Formal Models Used for Automation in Software Development

Luqi and Valdis Berzins

Computer Science Department
U.S. Naval Postgraduate School
Monterey, CA 93943-5000

U.S. Army Research Office
P.O. Box 12211
4300 South Miami Blvd.
Research Triangle Park, NC 27709-2211

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

Approved for public release; distribution is unlimited.

This project is investigating formal models that can support automated methods supporting software development. We have focused on automation support for requirements elicitation, particularly for prototyping and the gathering of requirements remotely via the Internet; on automation support for software evolution, particularly for automatically detecting the need for software maintenance actions using non-monotonic logic, for capturing requirements dependencies and justifications using the REMAP extension of the IBIS model, for combining several modifications to a system, for coordinating parallel efforts of several designers and automating the associated configuration management tasks, and on automation support for software construction, particularly for using specifications in the design of software architectures, for automated generation of schedules for hard real-time software, and for retrieval of reusable software components.

NSN 7540-01-280-5500

19971215 084
1. Statement of the problem studied:

The objective of this research is the design of an integrated set of formal models and methods for automating a wide range of design and development tasks for real-time systems. The methods we used focus on automation of design activities that appear in an evolutionary prototyping approach to software development. This research used the state-of-the-art formal methods in software engineering to construct a cohesive set of formal models. These models were used to create and to unify automated processes for computer aided prototyping. Mathematical models for implementing a set of automated and integrated software tools were also developed. This research combines very-high-level specification abstractions and concepts with formal real-time models, automated management of software design data and human resources, transformations, change merging, and automated retrieval of reusable software components to provide automated methods for generating real-time programs and for coordinating teams of developers.

2. Summary of the most important results:

This project is investigating formal models that can support automated methods supporting software development. We have focused on automation support for requirements elicitation, particularly for prototyping and the gathering requirements remotely via the Internet; on automation support for software evolution, particularly for automatically detecting the need for software maintenance actions using non-monotonic logic, for capturing requirements dependencies and justifications using the REMAP extension of the IBIS model, for combining several modifications to a system, for coordinating parallel efforts of several designers and automating the associated configuration management tasks; and on automation support for software construction, particularly for using specifications in the design of software architectures, for automated generation of schedules for hard real-time software, and for retrieval of reusable software components.

We have explored applications of non-monotonic logic to software evolution in two different contexts: capturing design rationale and detecting context shifts via inconsistencies. In the first effort, we used extensions of the IBIS model to capture relationships between requirements issues to be resolved via prototyping, possible designed choices, and reactions to prototype demonstrations by representatives of potential user groups for a proposed system. Non-monotonic logic appears to be useful in this context because requirements are often the results of trade-offs between conflicting concerns or negotiations between user groups with different value judgments on particular issues. We also developed models and tools to support the gathering of user input remotely based on the World-wide Web technology.

We have developed a front end to Prolog that realized an answering mechanism corresponding to an extended non-monotonic logic, and have combined it with a simulator for a subset of the prototyping language PSDL. The simulator for the subset of PSDL that incorporates this answering mechanism can monitor the execution of a prototype against assumptions about the system environment. The extended answering mechanism can detect situations where the assumptions about the environment of the proposed system have changed to the point where the previous version of the requirements is no longer completely valid, and an evolution step is needed to bring the requirements back into conformance with reality.

We have developed change-merging methods for software specifications and software architectures. The software specification work treats the black-box specifications expressed in a specification language based on second order logic. The approach integrates a model of interfaces that can support merging changes to module signatures as well as changes to details of module behav-
The work on merging changes to software architectures extends our previous approach based
on program slicing for the prototyping language PSDL and an algorithm for merging changes to
PSDL programs. The previous method produced a merged design in the form of a single level
data flow diagram with annotations. This is not satisfactory for large scale applications because
the single level data flow diagrams are too complex for people to understand and use as a basis for
further design enhancements. We developed an improved method that combines the correspond-
ing changes to the design hierarchies and uses the result to reconstruct the updated hierarchical
structure of the merged design. We also provided a technique for automatically resolving struc-
tural conflicts between changes.

Our work on software reuse has resulted in a semantic method for software component
search that can simultaneously achieve high precision and high recall, a software architecture for
efficiently implementing the method. Partial matches are ranked by semantic closeness. The
method uses symbolic test cases in the form of ground equations. The software architecture is
based on multi-level filtering approach that uses database indexing and fast rough filters to cut
down the number of candidates before the more computationally expensive semantic filters are
applied.

3. Publications and technical reports:

(A) Refereed Journal Publications

(1) D. Dampier, Luqi, V. Berzins, “Automated Merging of Software Prototypes”, Journal of

(2) V. Berzins, “Software Merge: Semantics of Combining Changes to Programs”, ACM

(3) Luqi, D. Cooke, “How to Combine Nonmonotonic Logic and Rapid Prototyping to Help
Maintain Software”, International Journal of Software Engineering and Knowledge Engi-

(4) B. Ramesh, Luqi, “An Intelligent Assistant for Requirements Validation for Embedded Sys-


(8) V. Berzins, D. Dampier, “Software Merge: Combining Changes to Decompositions”, Jour-
nal of Systems Integration, special issue on Computer-Aided Prototyping (Vol. 6, No. 1-2,
1996), pp. 135-150.

(9) V. Berzins, Luqi, “Software Evolution in Prototyping”, Chinese Journal of Advanced Soft-


**B) Conference Publications:**


(12) D. Dampier, M. Kindl, R. Byrnes, Luqi, “Rapid Prototyping of Army Embedded Software


4. Scientific personnel supported by this project: Luqi, Valdis Berzins

Appendix- abstracts of papers published in 1997

A Requirements Evolution Model for Computer Aided Prototyping

This paper presents a model for requirements evolution and analysis in the context of iterative prototyping of large embedded real-time systems. This model captures user reactions to demonstrated behavior of the prototype and maps these reactions into requirements changes. The model provides the basis for automated support for requirements evolution and validation. This paper explores how a request of a change can be derived from the justifiable user responses to the demonstrated behavior of the prototype.

Gathering Requirements from Remote Users

We describe a distributed requirements engineering environment using computer aided software engineering tools linked together through the Internet. We created this distributed requirements engineering environment using Microsoft's Personal Web Server (PWS), Microsoft's Open Database Connectivity (ODBC) technology, Netscape Communicator, Microsoft's Internet Explorer, Microsoft's Access97 database, and a set of PERL scripts that are executed by users of the environment to perform database operations. We show how we added basic security features to the Internet accessible database.

Recombining Changes to Software Specifications

This paper proposes a model of software changes for supporting the evolution of software prototypes. We decompose software evolution steps into primitive substeps that correspond to monotonic specification changes. This structure is used to rearrange chronological derivation sequences into idealized conceptual derivation structures containing only meaning-extending changes, and to automatically combine different changes to a specification. A set of example illustrates the ideas.

A Logic-Based Approach to Software Maintenance

This paper provides an overview of the relationship between recent work in logic programming and recent developments in software engineering. The relationship to software engineering is more specifically concerned with how formal specifications can be used to explain and represent the basis of software maintenance and evolution.