Aspect oriented software development using clustering GEA Techniques

M.K.Mathiyazhagaan
R.B.Vinothkumar
Asst. Professor
Sona College of Technology

Abstract:

Early aspects focus on the problem domain, representing the goals and constraints of users, customers, and other constituencies affected by a software intensive system. However, analysis of early aspects is hard because stakeholders are often vague about the concepts involved, and may use different vocabularies to express their concerns. In that the Goal modeling fits model-driven engineering (MDE) in that it captures stakeholder concerns and the interdependencies using concepts that are much less bound to the underlying implementation technology and are much closer to the problem languages. Aspect-oriented Programming (AOP) provides language constructs to facilitate the representation of multiple perceptions. Synthesis of AOP and MDE not only manages software complexity but also improves productivity, as well as model quality and longevity. In this paper, we propose a model-driven framework for tracing aspects from requirements. These aspects can be discovered during goal-oriented requirements analysis. This proposal includes a systematic process for discovering aspects from relationships between functional and nonfunctional goals.

Keywords:

Discovering Early aspects, Goal-oriented Requirements, Aspect-oriented Programming(AOP), Model-driven Engineering(MDE)

Introduction:

The early aspects can help to improve modularity in the requirements and architecture design and to detect conflicting concerns early, when trade-offs can be resolved more economically. Analyzing early aspects also enables stakeholder interests to be traced throughout the software development Aspect-oriented programming (AOP) is founded on the idea of aspect as a cross-cutting concern during software development. Aspects are usually “units of system decomposition that are not functional”, such as “no unauthorized access to data” or “efficient use of memory”. Aspects cut across different components of a software system. The Requirements goal models use goal decomposition to support the description and analysis of stakeholder intentions that underlie the required software system. Our analysis focuses on clarifying the problem domain concepts that underlie the candidate early aspects. By investigating the meanings of the terms that stakeholders use to describe these high-level goals, we can determine whether they do indeed represent concerns that crosscut requirements and design artifacts.

Aspect Oriented Software Development (AOSD) should be used from the early stages of software development such as domain analysis and requirements engineering. Identifying aspects at an early stage helps to achieve separation of concerns in the initial system analysis, instead of deferring such decisions to later stages of design and code, and thus, having to perform costly solutions. The use of models in engineering software is pervasive across different phases, from requirements and design, to verification and validation. It is the emerging paradigm of model-driven engineering (MDE) that advocates the systematic use of models as primary engineering artifacts throughout the software system.

Goal-oriented requirements engineering focuses on goals which are” roughly speaking, precursors of requirements”. Goal-based models support the description and analysis of intentions that underlie a new software system. The aspect-oriented programming is a programming methodology. However, this methodology does not deal with the origins of aspects. This framework for tracing aspects

from requirements goal models to implementation and testing. Goal-oriented requirements engineering uses goal models to elicit, specify, and analyze requirements. We provide language support for modeling goal aspects and mechanisms for transforming models to aspect-oriented programs. Test cases are derived from requirements models to guide verification and validation of aspects.

Literature survey:

Goal-directed requirements Acquisition, Most existing specification languages focus on functional requirements-that is, requirements about what the software system is expected to do. Nonfunctional requirements are most often left outside of any kind of formal treatment. Such requirements form an important part of real requirements documents; they refer to operational costs, responsibilities, interaction with the external environment, reliability, integrity, flexibility, and so forth. The limited scope of current formal specification languages results from the restricted set of built-in abstractions in terms of which the requirements must be captured.

Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, Aspects provide the mechanism that enables the source code to be structured to facilitate the representation of multiple perceptions and to alleviate tangling and scattering concerns. Many of these concerns often arise in the problem domain. Therefore, it is important to identify and represent concerns that arise during the early phases of software development, and to determine how these concerns interact. A combination of qualitative and quantitative analyses is needed to examine more quality attributes of the concept-driven framework, such as scalability, scope of applicability, relevance to functional requirements, and capability to deal with complex specifications.

Aspect-Oriented Refactoring of Legacy Applications: An Evaluation, In this Paper To highlight aspect-specific test coverage data, we measure join point coverage. Join point coverage requires executing each join point that is matched by each aspect, focusing on testing each aspect in all of the contexts where it is woven. We developed tools to gather information about coverage of advised join points. Using aspect-specific coverage information helps us check whether the aspect was woven in the right places. We cannot just rely on passing the regression tests because in general, no test suite can be guaranteed to test everything in a system.

Obvious or Not? Regulating Architectural Decisions Using Aspect-Oriented Programming,

This model tends not to be enforced on the system, leaving room for the implementors to diverge from it, thus differentiating the designed system from the actual implemented one. The essence of the problem of enforcing such models lies in their globality. The principles and guidelines conveyed by these models cannot be localized in a single module, they must be observed everywhere in the system. A mechanism for enforcement needs to have a global view of the system and to report breaches in the model at the time they occur.

Background:

This section aims to situate the existing literature on requirements engineering (RE), MDE, and AOSD. Aspect-oriented software development (AOSD) emerged from a rethinking of the relationship between modularization (partitioning software into discrete, non overlapping implementation units) and the time-honored principle of separation of concerns. Any separation-of-concerns criterion leads to a particular partitioning, as though the software were sliced into pieces in a particular direction.

Work in aspects has been mostly limited to the implementation phase, dealing with concerns that implementation units have in common, factoring those out as aspects, and employing programming-language-level support to weave the aspects back at loading time, compilation time, or runtime. In fact, AOSD has become nearly synonymous with aspect-oriented programming (AOP) and dominant decomposition usually refers to the system's decomposition into implementation units, such as subsystems, classes, and objects.

Most AOSD approaches place the burden for aspect identification and management on the programmer. But crosscutting concerns are often present well before the implementation, such as in domain models, requirements, and the architecture. Dominant decomposition, however, means something different in the early software development activities. A requirements aspect, then, is a concern that cuts across other requirement-level concerns or artifacts of the author's chosen organization. It is broadly scoped in that it's found in and has an (implicit or explicit) impact on more than one requirement artifact. Broadly scoped properties can be quality attributes (nonfunctional requirements) as well as functional concerns that the requirements engineer must describe with relation to other concerns.

The Requirements has generated a number of notations for modeling stakeholder goals and the relationships between them. Goals express, at various levels of abstraction, stakeholders' many objectives for the system under consideration. Goal-oriented RE uses goal models to elicit, elaborate, structure, specify, analyze, negotiate, document, and modify requirements. Goal modeling shifts the emphasis in requirements analysis to the actors in an organization, their goals, and the interdependencies between those goals, rather than focusing on processes and objects.

AOSD applies the principle of separation of concerns to make systems modular so that the intended software is easier to produce, maintain, and evolve. The AOSD community has recognized the importance of considering aspects early on in the software system. Aspects at the requirements level present stakeholder concerns that crosscut the problem domain. Discovering aspects early can help detect conflicting concerns early, when trade-offs can be resolved more economically.

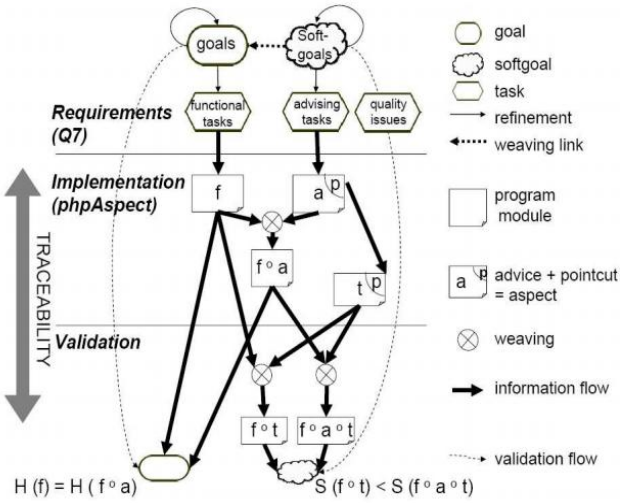

Fig: Process overview of the aspect-tracing framework

Aspect-oriented concepts are modeled explicitly in requirements at the beginning of the development process. Advising tasks, which operationalize soft goals and relate to hard goals, are modularized as aspects and weaved into the goal model to enable aspect-oriented requirements analysis? Goal modeling has become a central activity in RE. It shifts the emphasis in requirements analysis to the actors within an organization, their goals, and the interdependencies between those goals, rather than focusing on processes and objects.

Conclusion:

The initial AOP claim that it is natural to implement the globally concerns NFRs as aspects that cut across the subsystems. AOSD offers language constructs to tackle software complexity. Aspects provides the mechanism that enables the source code to be structured to facilitate the representation of multiple perceptions and to alleviate tangling and scattering concerns. Many of these concerns often arise in the problem domain. Therefore, it is important to identify and represent concerns that arise during the early phases of software development, and to determine how these concerns interact.

MDE tackles conceptual complexity in software development. The major advantage of MDE is that we express models using concepts that are much less bound to the

underlying implementation technology and are much closer to the problem languages. Goal modeling fits in the MDE picture in that it captures stakeholder intentions, beliefs, commitments, and the relationships among the various concerns. This higher level of abstraction makes the goal model easier to specify, understand, and maintain. A goal based framework that synthesizes AOSD and MDE, thereby managing complexity in both language and conceptual dimensions. A goal aspect models a system from a stakeholder-defined viewpoint.

References:

[1] L. Liu, E. Yu, and J. Mylopoulos. Security and privacy requirements analysis within a social setting. In RE 2013, pages 151–161, 2013.

[2] J. Mylopoulos, L. Chung, and B. Nixon. Representing and using nonfunctional requirements: A process-oriented approach. IEEE Transactions on Software Engineering, 18(6):483–497, June 2012.

[3] J. Mylopoulos, L. Chung, and E. Yu. From object-oriented to goal-oriented requirements analysis. Communications of the ACM, 42(1):31–37, Jan. 2011.

[4] pair Networks. Os commerce: Open Source E-Commerce Solutions, http://www.oscommerce.com.

[5] A. Rashid, P. Sawyer, A. M. D. Moreira, and J. Arajo. Early aspects: A model for aspect-oriented requirements engineering. In RE 2012, pages 199–202, 2012.

[6] C. Rolland and N. Prakash. From conceptual modelling to requirements engineering. Annals of Software Engineering, 10:151–176, 2010.

[7] G. Sousa, I., and J. Castro. Adapting the NFR framework to aspect-oriented requirement engineering. In The XVII Brazilian Symposium on Software Engineering, Manaus, Brazil, October, 2013, 2013.