MORALE Cluster Interaction Plan for

Rationale Capture and Software Understanding
and
Architecture and Generation Clusters

Georgia Institute of Technology
July 5, 1996

Background

The Mission Oriented Architectural Legacy Evolution (MORALE) project focuses on supporting evolution that is driven by a change in the purpose, or mission, of a software system. The key to our approach is exploiting the synergy between requirements analysis, architecture, and reverse engineering. Effective system evolution requires understanding both the way that an existing system accomplishes its tasks and the mission-oriented rationale for any changes that feed its evolution. Understanding the architecture of an existing system aids in predicting the impact of changes mandated by new mission-oriented requirements. Requirements analysis techniques, in turn, suggest what concepts are most useful in understanding how an existing system works and how it should evolve. The MORALE suite of tools and techniques will leverage this synergy by growing a common model of the architecture for multiple versions of a system. The common model is a basis for assessing the effects of proposed changes and the extent to which legacy code can be reused.

Scenarios Express Rationale, Focus Understanding

An important idea underlying our approach is the use of scenarios as a way of expressing design rationale. We view rationale as specifying the relation between software quality factors and design elements, and use scenarios and design abstraction categories as annotations. Scenarios are descriptions of concrete use or change situations, illustrating the kinds of activities or behaviors expected of the system and the kinds of maintenance, migration, or other evolutionary changes that may be made to the system over time. Abstraction mechanisms include those normally found in software design, such as aggregation, classification, representation, etc.

Using scenarios as a way of expressing rationale has several advantages:

- They elucidate abstract mission-oriented goals and desired properties, making it easier to understand how to operationalize the goals and map them onto architectural components.
- They are easier to elicit from a customer or analyst.
- Scenarios concretize abstract properties and quality criteria (such as reliability and portability) that are part of a system's rationale. This makes it easier to assess whether an evolving system is remaining consistent with the rationale underlying previous versions.
- They provide focus for software understanding by providing a context in which to extract behavioral descriptions of the interactions among architectural components. This helps guide the recovery of decisions, assumptions, protocol constraints, and other information related to the rationale.

Architectural Models Specify Architecture and Focus Reasoning

We view an architecture as specifying (i) the structural components of a given software system, (ii) the functional abstractions of the system and its structural components, but also (iii) the configurations of the components and the connections among them, and (iv) the internal behaviors of the system that compose the functions of the components into the functions of the system as a whole.

Using architectural models for guiding evolution has several advantages:

- Since they explicitly express the system's functions, they directly enable an analysis of new functional requirements.
- Since they explicitly specify the functions of each structural component and behaviors that compose these functions, they enable identification of the architectural modifications needed to meet the new requirements.
- For the same reason, they enable an assessment of the impact of the modifications on the system's performance and other quality factors.
- Since they explicitly use functional and behavioral abstractions, they provide a vocabulary for guiding the extraction of the architecture of a legacy system.

Products That May Be Of Interest To This Cluster:

Methodology Descriptions

- Software Architecture Analysis Method (4Q96): method for assessing impact on software quality of a set of scenarios with respect to an architecture.
- Synchronized Refinement (4Q96): method for systematic program analysis and application model construction.
- Inquiry Cycle (4Q96): support for converting a mission statement into a network of goals and for elaborating and refining these goals into operational requirements that give the desired behaviors of architectural components.
- SBF (4Q96): tutorial and reference manual for Structure/Behavior/Function system representation notation.
- MORPH: (1Q97): support for reengineering of user interfaces (UI); rule set for manual detection of UI constructs in source code.

Tools

- Event Visualizer (1Q97): graphical tool that supports browsing and analysis of execution event traces.
- GBRAT (TBD): Hierarchical goal representation editor/viewer.

Case Study

- Case Study Schema (4Q96): preconditions, structure, procedure, product definition, and evaluation metrics for case studies of legacy system evolution.
- Assessment of SBF as an architectural description language for MORALE (1Q97).