IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITB12862

This paper appears in the book, *Emerging Trends and Challenges in Information Technology Management, Volume 1 and Volume 2* edited by Mehdi Khosrow-Pour © 2006, Idea Group Inc.

Reasoning about Functional and Non-Functional Concerns During Model Refinement: A Goal-Oriented and Knowledge-Based Approach

Lawrence Chung, The University of Texas at Dallas, chung@utdallas.edu

Sam Supakkul, The University of Texas at Dallas, ssupakkul@ieee.org

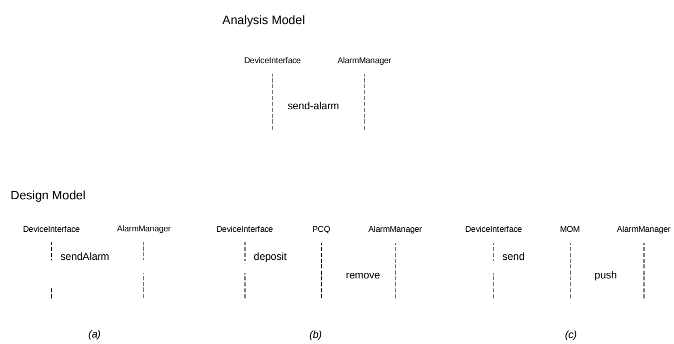
ABSTRACT

Traditional model driven development follows the stepwise refinement approach where early phase models are gradually refined with more details from one version to another and from one phase to another successively until they are expressed in the terms of the underlying programming language. Every refinement step implies some design decisions. The quality of a software system largely depends on how good, or bad, these decisions are. The quality of decisions in turn would depend on what kind of alternatives are explored, what kind of trade-offs are analyzed, and how a particular selection is made. However, the process of decision making is carried out only informally, where the knowledge and rationale that led to the decision are not explicitly documented. This makes it difficult for others to understand why certain decisions were made and to reuse the knowledge. This paper presents a goal-oriented and knowledge-based approach for explicitly representing, organizing, and reusing software development knowledge. In this framework, non-functional characteristics, such as performance and security, are treated as (soft) goals to be achieved and act as the criteria for selecting the alternatives. The application of this framework is illustrated using the refinement of a UML sequence diagram message.

I. INTRODUCTION

Traditional model driven development such as in UML-based [8] development follows the stepwise refinement approach. Every refinement step implies some design decisions [14]. The quality of a software system largely depends on how good, or bad, these decisions are. For example, Fig. 1 shows three alternatives for refining the "send-alarm" message on a UML sequence diagram from analysis to the design level. Option *(a)*, DeviceInterface makes a synchronous method invocation call and is blocked until the AlarmManager is done handling the message. Option *(b)* uses Producer-Consumer-Queue (PCQ) pattern [21] to deposit the new alarm into a synchronized buffer to be picked up by AlarmManager running in a separate thread/process, and option *(c)* uses Message-Oriented Middleware (MOM) [23] to asynchronously send the new alarm. Exploring and evaluating design decisions are usually carried out only informally without records of the knowledge and rationale used during the process [20]. This makes it difficult for others to understand why certain decisions were made and also to reuse the knowledge. These problems are the main focus of the design rationale research that produces a number of methods. However, these methods are generic for general design that is not tailored for software. The NFR Framework [4,5] provides a framework that is more specific and suitable for software development, especially for non-functional requirements (NFRs) modeling and architectural design. This paper adopts and extends the NFR Framework [4,5] to present a goal-oriented and knowledge-based framework for representing and organizing knowledge used for exploring design alternatives and evaluating trade-offs. We illustrate the application of the method using the refinement of the sequence diagram message shown in Fig. 1 as running examples throughout the paper.

Figure 1. Examples of alternatives for refining a message in a sequence diagram



The rest of the paper is organized as follows. Section II gives a brief overview of the NFR Framework. Section III describes the knowledge representation in design model and how to capture it as Method. Section IV describes how to organize Methods. Section V describes how the Methods are reused. Finally, Sec. VI offers some conclusion remarks.

II. OVERVIEW OF THE NFR FRAMEWORK

The NFR Framework [4,5] is a goal-oriented method for dealing with NFRs, which are represented as softgoals to be satisficed. The framework employs "goal-refinement, exploration of alternatives, and evaluation" analysis pattern. Using this pattern, first, high level goals are identified and refined using AND/OR decomposition. Then, design decisions for operationalizing the NFR softgoals are identified, refined, or further operationalized by lower level operationalizations. Last, the design decisions are evaluated based on how they contribute (positively or negatively) to the NFR softgoals. This entire process is recorded in a diagram called Softgoal Interdependency Graph (SIG). In the SIG, all softgoals are named with "Type[Topic]" nomenclature. In the case of NFR softgoal, "Type" indicates the NFR concern and "Topic" the context for the NFR. In the case of operationalizing softgoal, "Type" indicates the operationalization concept and "Topic" the context for which the solution is applicable. Finally, in the case of argumentation softgoal, "Type" indicates either FormalClaim or (informal) Claim [4] and "Topic" the corresponding argument description. Figure 2.a shows an example of the SIG. The individual pieces knowledge used to build each piece of the SIG can be captured as Methods as shown in Fig. 2.b.

Figure 2. A softgoal interdependency graph representing NFRs related concepts (a) that are captured as methods (b)

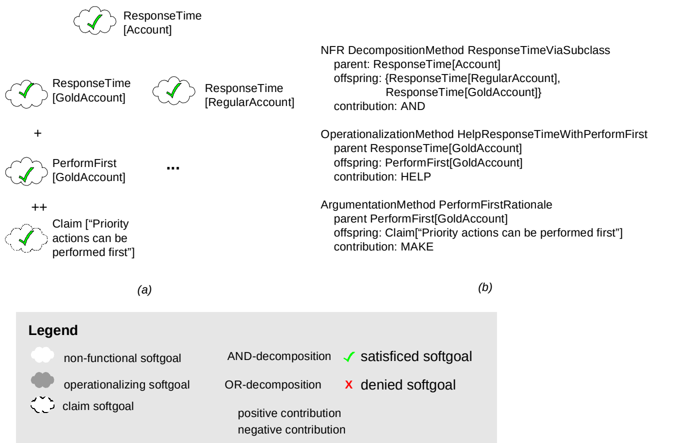
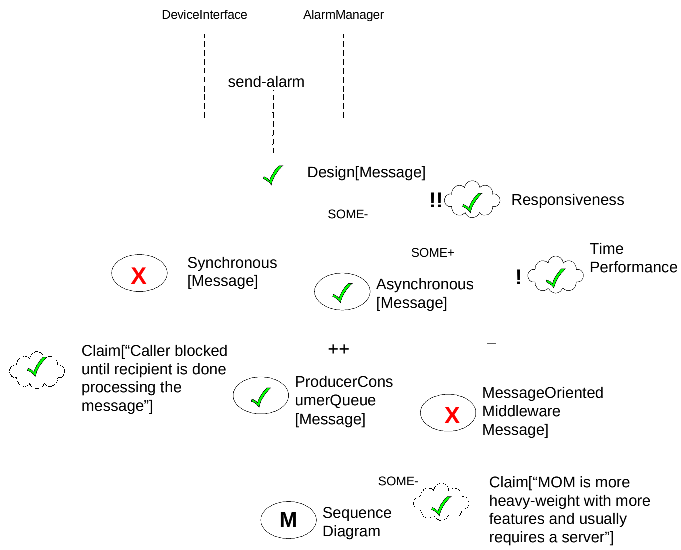


Figure 3. Representation of Functional and Non-functional Knowledge for Refining a Sequence Diagram Message



III. REPRESENTING AND CAPTURING DEVELOPMENT KNOWLEDGE

A. Representing Development Knowledge

Figure 3 shows a design decision process for refining the "send-alarm" sequence diagram message. In this paper, "goal" refers to a functional goal and "softgoal" refers to a non-functional goal. First, we identify and refine functional goals (i.e. "Design[Message]"). Second, design decisions ("Synchronous[Message]" and "Asynchronous[Message]") are identified. We repeat the refinement and operationalization of operationalizing goals until they are low-level enough for implementation. Last, the design decisions are evaluated based on their positive or negative contributions toward the highest criticality NFR softgoals (Reponsiveness).

B. Capturing Development Knowledge

We adopt and extend the Method mechanism from the NFR Framework to capture individual pieces of FRs-related knowledge with three additional types of Methods: Model Refinement, Functional Operationalization, and Model Mapping Methods. Attributes of the Methods (e.g. *parent*, *contribution*, and *applicabilityCondition*) are used as the selection criteria for selecting applicable Methods to apply. When a Method is applied against a parent goal, the goals described by the *offspring* attribute would be generated and linked to the parent goal.

Model Refinement Method

Using Fig. 3 as an example, refining the send-alarm message to design level messages is represented by the root goal "Design[Message]". An example of Model Refinement Method definition based on Fig. 3 is given below.

RefinementMethod DesignMessage

parent: UML.Message /* a UML metaclass */

offspring: Design[Message]

contribution: DesignRefinement

applicabilityCondition: /* user defined */

Functional Operationalization Method

This method captures the knowledge that creates and links an operationalizing goal to a parent functional or operationalizing goal. An example is given in the following.

FnOperationalizationMethod OperationalizeMessage_Sync

parent: Design[Message]

offspring: Synchronous[Message]

contribution: SOME+ !!Responsiveness

applicabilityCondition: /* user defined */

Model Mapping Method

Model Mapping Method captures the knowledge for mapping the parent of the root goal to a target model. An example of Model Mapping Method is given below. The *mappingMeans* attribute indicates the mechanism or technique used for the mapping. The *mappingSpec* attribute specifies the detailed mapping based on the *mappingMeans*.

MappingMethod AnalysisMessageToDesignMessage_PCQ

parent: ProducerConsumerQueue[Message]

offspring: SequenceDiagram

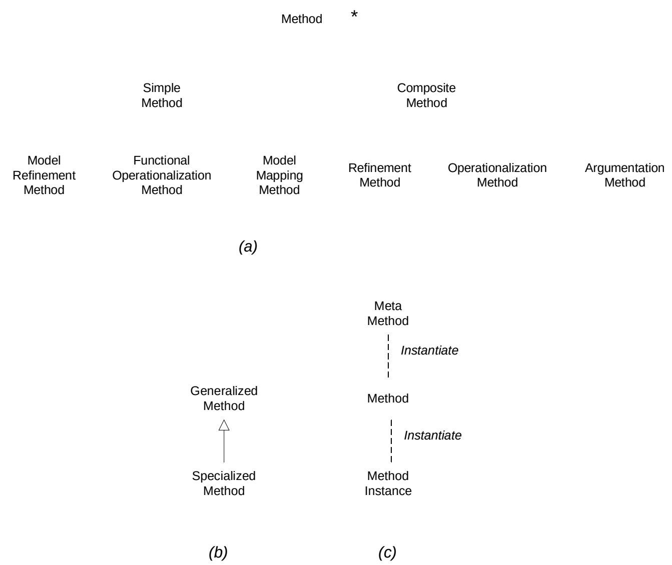
applicabilityCondition: /* user defined */

mappingMeans: TemplateMOF

mappingSpec:

```
 …
  deposit = factory.create(Message)
  deposit.sendEvent = factory.create(MessageEnd)
  deposit.receiveEvent = factory.create(MessageEnd)
  deposit.sendEvent.covered = <caller's lifeline>
  deposit.receiveEvent.covered = <recipient's lifeline>
  remove = factory.create(Message)
  remove.sendEvent = factory.create(MessageEnd)
  remove.receiveEvent = factory.create(MessageEnd)
  remove.sendEvent.covered = <caller's lifeline>
  remove.receiveEvent.covered = <recipient's lifeline>
 …
}
```

Figure 4. Methods organization along aggregation (a), generalization (b), and classification dimensions



IV. ORGANIZING DEVELOPMENT KNOWLEDGE

It is not only important that we can represent knowledge, but also how we structure and organize it [10]. This section discusses the organization of Methods along the three organizational dimensions [11].

A. Aggregation/Decomposition Dimension

In Fig. 4.a, following the composite design pattern [16], Methods may be combined to form a CompositeMethod. Because CompositeMethod is also a Method, it can be contained in other CompositeMethods. An example of the CompositeMethod definition is given below. When the OperationalizeMessage is applied, the two contained Methods are applied against the parent goal.

CompositeMethod OperationalizeMessage

parent: Design[Message]

applicabilityCondition:/* user defined */

methods: OperationalizeMessage_Synchronous, Operationalize Message_Asynchronous

B. Generalization/Specialization Dimension

Figure 4.b shows that a Method may be specialized by another Method. The specialized Method inherits all of the attributes from the generalized Method, optionally adds or re-defines one or more attributes. An example of a specialized Method is given in the following.

FnOperationalizationMethod OperationalizeMessage_PCQHurt **extends** OperationalizeMessage_PCQ

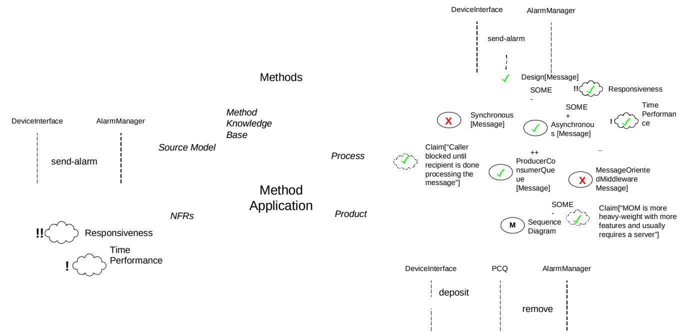
parent: Asynchronous[Message]

offspring: ProducerConsumerQueue[Message]

contribution: MAKE !TimePerformance, HURT !!Reliability

applicabilityCondition:/* specific condition */

Figure 5. Methods application that generates a goal graph and the target model



C. Classification/Instantiation Dimension

Figure 4.c shows the classification/instantiation relationship of MetaMethod, Method, and MethodInstance.

V. REUSING DEVELOPMENT KNOWLEDGE

When sufficient Methods are defined and stored in a knowledge base, they may be selected and applied successively to generate or update a goal graph to record the design decision process (i.e. Process) and also the target model elements (i.e. Product). Figure 5 depicts the Method application process.

VI. CONCLUSIONS

We have presented a goal-oriented and knowledge-based framework for representing, organizing, and reusing development knowledge. The framework extends the NFR Framework with the following extensions: 1) the “goal-refinement, exploration of alternatives, and evaluation” pattern is now made applicable to functional concerns; 2) three additional types of Methods have been proposed to capture individual pieces of FRs-related knowledge; 3) CompositeMethod is introduced to combine and reuse previously defined simple Methods and Correlation Rules. With these extensions, both functional and non-functional concerns can be analyzed together with NFRs as the criteria guiding the design decisions. Knowledge of such analysis can be captured, cataloged, tailored, improved, and reused. Future work of this research includes developing a metamodel to semi-formally describe the framework to extend the UML profile we previously defined for integrating the NFR Framework with UML [15], to also support functional goal analysis.

REFERENCES

[1] A. Dardenne, A. van Lamsweerde, and S. Fickas, “Goal-Directed Requirements Acquisition,” *Science of Computer Programming*, Vol. 20, 1993, pp. 3-50

[2] J. Mylopoulos, L. Chung, and E. Yu, “From Object-Oriented to Goal-Oriented Requirements Analysis,” *Comm. ACM*, vol. 42, no. 1, Jan. 1999, pp. 31-37

[3] J. Mylopoulos, L. Chung, S. Liao, and H. Wang, “Exploring Alternatives During Requirements Analysis,” *IEEE Software*, Jan./Feb.2001,pp. 2-6

[4] J. Mylopoulos, L. Chung, and B. Nixon, “Representing and Using Nonfunctional Requirements,” *IEEE Trans. Software Engineering*, Vol. 18, No. 6, June 1992,pp. 483-497

[5] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, Kluwer Publishing, 2000

[6] Y. Yu, J. C. S. do Prado Leite, and J. Mylopoulos, “From Goals to Aspects: Discovering Aspects from Requirements Goal Models,” In Proc. *12th IEEE Int. Requirements Engineering Conference*, 2004, pp. 38-47

[7] G. Caplat, J. Sourouille, "Considerations about Model Mapping," *Workshop in Software Model Engineering*, Oct. 2003, San Francisco, USA, http://www.metamodel.com/wisme-2003/18.pdf

[8] OMG, "UML 2.0 Superstructure Specification," http://www.omg.org/cgi-bin/apps/doc?ptc/04-10-02.zip, Oct. 2004

[9] OMG, "MDA Guide Version 1.0.1," http://www.omg.org/cgi-bin/apps/doc?omg/03-06-01.pdf, June 2003

[10] S. Greenspan, J. Mylopoulos, and A. Borgida, "Capturing More World Knowledge in the Requirements Specification," In Proc. *6th Intl. Conf. on Software Engineering*, Tokyo, Japan, 1982.

[11] R. Hull, and R. King, "Semantic database modeling: Survey, application and research issues," *ACM Comp. Surv.* Vol. 19, No. 3, 1987, pp.201-260

[12] W. Regli, X. Hu, M. Atwood, and W. Sun, "A Survey of Design Rationale Systems: Approaches, Representation, Capture, and Retrival," *Engineering with Computers*, Vol. 16, Springer-Verlag, pp.209-235

[13] K. Arnold, J. Gosling, and D. Homes, *The Java Programming Language, Third Edition*, Addison-Wesley, 2000

[14] N. Wirth, "Program Development by Stepwise Refinement," *Comm. ACM*, Vol. 14, 1971, pp.221-227

[15] S. Supakkul and L. Chung, "A UML Profile for Goal-Oriented and Use Case-Driven Representation of NFRs and FRs", In Proc. *SERA'05*, IEEE Computer Society. pp. 112-119

[16] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995

[17] OMG, "Meta Object Facility (MOF) 2.0 Core Specification," http://www.omg.org/cgi-bin/apps/doc?ptc/03-10-04.pdf, Oct. 2003

[18] E. Kavakli, "Goal-Oriented Requirements Engineering: A Unifying Framework," *Requirements Eng.*, vol. 6, no.4, 2002, pp. 237-251

[19] A.I. Anton, "Goal-based Requirements Analysis," In Proc. *2nd IEEE Intl. Conf. Requirements Engineering*, 1996, pp.136-144

[20] S. Shum, and N. Hammond, "Argumentation-Based Design Rationale: What Use at What Cost?" *International Journal of Human-Computer Studies*, Vol. 40, No. 4, 1994

[21] K. Jeffay, "The Real-Time Producer/Consumer Paradigm: A paradigm for the construction of efficient, predictable real-time systems," In Proc. *ACM/SIGAPP Symposium on Applied Computing*, Indianapolis, IN, February, 1993, pp.796-804

[22] M. Wahler, "Formalizing Relational Model Transformation Approaches", Research Plan, Swiss Federal Institute of Technology Zurich, 2004, http://www.zurich.ibm.com/~wah/doc/research_plan_wahler.pdf

[23] W. Emmerich. "Software Engineering and Middleware: A Roadmap" *The Future of Software Engineering*, ACM Press 2000

[24] S. Supakkul and L. Chung, "Representing, Organizing and Reusing Knowledge about both Functional and Non-Functional Concerns during Software Development," Submitted