Requirements Engineering Technologies at Rome Laboratory

William E. Rzepka, James L. Sidoran, Douglas A. White

Rome Laboratory
Griffiss AFB, NY 13441-5700

Abstract

This paper describes the program of research and development of requirements engineering technologies ongoing at Rome Laboratory since the early 1980s. One research thrust is based on a process model of requirements engineering and is implementing this model using conventional technologies. R&D activities have led to the enhancement of a requirements analysis tool and the development of two requirements validation tools based on rapid prototyping. On-going work is integrating these three tools into a requirements engineering workstation. Another research direction is taking advantage of artificial intelligence work which is developing an alternative software development paradigm called the knowledge based software assistant. These efforts are described as well as their impact on future plans for evolutionary development of the workstation environment to incorporate these knowledge-based approaches.

1: Introduction

The goals of the Rome Laboratory (RL) program in requirements engineering are to carry-on research and development of requirements engineering methods and tools, to apply the R&D results to the analysis, specification and validation of requirements for large-scale command and control systems and to assess the effectiveness of the processes prescribed by the methods and the quality of the products produced by the tools. The approach is to investigate both established techniques and develop new technologies such as executable specification languages, reusable specifications, requirements process and product metrics and rapid prototyping.

2: The Requirements Engineering Process Model

A process model (see Figure 1) is the basis for the RL program. It provides a detailed description of the fundamental activities occurring during requirements engineering, but avoids prescribing specific methods for accomplishing them. The model recognizes three basic activities being performed: eliciting requirements from the various sources (e.g., users, interfaces to other systems) of a system’s requirements, specifying them and insuring their consistency; insuring that the needs of all users taken together do not conflict and are technically and economically feasible; and validating that the requirements so derived are an accurate reflection of user needs.

3: Initial Technology Developments

RL has specifically selected three requirements engineering methods and tools for development and enhancement based on their support for the process model activities of elicitation, analysis, specification and validation utilizing prototypes. Each tool will be briefly described below.
3.1: Controlled Requirements Expression (CORE)

CORE [1] is a requirements definition and analysis method whose procedures explicitly support the notion that requirements originate from several, diverse viewpoints of how the same needed capability will be used. CORE organizes these viewpoints as a functional hierarchy. CORE procedures prescribe data flow techniques to elaborate the viewpoints. Self-consistency of each viewpoint is supported by specific checking procedures. The Analyst [2] is a Macintosh™ based tool supporting requirements elicitation and analysis using the CORE method. Capabilities include diagram construction, maintenance and consistency checking.

3.2: Rapid Prototyping System (RPS)

The RPS [3] is a collection of tools which supports the activities of building, executing and analyzing user interface and performance assessment prototypes. Prototypes developed using the RPS tools are not intended for eventual use in the target system. Instead they are meant to permit detailed examination of critical functions of computer based systems for the purpose of improving understanding of the requirements for those systems. RPS has been implemented on an Apollo/Aegis platform.

3.3: Prototyping Environment (Proto)

Proto [4] is a prototyping system consisting of a data flow based methodology, high-level, executable prototyping language and integrated tools supporting prototype development and reusable prototype components. Proto has been implemented on an Apollo/Unix platform.

The Proto methodology is based on the data flow model. The user creates a graphical representation of the process architecture of a target system. The data flow graph defines the system's processing activities and their interconnections. The design details of any processing activity may be specified at that level of the design or may be postponed to some lower level of abstraction. In specifying a processing activity's design the prototype developer can use Proto's graphical/textual data flow language, can augment this with a conventional programming language, C, or can browse a library of reusable components to locate components which are appropriate to the current process development. The reusable component is integrated into the evolving prototype using the same graphical interface being used to develop the prototype.

4: Evolving the Technologies into the 1990s

The first phase objective was to develop sufficient capabilities to permit analysis and validation of realistic requirements problems. Initial versions of the methods and tools described above were completed by the fall of 1988.

4.1: Enhancing Proto

Two additional efforts [5,6] were undertaken to enhance and extend Proto and to move it to a Sun/Unix platform utilizing the Open Look graphical interface standard. Its data manager has been replaced by a commercial object management system.

The data flow model has been extended to include process concurrency, explicit data flow protocols (e.g., stream, synchronized and sample) and performance assessment. The interpreter has been replaced with a simulator making it possible to assess the relative durations of prototype executions. The simulator also maintains records of resources consumed by the executing graph. With these enhancements the effects of distributing a software architecture among simulated processors may be studied.

Proto's distributed system design mechanisms have naturally evolved to include facilities for specifying hardware architectures in terms of the number of processors, their bus architecture and memory configuration. An allocation tool is used to assign the logical processes of the data flow graph to the processors and memories of the hardware architecture. Processor usage and memory access history data are maintained so that the relative performance merits of various software architectures allocated to different hardware configurations may be assessed. The two efforts which developed these improved Proto capabilities were completed by the summer of 1991.

4.2: Tool integration - the Requirements Engineering Workstation

The next phase of development is an integration of the three existing tools into a single unified methodology for requirements elicitation, analysis, specification and validation whose procedures are supported by a Requirements Engineering Workstation (REW). The primary integration vehicle will be the object management system. It will store all information which is not in the exclusive domains of the individual tools, thereby allowing sharing of information which is needed by one or more of the other tools.

The architecture for the REW environment will tightly integrate the RPS and Proto tools on a Sun/Unix platform while loosely coupling the Macintosh™ based CORE/Analyst through a file transfer facility. The RPS tools will be rewritten for the Sun environment and, like Proto, will conform to the window oriented, Open Look user interface standard.
The results of CORE requirements elicitation and analysis activities, contained in various Analyst generated diagrams, will be file transferred into the REW object manager. There they will automatically be converted into Proto equivalents. For example, the CORE single viewpoint model is similar to a data flow diagram and will appear in Proto as a graph with process nodes corresponding to CORE activities. This means that the requirements as captured by Analyst will be available in Proto as a partially completed prototype specification.

Integrating the RPS into the REW will involve rewriting its individual user interface prototyping and performance assessment tools for the Sun/Unix environment. This is an extensive redesign which will integrate capabilities for generating static displays (e.g., maps) with dynamic object movement (previously separated in RPS) and emphasize graphic interfaces to the performance models (mainly template driven in RPS). Performance model output will also be redesigned to provide analysts with graphic displays of modeling results, such as plots of resource utilization as a function of time.

The integration of Proto into REW will involve the least amount of effort since it is native to the REW platform. However, the REW will integrate Proto and the user interface prototyping tools so that displays can include dynamic graphic objects which are known to Proto and which can be driven from a Proto behavior using SSDL.

It is planned to have the REW operational by the fall of 1992.

5: Future Directions

In the future, the emphasis in requirements engineering technologies at RL will shift to considering more formal approaches which take advantage of matured AI technologies such as the Knowledge Based Software Assistant (KBSA) [7]. Since the early 1980's RL has been developing the KBSA as an alternative software development paradigm in which a formal executable system specification evolves through the elicitation and transformation of informal requirements expressed in representations familiar to the application scientist or engineer.

5.1: Knowledge Based Requirements Assistant (KBRA)

The Knowledge-Based Requirements Assistant (KBRA) [8] facet of the KBSA was developed to support requirements engineers who typically express requirements and preliminary designs in a variety of ways, using the method that best fits the problem domain. It provided multiple styles of requirements representation, (i.e., data flow, entity relation, and state diagrams as well as spread sheets and text), and captured the evolving system description in a single underlying knowledge base. This made it possible for the analyst to move among representation styles with the system automatically updating and maintaining consistency among the representations.

5.2: Specification Assistant

In parallel with the KBRA development, a project was undertaken to develop a system, the Specification Assistant [9], that would assist in the creation, elaboration, analysis and understanding of formal, executable system specifications. The Specification Assistant presumes that starting from an initial high-level definition of a system, a specification evolves through a gradual process of elaboration, refinement, and revision. Specifications may also be formed by assembling smaller, previously developed components. The specification editor provides support for specification evolution in the form of high-level transformations which provide controlled design elaboration through stereotypical changes. Errors and inconsistencies are discovered and corrected as the specification is developed by providing assistance in analysis and understanding of the specification. Capabilities such as symbolic evaluation, influence graph generation, and resource analysis supplemented by behavior explanation and specification paraphrasing assist in the evaluation and validation of the design. Supported by a knowledge base in which specifications are represented, this system provides a set of tools for analysis, browsing, editing and paraphrasing formal specifications. The tedium of system specification is significantly reduced with the added benefits of greater understanding and validity of the specification along with the higher quality software produced through this method.

Significant progress toward KBSA goals is found in a recently completed integrated system for Acquisition of Requirements and Incremental Evolution of Specifications (ARIES) [10] built upon the results of the earlier efforts. ARIES is a transformation-based software environment supporting the acquisition of requirements and the elaboration and validation of specifications. Since all presentations that ARIES supports are views of the same underlying system description represented in the knowledge base, no real distinction is drawn between requirements and specifications. Requirements may initially be expressed informally, in hypertext and graphical diagrams much as in the KBRA. They are then gradually elaborated and transformed, evolving into formal specifications as in the Specification Assistant. Validation assistance is provided by paraphrasing
specifications in natural language, presenting them in graphical diagrams, and testing them through simulation of the system being built. The specification of large systems was a specific concern of ARIES and mechanisms have been provided to enable reuse and for dealing with the inevitable conflicts arising among groups of collaborating individuals. Use of this knowledge-based technology will greatly improve the ability to develop complex systems. ARIES is demonstrating that significant change will occur in the way systems are developed. Multiple dynamic visualizations of system descriptions will greatly enhance the ability of engineers to understand and manipulate designs. Designs and design components will be more effectively reused within and across domains with a much greater payoff than for reuse at the code level. Automated reasoning about system descriptions will greatly improve productivity and quality by providing increased feedback in the form of critiquing and consistency checking and in propagation of design decisions. Useful system life times will be greatly lengthened as they evolve in a controlled manner. Responsiveness to change will be more rapid and the integrity of the resulting system will not be compromised by interacting or conflicting patches.

5.3: Advanced REW design

RL is currently conducting a study of design alternatives for integrating and extending KBRA/ARIES technology within the REW. A major design issue is the degree of integration between the two environments. A higher risk approach would design the inference engine as an integral part of the integration, whereas a more conservative approach would have, for the near-term, a loosely coupled integration. This lower risk approach will allow the user access to full KBRA functionality such as presentations and localized inferencing, while still utilizing the current OODB architecture. The long-term goal includes a tightly coupled integration that directly utilizes the knowledge base.

6: Summary and Conclusions

Rome Laboratory has been conducting a program of research and development in requirements engineering and related technologies since the early 1980s. Its goals are to develop methodologies for problem analysis, specification and validation along with support tools which insure that stating the problem, changing it and ensuring its self-consistency are more productive. All developments are aimed at getting the technologies into the hands of domain users and system acquisition engineers so that they can better understand and state their needs.

The program is pursuing both conventional software engineering technologies and artificial intelligence approaches to develop an environment for modeling requirements which supports multiple external views of the requirements while maintaining a single consistent internal representation system which allows reasoning about the requirements. Work is currently underway to develop the architectural design for this environment.

7: References


