifier and name of the extended or included UC (Fig. 9).

Special Requirements. Identification of any additional requirement of the UC is determined via a <specialRequirement> element (Fig. 9).

Issues. Issue is defined by a distinct <issue> element which concludes a list of UC issues (Fig. 10).

General Notes. A <generalNote> element holds an <information> element with any additional UC information descriptions, an <author> element identifying the name of a person who initially documented the UC, a <priority> element supplied the priority of the UC, a <version> element with the UC version specification, a <date> element indicating when UC version was created, and an <approved_by> element designating the name of a person who the UC approved by (Fig. 10).

V. Conclusion

In this paper, we proposed an UML compatible template utilized for the documentation of UCs; furthermore, the simplicity usage of the presented template should be acceptable. Moreover, we show the UCDML formalization of UCs descriptions. The description of UCs through UCDML can be easily published and edited in distributed systems, and converted into various formats using the XSLT technology.

References

[1] *UML Semantics version 1.1*, Rational Software Corporation, September 1997. Available: http://www.rational.com/uml/index.shtml.

[2] I. Jacobson, M. Christerson, P. Jonsson, and G. Oevergaard, *Object-Oriented Software Engineering: a Use Case Driven Approach.* Addison-Wesley Longman Publishing Co., 1992.

[3] G. Della Penna, B. Intrigila, A.R. Laurenzi, and S. Orefice, "An XML definition language to support scenario-based requirements engineering," *International Journal of Software Engineering and Knowledge Engineering* 13 (3), 2003, pp. 237–256.

[4] P. Du Bois, E. Dubois, and J.-M. Zeippen, "On the use of a formal RElanguage: The generalized railroad crossing problem," *Heitmeyer and Mylopoulos* [18].

[5] S.J. Greenspan, A. Borgida, and J. Mylopoulos, "A requirements modeling language, Inform," *The Journal of Systems and Software* 11 (1), 1986, pp. 9–23.

[6] S.J. Cunning, J.W. Rozenblit, "Test scenario generation from a structured requirements specification," *IEEE Conf. Wksp Eng. Comput.-Based Syst. (CWECS)*, 1998.

[7] N. Dulac, T. Viguier, N. Leveson, and M.A. Storey, "On the use of visualization in formal requirements specification," *Dubois and Pohl* [11], pp. 71–80.

[8] D. J. Chen, W. C. Chen, and K.M. Kavi, "Visual requirements representation," *The Journal of Systems and Software* 61, 2002, pp. 129–143.

[9] J. Arlow, "Use cases, UMLvisual modelling and the trivialisation of business requirements," *Requirements Engineering Journal* [3], 1998, pp. 150–152.

[10] A. Cockburn, *Writing Effective Use Cases.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[11] C. Ben Achour, C. Rolland, N. Maiden, and C. Souveyet, "Guiding use case authoring: Results of an empirical study," *In 4th IEEE international symposium on requirements engineering*, RE'99, Limerick, Ireland, June 1999, pp. 36–43.

[12] Cox, K, *Heuristics for use case descriptions. PhD Thesis*, Bournemouth University, UK, 2002.

[13] C. Rolland, C. Ben Achour, "Guiding the construction of textual use case specifications," *Data and Knowledge Engineering Journal*, 25(1–2), 1998, pp. 125–160.

[14] I. Graham, *Requirements engineering and rapid development*. Harlow: Addison-Wesley, 1998.

[15] I. Alexander, R. Stevens, *Writing better requirements*. Harlow: Addison-Wesley, 2002.

[16] S. Adolph, P. Bramble, A. Cockburn, and A. Pols. *Patterns for Effective Use Cases.* Addison Wesley, 2003.

[17] D. Kulak, E. Guiney, *Use Cases – Requirements in Context.* Harlow, Addison-Wesley, 2000.

[18] R. Pooley, P. Stevens, *Using UML—Software Engineering with Objects and Components*. Harlow, Addison-Wesley, 1999.

[19] M. Fowler, K. Scott, *UML Distilled 2nd Edition.* Harlow, Addison-Wesley.

[20] Object Management Group (OMG), *Unified Modeling Language v1.4 – Semantics.* Document 01-09-73. Version taken from: http://www.omg.org/pub/docs/formal/01-09-73.pdf, taken Jan 2002.

[21] M. Jackson, *Problem frames*. Harlow: Addison-Wesley, 2001.

[22] G. Booch, I. Jacobson, and J. Rumbaugh. *The Unified Modeling Language – Users Guide.* Addison-Wesley Longman Publishing Co., Inc. Reading, MA, 1999.

[23] M. Jackson, "A discipline of description," *Requirements Engineering Journal* [3], 1998, pp. 73–78.

[24] B. Kovitz, *Practical Software Requirements: A Manual of Content and Style*. Manning Publications, Greenwich, CT, 1999.

[25] D. Rosenberg, K. Scott, *Use Case Driven Object Modeling with UML: A Practical Approach*. Harlow, Addison-Wesley, 1999.

[26] D. Rosenberg, K. Scott, *Applying Use Case Driven Object Modeling with UML.* An Annotated E-Commerce Example. Reading, MA, Addison-Wesley, 2001.

[27] A. I. Anton, W. M. McCracken, and C. Potts, “Goal decomposition and scenario analysis in business process reengineering,” in *Proc. 6th Conference on Advanced Information Systems Engineering,* Utrecht, The Netherlands, 1994, pp. 94-104.

[28] C. Potts, K. Takahashi, J. Smith, and K. Ora, “An evaluation of inquiry based requirements analysis for an internet service,” in *Proc. Second IEEE Symp. Requirements Eng.,* IEEE Computer Society, 1995, pp. 27-34.

[29] C. Rolland, C.B. Achour, “Guiding the construction of textual use case specifications,” *Data Knowl. Eng. J.* 25 (1–2), 1998, pp. 125–160.

[30] A. Fantechi, S. Gnesi, G. Lami, and A. Maccari, “Application of linguistic techniques for use case analysis,” in *RE’02, Proceedings of the 10th Requirements Engineering Conference*, 2002.

[31] K. Bo¨ttger, R. Schwitter, D. Richards, O. Aguilera, and D. Molla´, “Reconciling use cases via controlled languages and graphical models,” in *INAP 2001, Proceedings of the 14th International Conference on Applications of Prolog*, 2001, pp. 186–195.

[32] Glinz, “Improving the quality of requirements with scenarios,” in *Proceedings of the Second World Congress on Software Quality*, 2000, pp. 55–60.

[33] A. Cockburn, “Using goal-based use cases,” *Journal of Object-Oriented Programming* 10(7), 1997, pp. 56-62.

[34] A. Duran, B. Bernardez, A. Ruiz, and T. Toro, “An XML-based approach for the automatic verification of software requirements specifications,” in *Proc. of the Fourth Workshop on Requirements Engineering (WER01),* 2001, pp. 181-194.

[35] A. Dura´n, A. Ruiz-Corte´s, R. Corchuelo, and M. Toro, “Supporting requirements verification using XSLT,” in *Dubois and Pohl* [11], pp. 165–172.

[36] MOF 2.0/XMI mapping specification, v2.1. Document formal/05-09-01, Object Management Group, Framingham, MA, September 2005.

[37] C. Nentwich, W. Emmerich, and A. Finkelstein, and E. Ellmer, “Flexible consistency checking,” *ACM Trans. Softw. Eng. Methodol,* 12(1), 2003, pp. 28–63.

[38] Thomas A. Alspaugh, “Relationships between scenarios,” ISR Technical Report, 2006.