Representing, Organizing and Reusing Knowledge About Functional and Non-Functional Concerns During Software Development

Sam Supakkul, Titat Software LLC, , ssupakkul@ieee.org

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

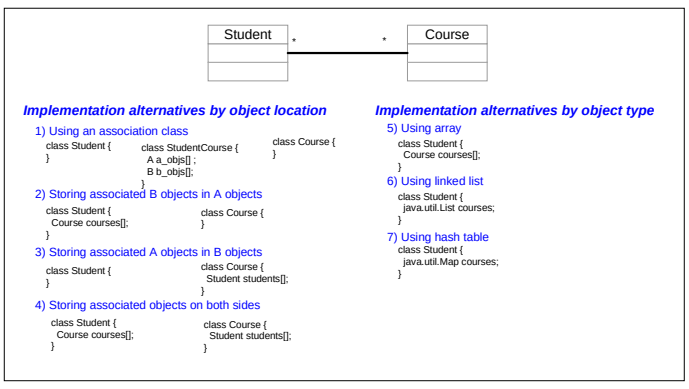
ABSTRACT

The quality of a software system largely depends on how good, or bad, the various development decisions are, during just about any phase of software lifecycle. The quality of development 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 for the software system. Usually, 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 also to reuse the knowledge in future projects in a systematic manner. This paper presents a goal-oriented methodology for explicitly representing, organizing and reusing the variety of knowledge of development alternatives, along with their trade-offs. In this methodology, non-functional characteristics, such as performance and security, are treated as (soft) goals to be achieved and act as the criteria to select among the alternatives for meeting functional concerns. The methodology is illustrated using the implementation of a class association as an example.

I. INTRODUCTION

The quality of a software system largely depends on how good, or bad, the various development decisions are, during model refinement [14] in just about any phase of software lifecycle. The quality of development 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 for the software system. For example, Fig. 1 shows alternatives for implementing a class association between Student and Course classes. A software engineer may choose option 2 to minimize memory usage (space performance) in tight memory environments such as in embedded systems, but may choose option 1 in less stringent environments for better average time performance and more flexibility in different usage scenarios.

Figure 1. Examples of Alternatives for Implementing a Class Association



However, the decision making is 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 to reuse the knowledge. These problems are the main focus of the design rationale research that produces a number of methods to capture design rationale. However, these methods are generic for general design that is not tailored for software. The NFR Framework [4,5] is an approach 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 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 approach using the class association implementation shown in Fig. 1 as running examples throughout the paper.

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 framework 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 NFRs are identified as NFR softgoals and refined using AND/OR decomposition. Then, design decisions, identified as operationalizing softgoals, for operationalizing the NFR softgoals are identified, refined, or further operationalized by lower level design decisions. 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). A claim softgoal may be defined to make an argument for or against any node or link in the SIG. In the SIG, all softgoals are named with "Type[Topic]" nomenclature. For NFR softgoals, "Type" indicates the NFR concern and "Topic" the context for the NFR. For operationalizing softgoals, "Type" indicates the operationalization concept and "Topic" the context for which the solution is applicable. Finally, for claim softgoals, "Type" indicates either FormalClaim or (informal) Claim [4] and "Topic" the corresponding argument description. Figure 2.a shows an example SIG. The

Figure 2. A Softgoal Interdependency Graph Representing NFRs Related Concepts (a) that are Captured as Methods (b)

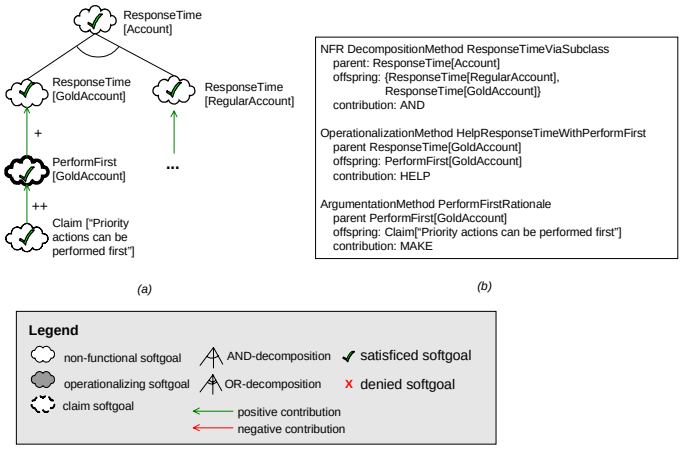
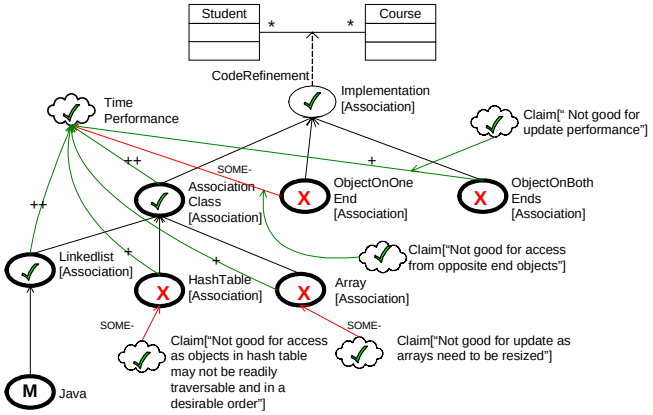


Figure 3. Representation of Functional and Non-functional Knowledge for Implementing a Class Association



individual pieces knowledge used to build each piece of the SIG can be captured as Methods as shown in Fig. 2.b.

III. REPRESENTING AND CAPTURING DEVELOPMENT KNOWLEDGE

A. Representing Development Knowledge

Figure 3 shows a design decision process for implementing a class association. 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. "Implementation[Association]"). Second, design decisions ("AssociationClass[Association]", "ObjectOnOneEnd[Association]" and "ObjectOnBothEnds[Association]") 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 (i.e. TimePerformance).

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, implementing the class association is represented by the root goal "Implementation[Association]". An example of Model Refinement Method definition based on Fig. 3 is given below.

RefinementMethod RefineClassAssociation
parent: UML.Association /* a UML metaclass */
offspring: Implementation[Association]
contribution: CodeRefinement
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 as follows.

FnOperationalizationMethodOperationalizeAssociation_AssociationClass
parent: Implementation[Association]
offspring: AssociationClass[Association]
contribution: MAKE TimePerformance, HURT !!SpacePerformance
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 LinkedListAssociationClassToJava
parent: LinkedListAssociationClass[Association]
offspring: Java
applicabilityCondition: /* user defined */
mappingMeans: template
mappingSpec:
class ${end1}_${end2}_Assoc {
Java.util.List ${end2};
Java.util.List ${end1}
}

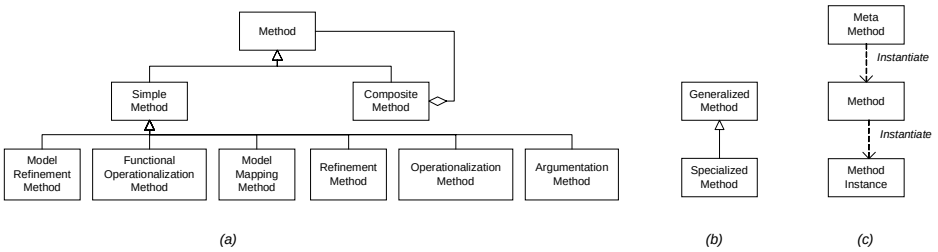
IV. ORGANIZING DEVELOPMENT KNOWLEDGE

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

Figure 4. Methods Organization along Aggregation (a), Generalization (b), and Classification Dimensions



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 AssociationClassJavaImpl

parent: Implementation[Association]

applicabilityCondition:/* user defined */

methods: OperationalizeAssociation_AssociationClass, LinkedListAssociationClass, LinkedListAssociationClassToJava

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 redefines one or more attributes. An example of a specialized Method is given below.

FnOperationalizationMethod OperationalizeAssociation_AssociationClass_Time **extends** OperationalizeAssociation_AssociationClass

parent: Implementation[Association]

offspring: AssociationClass[Association]

contribution: MAKE !TimePerformance, HURT !!SpacePerformance

applicabilityCondition:/* specific condition */

C. Classification/Instantiation Dimension

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

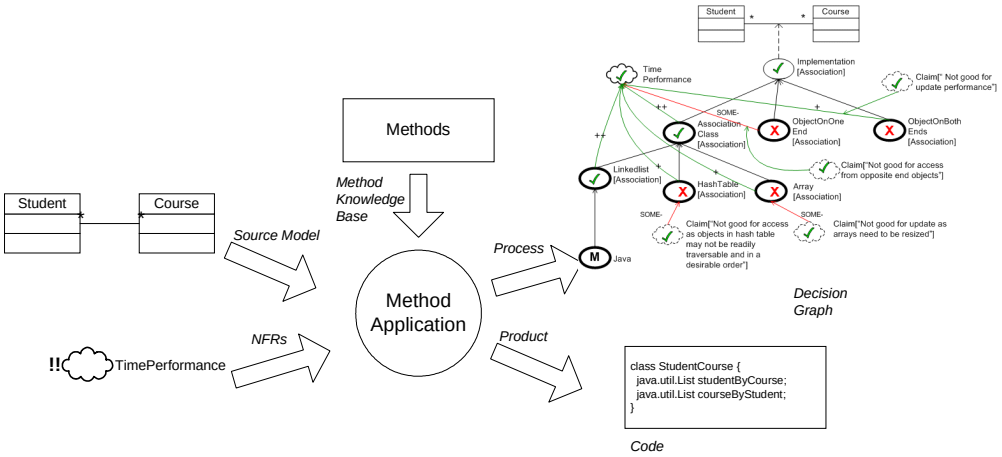
V. REUSING DEVELOPMENT KNOWLEDGE

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 refine and semi-formally describe the Methods proposed in this paper. With that, we could then extend the UML profile we previously defined for integrating the NFR Framework with UML [15], to include the functional goal analysis concepts to facilitate tool support.

Figure 5. Methods Application that Generates a Goal Graph and the Target Model



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