Documentation Driven Development for Complex Real-Time Systems

Luqi, *Fellow*, *IEEE*, Lin Zhang, Valdis Berzins, and Ying Qiao

Abstract—This paper presents a novel approach for development of complex real-time systems, called the documentation-driven development (DDD) approach. This approach can enhance integration of computer aided software development activities, which encompass the entire life cycle. DDD will provide a mechanism to monitor and quickly respond to changes in requirements and provide a friendly communication and collaboration environment to enable different stakeholders to be easily involved in development processes and, therefore, significantly improve the agility of software development for complex real-time systems. DDD will also support automated software generation based on a computational model and some relevant techniques. DDD includes two main parts: a documentation management system (DMS) and a process measurement system (PMS). DMS will create, organize, monitor, analyze, and transform all documentation associated with the software development process. PMS will monitor the frequent changes in requirements and assess the effort and success possibility of development. A case study was conducted by a tool set that realized part of the proposed approach.

Index Terms—Software development, documentation, agility, information representation, complex systems, real-time systems.

1 Introduction

A complex real-time system is generally composed of individual real-time systems that were developed by different organizations with different tools and run on different platforms. Development of a complex system is much more difficult than development of individual real-time systems. Nonessential software complexity of complex systems can have a greater negative impact on system behavior than for a single system. In general, complex real-time systems are usually deployed for long periods of time, are used globally, and have mission critical requirements. They demand real-time performance and high confidence. Attributes like system effectiveness, availability, reliability, safety, security, and clarity of design are all essential. Most importantly, complex systems must rapidly accommodate frequent changes in requirements, mission, environment, and technology. Consequently, they are often structured as coalitions of separate components to form systems of real-time systems with dynamic configurations. These component systems were generally developed by different organizations with different tools and run on different platforms. In addition, a wide variety of stakeholders (sponsors, developers, users, maintainers, etc.) are involved in the overall lifecycle of a complex system [9], [35].

These common traits of complex real-time systems invoke some complicated challenges in software system development, such as:

- How to improve the agility of system development, which means making development promptly adapt to quickly changes in requirements, mission, etc.
- How to lower the cost and shorten the time of development as well as guarantee the success.
- How to guarantee the quality and high confidence of the system.
- How to support the participation of a variety of stakeholders.

A large amount of research has been conducted on real-time systems. Progress has been made, but mostly on "point solutions" that address subareas of complex system development, e.g., real-time constraints and high confidence issues. Integrated systematic approaches that are clearly defined and easy to use are needed to meet the challenges in development of complex real-time systems.

1.1 Contributions

Based on our many years of research on real-time software systems, we realize that documentation plays a crucial role in overcoming problems throughout the software life cycle. This is especially true for complex systems. Despite the fact that documentation has proven to be a requirement in the quest to improve software quality, the full capabilities of using appropriate documentation have yet to be realized. This paper introduces a new documentation-based approach to software development, Documentation Driven Development (DDD) approach, which is comprehensive enough to apply to all phases of software development yet robust enough to handle the complicated issues associated with complex real-time systems.

In software development, the concept of documentation has been augmented to not only include static informal text and diagrams intended for human consumption, but also include dynamic information, such as executable test cases [14], [25]. In the approach proposed in this paper, we further extend documentation to all information needed to

- *The authors are with the Software Engineering Automation Center, Department of Computer Science, US Naval Postgraduate School, 833 Dyer Rd., Monterey, CA 93943.*
E-mail: {luqi, lzhang, berzins, yqiao}@nps.edu.

Manuscript received 5 Jan. 2004; revised 6 Oct. 2004; accepted 18 Oct. 2004.
Recommended for acceptance by D. Rombach.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0003-0104.

carry out the development process. Effective documentation should support humans to the extent that the relevant development processes are carried out by humans, and should support software tools to the extent development processes are carried out by tools. In the common case where an aspect of the development process is carried out by a collaboration of both humans and software tools, the documentation should provide two views, one for humans and one for tools. For such aspects, consistency and accurate correspondence between the two views are of most importance, and computer aid is needed to effectively realize these properties.

In this approach, models and simulations are included as documentation. Some typical models include computational models and design models. They serve as the basis to support development activities such as requirements analysis, architecture design, validation, and verification. Simulation and prototyping are examples of computer aided processes used to check the correctness of the requirements for the system under development. With this extension, documentation can provide more effective support for whole development process. The DDD approach proposed in this paper will address issues associated with complex real-time systems including needs to promptly adapt to new requirements and support participation of diverse stakeholders, while preserving high confidence and timing constraints. This approach will significantly improve the agility of software development and support partial automation of software development as well.

1.2 Related Work

Software Engineering aims to improve software quality and productivity by providing systematic, disciplined, and quantifiable approaches to software development. Documentation has been proven to play a key role in software engineering. Many theories, methods, and techniques related to documentation have been developed in the past decades. There are different specific documents associated with different development phases. Typical phases in the software life cycle include requirements analysis and definition, architectural design, implementation, composition, deployment, maintenance, and evolution.

In the requirements phase, a requirement definition, which is a kind of documentation, serves as a starting point for the whole software development process. Natural language is the most common form of requirement definition [22]. By modeling and formalizing the requirement definition, the formal documentation—the requirement specification—can be derived. In this case, the requirements specification is usually written in formal language. Typical examples include [13], [21], [52]. They use temporal logic to represent the formal requirements specifications that further serve as the basis for verification and validation.

The most important documentation used in the design phase is design specification. This acts as a blueprint for the actual coding by outlining the logic of individual code modules. It also assists maintenance programmers as they modify the program to add enhancements or fix errors. A design specification is generally described by formal or semiformal methods, such as hierarchy charts, logic charts, state transition diagrams, state machines, data flow diagrams, data dictionaries, object-oriented approaches, and a great number of formal languages [27]. Some typical formal and semiformal notations used for design specification include UML and some kinds of architecture description language. Prototype system description language (PSDL) [32], [33] is another typical design specification language for real-time embedded systems.

Configuration is another important aspect of software development that is done based on documentation support, such as architectural specification and component specification. In complex control systems, the configuration of components must be flexible enough to allow rapid online reconfiguration and adaptation to react to environmental changes and unpredictable events at runtime. For this purpose, an open software architecture [51] has been used for integrating control technologies and resources.

Although a lot of effort has been applied toward improving documentation technology [18], [20], [42], there are still open challenges that hinder documentation from providing efficient support for complex real-time systems development. The various representations of documentation increase the complexity of maintaining information consistency, increase the intellectual burden on stakeholders, and introduce the need for transformations that are tedious and error prone when carried out manually. Some formal representations with rigorous logic are conducive to machine manipulation but are difficult for human understanding. Informal representations such as natural language are comfortable for many system stakeholders but are too vague and ambiguous for direct use by computer tools. How to maintain consistency among information presented to both the humans and computer tools is still a challenge. In addition, to guarantee software quality in the end product, the information should be kept consistent among documents of successive development phases. Traditional documentation technologies do not solve this problem. Our DDD approach in this paper is going to attack above problems and enable documentation to provide more effective support for complex real-time system development.

Progress has been made on research on real-time systems. Stanford University conducted research on static verification of real-time embedded systems [6]. They presented a modular framework for proving temporal properties of real-time systems based on clocked transition systems and linear temporal logic. In this framework, the properties of real-time systems can be established by using deductive verification rules, verification diagrams, and automatic invariant generation. Carnegie Mellon University [12], [21], [47] and Kansas State University [15], [17] have conducted much research on static verification for high confidence embedded systems. This research uses model checking to verify the satisfactory realization of some properties related to high confidence that are defined in the requirements specification and architecture specification. In addition to research on static verification, research efforts have explored dynamic verification. Typical work in this area is runtime assurance based on formal specification, such as runtime testing and runtime monitoring. Lee and his real-time group at the University of Pennsylvania have conducted much research on this aspect [13], [26], [30]. Broy and Slotosch presented a method with comprehensive tool support to structure requirements and construct models from categorized requirements for embedded systems [9], [10]. These results can be used to address requirements changes. Within the framework of documentation driven development approach in this paper, many of these state-of-the-art methods can be incorporated and integrated to achieve a high confidence system.

How to deal with frequent changes of requirements is drawing increasing attention in the software engineering community. Agile software development has been presented as a solution [50]. This informal approach focuses on individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan [5]. Thus, compared to other methods heavily depending on traditional documentation, many current agile software development methods try to provide better communications with the user, reduce comprehensive documentation, and adapt rapidly to requirements changes. Some typical agile development methods are extreme programming (XP) [4], SCRUM [46], dynamic software development method (DSDM) [49], adaptive software development [24], feature-driven development [41], lean development [43], rapid application development [38], etc.

Agile methods obtain the advantage of speedy response by increasing direct communications between users and developers. A problem with these approaches is that the users are required to be knowledgeable and well versed in software domain skills to be able to participate in the development process. Following some of the agile principles runs a high risk when the participating individuals do not have the required domain skills [16].

Our idea is to improve agility on a large scale by integrating different disciplines through an efficient documentation system. Making suitable use of documentation in the development process can reduce the requirements for participants to have specific information. Moreover, by generalizing and abstracting the essence of documentation and exploiting the capability for computer-aided documentation, documentation can be used to significantly improve the agility of software development of complex systems while sacrificing automation to a minimum extent.

The rest of this paper is organized as follows: Section 2 gives an overview of the Documentation Driven Development (DDD) approach. Section 3 describes the Documentation Management System (DMS) in detail. Section 4 describes the Process Measurement System (PMS). Section 5 introduces the computational model used for automated software generation. Section 6 introduces the tool set CAPS-PC that has realized part of ideas of DDD. Section 7 gives a case study conducted by CAPS-PC. Section 8 provides the conclusions and future work.

2 Overview of Documentation Driven Development

Agility of development requires that the development group, comprising system designers, hardware developers, software developers, and customer representatives should be well-informed, competent, and authorized to consider possible adjustment needs emerging during the development process life-cycle [1]. Our idea to improve agility is to improve the documentation system. First, we generalize the idea of documentation to make it an active part of the process and to provide value added via automated decision support. Documentation in our approach is computationally active structured information with automated decision support and representations in multiple formats.

Documentation can be classified into two categories: documentation for tools and documentation for humans. Formats of documentation for tools include mathematical notations (such as temporal logic or process algebra), design languages (such as PSDL or ADL), programming languages (such as Ada or Java), system models, requirements/design specifications, ontologies, source code, test cases, and application data (such as geographic databases, results of measurements, medical records, financial databases, tables of properties of physical materials, and any other reference information relevant to system design). Formats of documentation for humans are typically graphical or in a form easily understood by humans including text annotations in natural language, decision tables, spread sheets, or computed attributes. They can be expanded to include video and audio clips, live simulations, queries, etc.

Fig. 1 depicts a framework for the Documentation Driven Development (DDD) approach. It includes a Document Management System (DMS) and a Process Measurement System (PMS). Information from any activity involved in the software development process as well as the entire software life cycle will be recorded, managed, and transformed by the DMS. The information will be stored in a form that will support a variety of formal and informal documents for different stakeholders and can be manipulated by a set of software tools. Eventually, the DMS will monitor and drive the overall development process and be applied throughout the entire software life cycle. The DMS makes the development processes transparent and traceable, enables documentation to be updated quickly, and facilitates communication and collaboration between stakeholders to promptly respond to changes in requirements. The Process Measurement System (PMS) is used to track and analyze changes in requirements to verify the feasibility of the requirements, assess effort and risk of development, provide clues to modify the requirements, and measure the required high confidence properties. PMS is based on a set of quantitative metrics, most of which can be automatically collected in requirements phase. These metrics are stored and organized in the documentation management system. PMS and DMS will help to rapidly accommodate frequent changes in requirements for the development of complex systems.

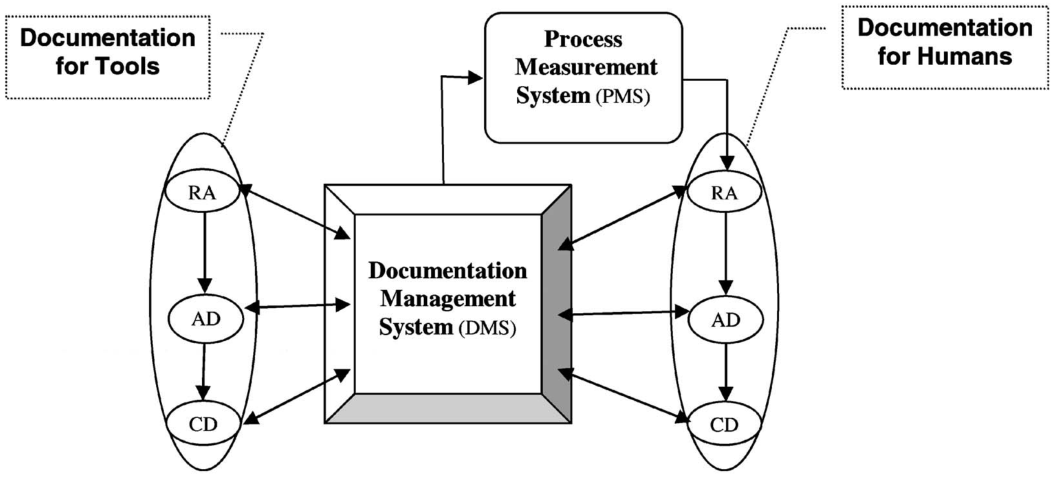

Documentation for Tools
Process Measurement System (PMS)
Documentation for Humans
RA
AD
CD
Documentation Management System (DMS)
RA
AD
CD

Fig. 1. DDD technical framework. RA: requirements analysis. AD: architecture design. CD: component design.

3 Documentation Management System (DMS)

DMS will create, organize, monitor, analyze, manipulate, and display documentation. These are basic operations on documents associated with system development. It will record documentation including requirement specifications, abstracted models, stakeholder input, design rationale, project management information, source code, etc. It will extract relevant information from all development activities such as requirements analysis, prototyping, architectural design, software composition, system verification and validation, and system deployment. It will also support key activities including automated software generation via computational models [35], interoperability via connection models [56], etc.

DMS mainly consists of three parts (Fig. 2):

1. The Documentation Repository (DR) is the core of DMS. It is used to store the information in a structured, well-organized format with a well defined meaning, to find appropriate subsets and projections of the documents for particular purposes, and to extract computed attributes of documents.
2. The Representation Converter is used to transform and present documentation in DR to different stakeholders and tools.

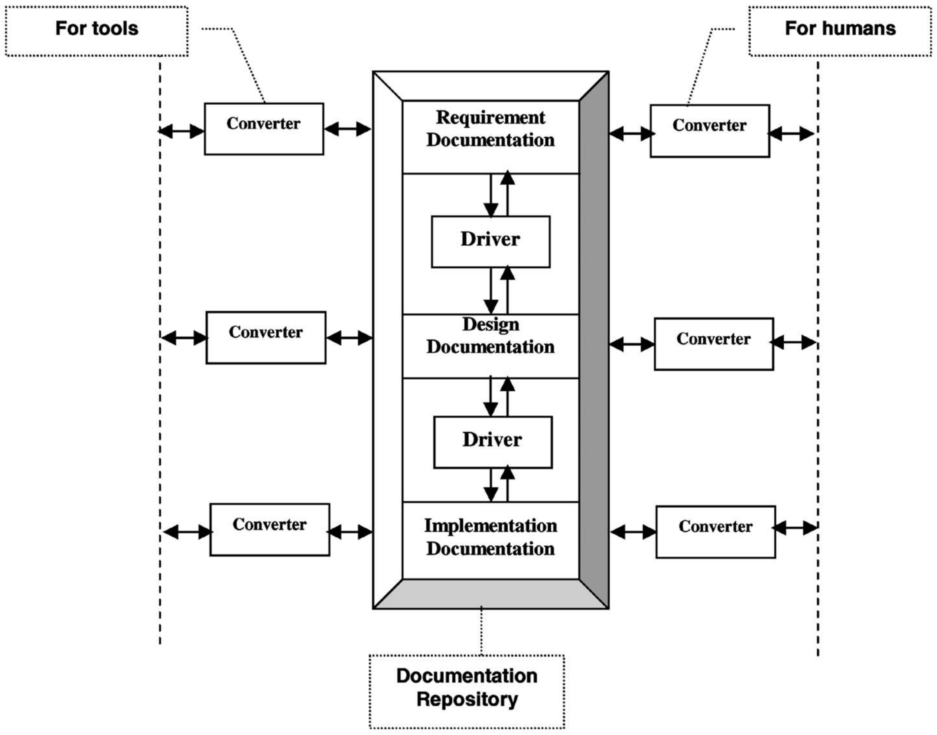


Fig. 2. Structure of documentation management system.

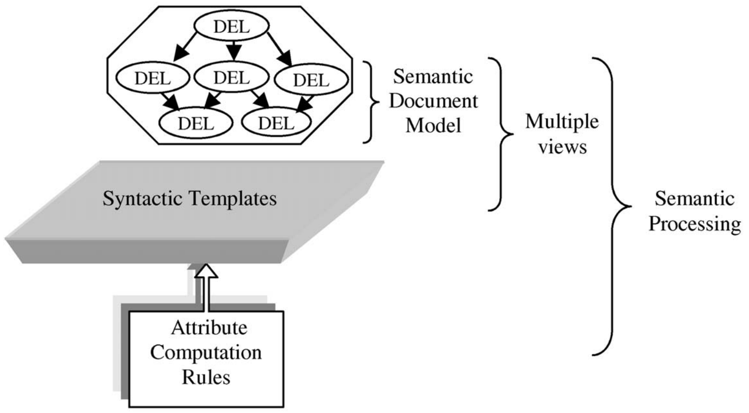


Fig. 3. Documentation repository.

3. The Transition Driver promotes the transition of information in the DR from one development process to the next.

3.1 Documentation Repository

Keeping documentation consistently up to date is difficult because of the various representations of information used in various stages of the development. Differing representations of the same documentation increase the complexity of maintaining information consistency and also decrease the efficiency of communications between humans and computer tools. This paper presents a documentation repository in which a common internal representation is used to represent all information contained in the documentation.

The repository representation is the core of the documentation driven development approach. All the information related to development process is modeled and stored in the documentation repository. Each development phase has its own area in the documentation repository and its own modeling level. The information is transformed between different documentation areas that belong to successive development phases. Typical examples of the information stored in the repository are requirement specifications, abstracted models, stakeholder input (from sponsors, end users, developers, technical supporters, etc.), ontologies, design rationale, project management information, and the source code. The repository uses a structured central representation for this information so that different stakeholders can communicate with each other based on consistent information and this information can be consistently transformed between successive development phases. Fig. 3 illustrates the repository representation and shows that the documentation repository includes three kinds of artifacts, i.e., document elements (DEL), a set of syntactic templates and a set of attribute computation rules. A document element is a basic building block consistent with the semantics of the information contained in the documentation. It is represented as an Attributed Object Graph Model (discussed in detail later). This object model represents the information contained in the documentation, and its instances form an attributed object graph. The schema for this object model is part of the ontology for the documents in the repository, and serves to define the meanings of the documents. The document elements are the nodes of this graph.

The amount of information associated with each node depends on the degree of formalization for each documentation type. Formal representations have explicit structure at a fine granularity and very simple information associated with individual document elements. Informal representations have only a large granularity structure and can have lengthy annotations attached to the nodes. Document elements hold the key information extracted from all the requirements, models, activities, and processes related to system development. Consistent transformations can be automated for the aspects of the information that have been formalized in the graph structure.

Syntactic templates are object operations with parameters. The purpose of a syntactic template is to materialize the part of a specific documentation view that corresponds to a given document element. The parameters represent the relevant properties of the context and the descendent nodes of the document element. Syntactic templates are designed together with specific sets of rules that govern the manipulation of the data stored in the document elements. The content of the document elements is treated as repository information and the different templates govern how that information is used and presented to the stakeholders and tools in the computer development environment. The combination of a document element and different syntactic templates forms the multiple view presentation of the same information. Combining document elements with appropriate templates and rules can also transform the information between representations written in different description languages.

Attribute computation rules represent the methods for computing derived document attributes. They turn the repository into an active project support system. These rules are organized in a rule base. The rule base is designed to be open in the sense that new rules can be added without changing the effect of any complete subset of the previous version of the rules. This property supports reliable incremental extension of the automation support provided by the repository and enables steady improvement of decision support processes. In the long term, the repository will perform a variety of automated and computer aided functions such as the following:

- materialize external representations of documents suitable for particular stakeholders or tools,
- find appropriate subsets and projections of the documents suitable for particular purposes,
- extract computed attributes of documents,
- transform data among different representations as needed to support integration of development processes and tools,
- configuration management of the documents,
- project management based on management documents, and
- consistency checking and consistency maintenance by rederivation of the dependant attributes in response to changes.

In summary, the proposed view of documentation is much closer to an active and intelligent information base than it is to a passive stack of paper.

3.1.1 Attributed Object Graph Model

Attributed Object Graph Model is an object model of information in the documentation repository. It has a nested structure with potentially shared nodes, i.e., directed acyclic graph structure. This representation is a generalization of abstract syntax trees and is designed to represent and efficiently analyze constructs that appear in more than one context. This is a common pattern in software artifacts—for example, an operation is typically defined once and called from many different contexts.

In the attributed object graph model, each node represents a semantically meaningful structure, such as an individual requirement, a subsystem, an operation, or an operator within a logical expression. The nodes are the finest grained structures visible to the attribute computation rules. Each node is an instance of an abstract data type. The computed attributes of each node correspond to operations of the data type. Invoking appropriate methods of the data type can derive the value of an attribute. Attribute computation rules are declarative definitions of these methods.

The semantics of attribute evaluation in the attributed object graph model is a generalization of the corresponding semantics in an ordinary attribute grammar. The two are the same when the graph is a tree. The difference shows up for inherited attributes of shared nodes: In an attribute grammar, each node can have at most one parent, but a shared node in an attributed object graph can have more than one parent.

We require the type of an inherited attribute to be a lattice. In implementation terms, the type must implement the lattice [T] interface with operations

$bottom : T$ — least element
$lub(T, T) : T$ — least upper bound
$le(T, T) : bool$ — approximation ordering

and these operations must satisfy the standard properties of a mathematical lattice.[1]

1. The following must be true for all x, y, z in T: $le(x, x)$, $le(x, y)\ \&\ le(y, z) \rightarrow le(x, z)$, $le(x, y)\ \&\ le(y, x) \rightarrow x = y$, $le(bottom, x)$, $le(x, lub(x, y))$, $le(y, lub(x, y))$, $le(x, z)\ \&\ le(y, z) \rightarrow le(lub(x, y), z)$.

The semantics of an inherited attribute A with a defining expression E is the least upper bound of the values of E in all contexts (i.e., the set of all parent nodes). In implementation terms, an attribute computation rule of the form $child.A = E(parent.A)$ can be realized with an initialization $node.A := bottom$ (for all nodes) and an incremental update step $child.A := lub(child.A, E(parent.A))$ which is enabled in the context of each parent node whenever the value of $parent.A$ changes in that context.

To make the above restriction on attribute types less burdensome, we propose a default extension of all types (a uniform subtype definition) that adds a new constant "bottom" representing an undefined value, another new constant "conflict_error" representing a conflict between two incompatible values inherited from different contexts, and the usual flat ordering on simple data types:

```
$le(x, y) = (x = bottom)\ or\ (x = y)\ or\ (y = conflict\_error)$
$lub(x, y) = if\ (x = bottom)\ or\ (x = y)\ then\ y$
    $else\ if\ (y = bottom)\ then\ x$
    $else\ conflict\_error$          — display an error diagnostic
```

This default can be explicitly overridden by the designer for data types where this makes sense. An example from the domain of timing constraints illustrates the idea:

```
TYPE DEADLINE EXTENDS INTEGER
    $bottom = MAXIMUM\_INTEGER$
    $le(x, y) = x \geq y$
    $lub(x, y) = MIN(x, y)$
```

This corresponds to the idea that if a program meets a given deadline, then it also meets any later deadline. This idea is made precise by the definition of the le operation on deadlines, and the corresponding definitions of $bottom$ (the least restrictive deadline possible) and the lub operation (the simultaneous application of two different deadline constraints). Thus, a component that inherits deadlines of 100ms, 75ms, and 120ms from three different requirement documents is subject to a design constraint to execute within 75ms (since $lub(lub(100, 75), 120) = 75$ for the deadline type defined above). This example illustrates the kind of structure needed to do decision fusion, for data types in which two different decisions may not necessarily conflict, and instead may be consistently combined into a refined composite decision.

To ensure the high confidence of a real-time system, it is important to keep timing properties consistent during the whole development processes. This means the information related to timing properties needs to be consistently identified in documents belonging to different development phases. In the next section, we will use timing properties as an example to illustrate the application of the proposed attributed object graph model to the problem of maintaining document consistency.

3.1.2 Attribute Computations for Document Management

The attributed object graph model was designed to realize documentation checks and transformations that support high confidence system development. These computations are used to

1. calculate the attributes from the information in the documentation repository,
2. transform the information from one development phase to another,
3. analyze the consistency between the information transformed between development phases, description languages, and information views, and
4. extract subsets of documents needed for particular purposes. The declarations of these computations form a set of attribute computation rules.

In the development process, the documentation generated in early development phases is taken as input for the next phases and guides the development activities in that phase to generate the output documentation. To ensure the quality of the end product, it is important to keep selected nonfunctional properties needed for high confidence visible and consistent during the whole development process. These high-confidence properties should be kept consistent between the documentation generated in an early phase and that generated in the next phase. Although the format of this kind of "key information" may be different between two development phases, the information of the later phase should imply that of the earlier. For example, in the requirement phase, requirement documentation may include information describing a customer request for deriving the computation result within a constrained time. In the design phase, the design documentation should include information with the same implication, such as information related to the deadline, period, and maximum execution time that is sufficient to guarantee a response within the required time interval.

In this paper, we use timing property transformation between the requirements phase and design phase as an example to illustrate the application of attribute computation rules. Suppose that the requirements specification includes a maximum response time (MRT) constraint for a given service S and that, at the architectural level, S is realized by a software component C. The MRT appears at the requirements level because it is directly visible to the system stakeholders and is of vital concern to them since late control signals can have catastrophic consequences.

At the design level, this constraint is transformed into lower level constraints on the period and maximum execution time (MET) of a periodic software process. If the document element S in the requirements document is a parent node of the document element C in the design document, the design rule that ensures consistency of the two documents with respect to this issue can be expressed by the following simple attribute computation rules, where MRT is an attribute of S; timing_check, period, MET, and diagnostic are four attributes of C:

$C.timing\_check = (C.period + C.MET \leq S.MRT)$
$C.diagnostic = Unless(C.timing\_check, error\_message)$
— Unless (C, M) displays M if
$C = false$ and does nothing otherwise.

The rationale for this rule is that the worst case occurs when a request arrives just after the request stream has been polled. In this case, the transaction will start processing one period later, and the software can take up to the maximum execution time after the transaction starts to produce the result. This simplified example assumes that all processing is done locally, so that we do not have to account for any latency in the communications link between the machine running the component C and the machine running the consumer process waiting for the output of C.

A mature documentation repository will actively check many different generic design rules like the one illustrated in this simple example. The rule base will gradually grow as processes are improved and constraints related to high confidence attributes are gradually formalized, and rules are created to automatically check or realize them.

3.2 Representation Converter

The representation converter presents the repository documentation to different stakeholders and tools in a traceable, consistent, and understandable way. The converter is based upon the combination of the syntactic templates and a collection of specifying document elements. It will "combine" the content of the document elements and the syntactic templates together to create and present desired documents for different stakeholders and tools.

Subject hierarchies are used to control the representation complexity of documentation. Subject hierarchies are defined according to the level of the software development activities with related specific document customers. Different stakeholders have different responsibilities and levels of interest in the overall project. For this reason, different levels of information need to be identified to support accurate review, modification, and evaluation of the delivered documents. Subject hierarchy controls the abstraction level of information presented to stakeholders thereby eliminating the problems and confusion caused by presenting too much or too little information.

Formal specification, graphical depiction, natural language narration, and even multimedia files are used to present abstracted data models and/or development activities. The very nature of formalized and graphical presentations of intellectual models, methods, and design activities can provide support for both the computer element (with formal specification) and the human element (through graphical depiction). The former serves as the input to various computer processes (different software tools such as a code generator or a model checker), while the latter provides an easily understood graphical depiction to the various human stakeholders (manager, designer, and implementer).

3.3 Transition Driver

A transition driver serves as a process transition tool based on the combination of syntactic templates and a collection of document elements. Its function is to analyze the key information held by the templates and the document elements and to promote the transition of repository information from one development process to the next. A transitional driver has the ability to act in both a forward and reverse direction. It can drive the transition of information from one process to a succeeding one (forward) or from one process to a preceding one (reverse). In the first mode, the transitional driver promotes forward engineering of software products. The transition driver analyzes the preceding information (information used as an input), guides user's intervention, and then generates succeeding information (process output). In the second mode, the driver promotes reverse engineering of legacy software systems if necessary. In this case, the driver serves as an extractor. It performs analysis and extracts useful information from what is normally considered the output information from a phase and generates what should have been the input information for that phase. A challenge in this area is how to best manage designer and user interaction to extract specification and design information the way it should have been built, rather than capturing the way it actually was built, including all of the errors and faults. A first step is to support annotations that identify such faults with links to explanations of why they constitute faults.

4 Process Measurement System (PMS)

The function of the process measurement system is to monitor the frequent changes in system requirements, assess the effort and success possibility of the project, and measure the high confidence properties of the system. The PMS obtains necessary information from the documentation repository. The analysis results will be presented to the developers and users as feedback. This quick communication is a key factor to make development of a system agile: Feedback is most useful when it can be delivered while the relevant aspect of the system is still in the process of being created, rather than after it has been completed and other system decisions have been made based on a faulty version of that aspect.

The process measurement system will include two parts: 1) a measurement model for effort and risk of a project, 2) a measurement model for high confidence. We have introduced a set of metrics to measure the effort and the risk in an evolutionary software project [19], [40]. These metrics can be automatically obtained early in the requirements phase. They accommodate changes in requirements, process, technology, and resources of a project. Based on the set of metrics, a measurement model for effort and risk of failure of a project has been proposed [39]. With respect to the high confidence measurement model, we developed an Instantiated Activity Model (IAM) that supports a formal approach for safety analysis by providing precise metrics [36]. However, due to the limit of length, this issue will not be discussed in this paper.

4.1 Indicators for Risk Assessment

The success of a software project relies on many different factors. Much research has addressed the problem of identification of risk factors. Different kinds of taxonomies were given [7], [29]. We identified four major risk contributors: resource risk, process risk, product risk, and technology risk. Each of these factors introduces risks individually and due to their interactions.

Resource risk is affected by organizational, operational, managerial, and contractual parameters, such as outsourcing, personnel, time, and budget among other resources. Various approaches use subjective techniques, like guidelines and checklists [23], [29], [48], which when even supported by metrics, require experts' opinions.

Engineering development work procedures, such as software development, planning, quality assurance, and configuration management cause process risk. The more complex a process is, the more difficult it is to manage, and the more education, standards, reviews, and communication are required. Consequently, complexity grows. The software process risk has been partially covered by research in terms of subjective assessments about maturity levels and expertise [23], [48]. However, a more precise and objective method is required.

The interaction between the process and the resources defines the organizational fit. If there is a perfect match between the process and the resources, it is expected that the efficiency of the organization will reach its maximum. By contrast, if mismatches between the process and the resources exist, then it is expected that the efficiency will decrease. This observation has been proved in previous research [11].

Product risk is related to the final characteristics of the product, its complexity, its conformance with specifications and requirements, its reliability, and customer satisfaction. Requirements volatility and complexity are two basic product-risk factors. Requirements volatility refers to the speed of changes in the requirements. This measure shows the difficulty of the requirement elucidation elicitation process. The requirements volatility can be measured from the requirements baseline. The complexity of an object, in general, is a function of the relationships among the components of the object. The correlation between complexity and size has been showed by [2]. More size implies more time, effort, cost, and defects [7]. Changes in requirements are inevitable. However, a change needs not to be directly related to an increase in the complexity. For that reason, complexity and requirements volatility are independent metrics that represent different aspects of the product.

Technology risk mainly consists of two parts. One arises from the software technologies that are selected to implement the project. The other arises from the domain technologies involved in the project. The choice of implementing technologies should be subordinated to the project domain technologies and requirements. We can expect that the risk of a project based on a new technology will be definitely greater than that of a project based on a mature technology. A complex system is usually deployed for long periods of time. In the process of evolutionary development of a complex system, the related technologies will change significantly. Generally, the newer the technology is, the more quickly the technology changes. The impact of technology maturity on risk is apparent and nonnegligible.

The indicators requirements volatility, organization efficiency, product complexity, and technology maturity will be the cornerstone of the risk management in software projects.

4.2 Metrics for Requirements Volatility

Requirements changing is the most significant characteristic for a complex real-time system. Requirements volatility clearly influences the possibility of project success. From the point of view of the metrics, a change in a requirement can be viewed as a death of the old version and a birth of the new one. The requirements volatility can be obtained from birth-rate and death-rate. Birth-rate is defined as the percentage of new requirements incorporated in each cycle of the evolution process. Death-rate is defined as the percentage of requirements that are dropped by the customer in each cycle of the evolution process. The requirements volatility (RV) is defined as:

$$\mathrm{RV} = \mathrm{BR} + \mathrm{DR},$$

where,

$$\mathrm{BR} = |\mathrm{NR} - \mathrm{PR}|/|\mathrm{NR} \cup \mathrm{PR}| * 100\%,$$
$$\mathrm{DR} = |\mathrm{PR} - \mathrm{NR}|/|\mathrm{NR} \cup \mathrm{PR}| * 100\%,$$

NR is the set of requirements in the current version, PR is the set of requirements in the previous version, and |S| denotes the number of elements in set S.

4.3 Metrics for Organization Efficiency

The efficiency of the organization can be measured by observing the fitness between people and their roles in the software process. The skill match between the person and the job is required to estimate the speed in processing information and the rate of exceptions, which in turn affect efficiency. Efficiency also depends on many factors like team structure, experience, and tools. Simulations we conducted have shown that there exists an easier way to estimate efficiency by observing the ratio between direct working time and idle time. The efficiency metric (EF) is defined as:

$$\mathrm{EF} = \mathrm{Dwork}\%/\mathrm{Idle}\%,$$

where $\mathrm{Dwork}\%$ is the percentage of direct working time and $\mathrm{Idle}\%$ is the percentage of idle time.

4.4 Metrics for Product Complexity

To estimate the complexity before a product is finished is challenging. Based on our previous work, the Prototype System Description Language (PSDL) [33], we developed complexity metrics that can be calculated in the requirements phase [19]. These metrics can be defined by using a hybrid complexity measure that properly accounts for data flow and the properties associated with each operator and data stream in PSDL. A complexity metric FC is defined as follows:

$$\mathrm{FC} = \sum_{i=1}^{n} w(o_i)[dsi(o_i) * dso(o_i)],$$

where, $w(o_i) = 1 + \sum_{k=1}^{m} pw_k * c_{ik}$ is the total property weight of operator o_i. pw_k is the property weight of the kth property, with $0 \leq pw_k \leq 1$ and $\sum_{k=1}^{m} pw_k = 1$. c_{ik} is the property occurrence coefficient, with $c_{ik} = 1$ if operator o_i has property p_k and $c_{ik} = 0$ otherwise. m is the number of property types in PSDL. $dsi(o_i)$ is one plus the number of data streams flowing into operator o_i. $dso(o_i)$ is one plus the number of data streams flowing out of operator o_i. n is the total number of operators.

4.5 Metrics for Technology Maturity

A new technology becomes mature in the process of transition from a scientific discovery to routine engineering practice in product development. Technology transition is referred to as diffusion in the literature. Diffusion is the process by which an innovation is communicated through certain channels over time among the members of a social system. Based on information theory, communication theory, and statistical mechanics, we developed a metric, named "technology temperature T", to measure the maturity of a technology [45].

According to information theory, the quantity of information in an ensemble of possible messages is measured by entropy. A message is made up of sets of terms. In this context, the relevant information is about a technology. Following reasoning similar to that used in statistical and condensed particle physics and recalling the standard definition from the thermodynamics, the temperature T for technology transition can be defined as

$$\frac{1}{T} = \frac{\Delta S_H}{\Delta n},$$

where, Δn is the change in the number of terms of a message alphabet Ξ. ΔS_H is the change in entropy. The entropy is defined as follows: For the message alphabet Ξ with the given probability mass function $p(x) = \Pr\{X = x\}, x \in \Xi$, X is a discrete random variable, the definition of information entropy is $S_H(X) = -\sum_{x \in \Xi} p(x) \log_2 p(x)$. The temperature is measured in "degrees" in a physical system; however, in the context of information degrees can be expressed in information units (bits). The value of T represents the maturity of a technology.

4.6 Measurement Model

A Weibull distribution can be used to build the measurement model. The Weibull distribution was originally used to model strength of Bofors's steel, fiber strength of Indian cotton, length of syrtoideas, fatigue life of steel, statures of adult males, and breadth of beans. Many authors have advocated the use of this distribution in reliability and quality control [28], [37]. The three parameter Weibull probability distribution function (pdf) is defined as follows:

$$\mathrm{pdf}: f(x) = \begin{cases} 0 & x < \gamma \\ (\alpha/\beta^{\alpha})(x-\gamma)^{\alpha-1} \exp(-((x-\gamma)/\beta)^{\alpha}) & x \geq \gamma, \end{cases}$$

where

- x is the random variable under study. In our context, x can be interpreted as development time.
- $\alpha(> 0)$ is a shape parameter. It affects the skew of the function. When $\alpha = 1$, the function reduces to the exponential distribution. The combined effect of α and β controls the variability of the pdf.
- $\beta(> 0)$ is a scale parameter that stretches or compresses the graph in the x direction.
- γ is a location parameter that determines the mean of the pdf.

Relationships between the parameters in the model and the quantitative metrics above have been identified via a large number of empirical experiments. This model works well especially for real-time applications [39]. When the metrics are input then development effort and success possibility of the project can be estimated by the model. The outputs of the model are important supporting information to help the sponsors and developers to make decisions about the next process.

5 Automated Software Generation Based on a Computational Model

DDD integrates key processes in the software life cycle by the documentation management system (DMS). Models, activities, prototypes, and simulations involved in these processes will be stored and manipulated in DMS. Supported by DMS, automated program generation can be realized based on a well-defined computational model and series of relevant techniques. A computational model was developed to describe the emergent properties, the interactions between component systems, and constraints associated with both functional and nonfunctional properties of a complex real-time system [35]. A system ζ is modeled as:

$$\zeta = (S, E, C, D, F_1, F_2),$$

where S is the component system set, $S = \{s_i | i \in [1, n]\}$, s_i denotes the component system constituting the whole system (n is the number of component systems in the whole system), $E = \{e_{jk} | j, k \in [1, n]\}$ denotes the interaction sets between component systems, e_{jk} denotes the set of interactions from component system s_j to component system s_k, and $C = \{c_i | i \in [1, n]\}$ denotes constraint sets on how the component systems are used in the given environment. c_i is a set of constraints on s_i. $D = \{d_{jk} | j, k \in [1, n]\}$ denotes constraint sets on interactions between component systems, d_{jk} is a set of constraints applied to interactions in e_{jk}.

Constraint sets C and D include the constraints for the design phase. They are refined from emergent properties G and high confidence constraints H of a system,

$$C = F_1(G, H); \quad D = F_2(G, H),$$

where G and H reflect the functional and nonfunctional aspects of requirements for whole system. G can be used for 1) generating code for monitoring failure events during the reliability assessment and testing and 2) verification. H can be used for 1) experimental assessment of high-confidence attributes and 2) possible static verification for some metrics, such as Maximum Execution Time (MET). F_1 and F_2 are two maps that map emergent properties and high confidence measures into local constraint sets on component systems and local constraint sets on interactions between component systems respectively. The maps specify what must be assessed to ensure that the system satisfies its requirement with high confidence, if it has already been certified that the individual s_i meet their requirements with high confidence. The constraint sets also represent a design for the systems integration, which will be realized by wrappers around the s_i.

Based on this model, a prototype system can be established to validate the requirements for a complex system. Well-formulated prototyping documentation can be used to promote system transition by extracting compositional architecture and evolving components. We found a way to build an explicit architecture for a prototyping system so that the product system can evolve through a transitional procedure [31]. The compatible composition model allows both explicit architecting and componential evolving by incorporating computer-aided prototyping techniques into a transitional process. Additionally, we introduced an object-oriented model for interoperability via wrapper-based translation [56]. This model performs transition from a computational phase, through a compositional phase, to a componential phase. During the transitional process, documentation passes throughout the development process. These results support automated software generation.

6 CAPS-PC

We have developed a Software Automation Integration Environment (CAPS-PC) that has realized part of ideas of Documentation Driven Development (DDD) approach proposed in this paper. CAPS-PC is based on our previously developed Computer Aided Prototyping System (CAPS) [33].

This tool set provides a PC-based computer-aided environment to support the modeling, analysis, and prototyping of systems under development. The environment helps create, modify, and maintain the requirements specification and architecture description documentation based on the information mapped from informal natural language descriptions. By building prototypes, CAPS-PC can be used to check the reliability of the software attributes and monitor the characteristics of the software according to changes in the context environment. It can also be used to explore the characteristics of high confidence real-time systems during the efforts of building and detailing the prototype models [34].

6.1 Overview of CAPS-PC

CAPS-PC is composed of five parts: Software Specification Editors, Software Project Management, Automatic Code Generation, Software Quality Facilities, and Software Execution Support. Each part is supported by extended facilities.

In CAPS-PC, a unified internal knowledge representation of software requirements is formalized in terms of PSDL definitions, which are designed for supporting automatic materialization of multiple views for different purposes. To the extent that the processes supported by documentation are performed manually, its representation should be understandable by humans. To the extent the processes are performed by tools, the representation should be tractable by software. CAPS-PC provides both kinds of views.

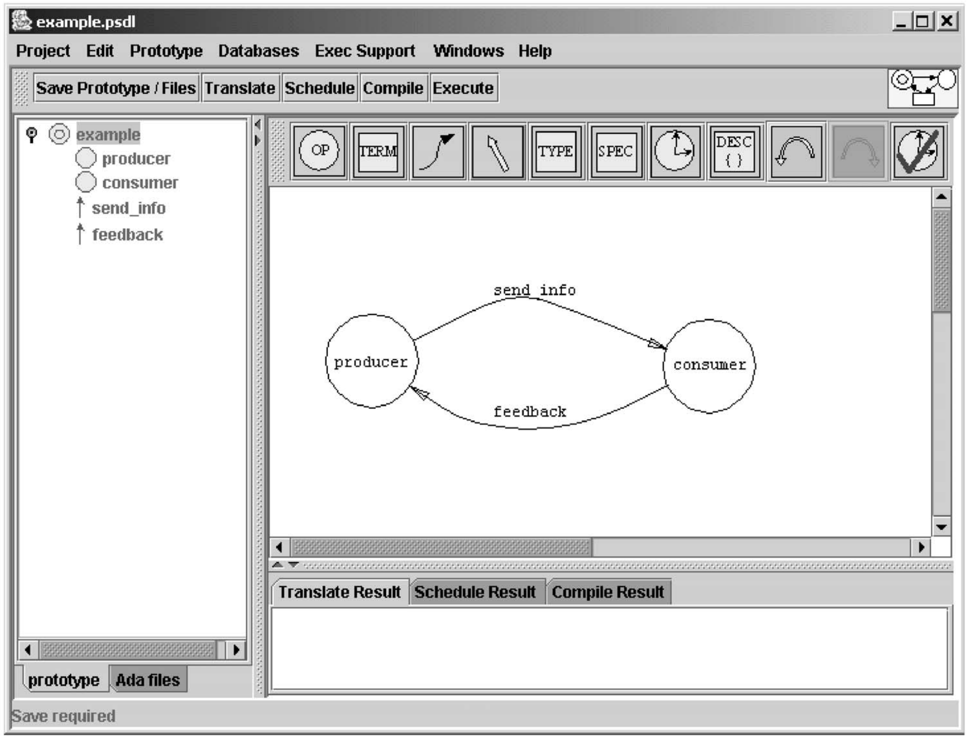

example.psdl
Project Edit Prototype Databases Exec Support Windows Help
Save Prototype / Files Translate Schedule Compile Execute
example
producer
consumer
send_info
feedback
OP TERM TYPE SPEC DESC { }
send_info
producer
consumer
feedback
Translate Result Schedule Result Compile Result
prototype Ada files
Save required

Fig. 4. The interface of CAPS-PC environment.

The tool set provides a system model editor for users to create and modify the computational model described in Section 5, a translator to check the syntax/semantics of the system model and to generate glue and wrapper codes to realize the design for the target system architecture, and a scheduler to analyze the timing constraints and to generate code to realize these constraints in the target architecture [32]. The software project management provides a platform to support the building of new software projects, retrieval of former projects, retracing of software development process, and software version control. The tool interface provides menus for users to manage their projects and compile source code into an executable prototype. The interface of CAPS-PC is shown in Fig. 4.

6.2 Main Features of CAPS-PC

6.2.1 *Unified and Multilevel Information Representation*

CPAS-PC provides a relatively simple documentation repository based on the DDD philosophy. Computation models described by PSDL are a main part of the repository. It is our initial effort to obtain a sophisticated documentation repository. The computational model will encapsulate most of the information related to the development process. The analysis of the software requirement constraints, such as real-time constraints, can be done by performing computation scheduling. The automatic prototype generation can also be completed by internally analyzing semantic meanings of the software specifications and doing software retrieval and adaptation based on matching of the specifications of the desired component and reusable component candidates.

CAPS-PC also provides multilevel point of view of the whole system. The construction of the prototype begins with a top-level definition, followed with a number of levels of refinement based on the functional decomposition; the whole prototype can be built with different granularities for different aspects, depending on the focus of the effort. For components with complex control functions that have many data transitions, the design can be detailed to trace all the process of data computation. For components with simple data handling, although the data may be larger scale, it can be designed as a single operator to handle all the incoming information.

The hierarchical design of models helps to organize the requirements specification in a way that can be tracked throughout the system's development according to abstraction level and responsible functionality. The clear and precise diagrams accompanying the documentation make it easy for a designer to check the consistency with text. Each operator defined in the diagram refers to the requirement item number in narrative documents, which makes it easy to find cross-references in the whole design model.

6.2.2 *Multiview Presentations*

CAPS-PC can provide both formal specifications that are suitable for use by software tools and informal representations that are easily acceptable for end users and sponsors and other stakeholders. Multiple views of the documents for different purposes are used to solve the confliction problem in information presentation. Views intended for human consumption will be tailored to stakeholder role (e.g., developer views are different from manager or user views), and those for tools will be tailored to the appropriate API's or tool input language.

The graphical display of the software structure in CAPS-PC provides the designer an easy way to define the functionality and software constraints. Meanwhile, it also makes it easy for the sponsor to get knowledge of the rough software product. A dynamic executable prototype with a user-friendly interface is another efficient information presentation style, which we treated as another kind of software documentation. The prototype generated by CAPS-PC provides another way to show designers their design results and helps them to find the defects and incompleteness, which also provides a vivid information display to show sponsors and let them check if it meets their imagination of the product. Engineers can instrument the prototype with gauges that measure and display runtime properties relevant to the design, such as the longest observed running time for a time-critical component. This mechanism will support quick and efficient communications between different stakeholders and tools and, therefore, support quick responses to requirements changes.

6.2.3 User-Centered Design

CAPS-PC not only has strong capabilities for building prototypes, but also provides high usability for prototype designers by considering human factors in its own software design. By involving the human factor consideration in the design of CAPS-PC, we help the CAPS-PC user to correctly understand the entire range of functionalities offered by CAPS-PC, learn how to apply them, and use them efficiently in a specific context of use or for a specific project. Several types of interaction principles are designed and partially implemented in the CAPS-PC tool set:

1. Highlight most important functionalities in the prototyping effort, such as translating, compiling, etc.
2. Make visible program artifacts and CAPS-PC functionalities when they are relevant and required. Irrelevant or rarely needed artifacts and functionalities compete with the relevant ones and diminish their relative visibility.
3. Provide contextualized feedback and appropriate messages to developers at the time of happening, which efficiently inform the developer about the system status and hints for possible consequential results.
4. Keep the developer informed of the CAPS-PC status and the prototype being designed would be helpful for the continuousness of the design activities.

6.2.4 Automated Code Generation

The prototype tool set provides the user with an execution support system that consists of translator, scheduler, and a compiler. The translator/scheduler generates the glue code needed for timely delivery of information between subsystems across the target network. For prototypes that require sophisticated graphic user interfaces, an interface editor is provided to interactively sculpt the interface in Unix system by using the TAE+ Workbench and automatically generate corresponding code.

7 Case Study

To validate the feasibility and effectiveness of CAPS-PC, we used the CAPS-PC tool set to prototype the CARA software in the Life Support for Trauma and Transport (LSTAT) system. This effort was required and supported by Army Research Office.

The LSTAT is a system to administer therapeutic intravenous (IV) fluids. The purpose of the LSTAT stretcher is to sustain trauma patients in transit to a medical facility until they can receive emergency medical treatment. The LSTAT includes multiple blood pressure sensors and an intravenous (IV) infusion pump. Computer Assisted Resuscitation Algorithm (CARA) software is the key control software embedded in the LSTAT system, which is a closed loop software system that drives the high output infusion pump. By constantly monitoring blood pressure, CARA can help to prevent hypotension and enforce fluid resuscitation [3]. The algorithm calculates the driving voltage for the infusion pump. This determines the volume and rate of IV fluid administered. Proper operation of the CARA software should enable safe transport of critically injured patients without the need for continuous monitoring by medical professionals. The requirements for the LSTAT system were given by a requirements definition document written using natural language expression [53], [54], [55], which gives detailed descriptions of the real-time constraints for the application process of the rescue strategies This is a safety critical real-time system, and the requirements documents were written by domain experts who were independent from the prototyping teams.

The situation and context of using CARA is complex and varied in terms of the level of the patient's blood pressure, and the patient's different body effects after pumping IV fluid. The driving voltage used to control the IV fluid pump needs to be calculated dynamically. A delay in delivery or an improper amount of IV fluid to the patient can result in serious injury or loss of life.

CAPS_PC was used to design the CARA system. During the design process, several supporting documents were built by the designers or generated automatically by using CAPS-PC. As the central chain of development documentation, these documents drive the specification, design, implementation, and even testing of the system development. The consistency maintained by the tool set between these documents provides a solid baseline throughout the development effort. The documentation generation function of the tool set makes it easy for the customer, user, and sponsor to understand, handle, and review the system during development.

First, the specification of requirements can be generated with completeness and consistency checking, according to the system functionalities and constraints. The graphical design process maintains the syntax of the requirements specification, and a further translation process ensures the semantic consistency of the specification document. Fig. 5 gives the top level of the computational model designed

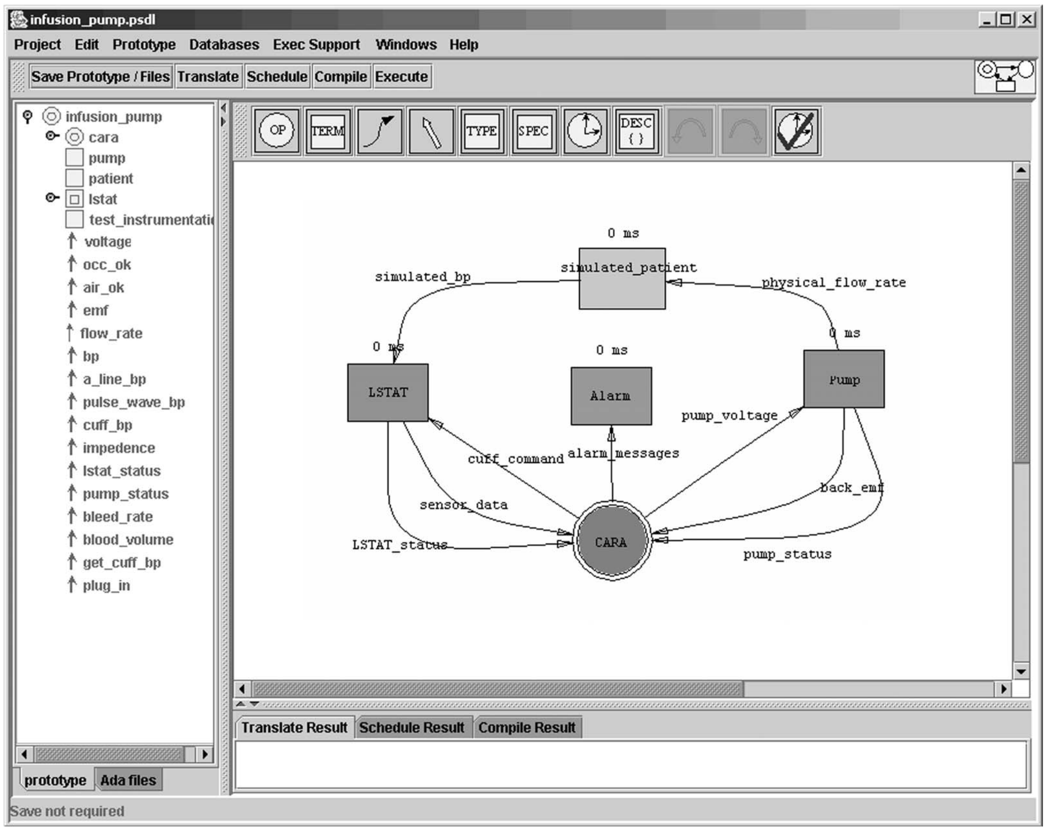


Fig. 5. Top level of the computational model of the system.

with CAPS-PC. This graphical presentation was obtained from requirements documentation. Fig. 6 is the decomposition of the "CARA" bubble in Fig. 5. The consistency of the decomposition was checked automatically. Second, the narrative description of a designed model in natural language can be generated based on the definition of model functionalities and constraints, which tell the customers and users what the system will and will not do. For example, for CARA's periodic blood pressure corroboration, the model generated the following narration: "if VALID_BP is CUFF_BP, and if MEAN_BP > 90, the CUFF_BP will get data from CUFF_MONITOR every 10 minutes." Third, by using technical terminology to describe the system's structure, data, and function can be generated based on the model specification. Information about input, such as where data comes from and how it is formatted; output,

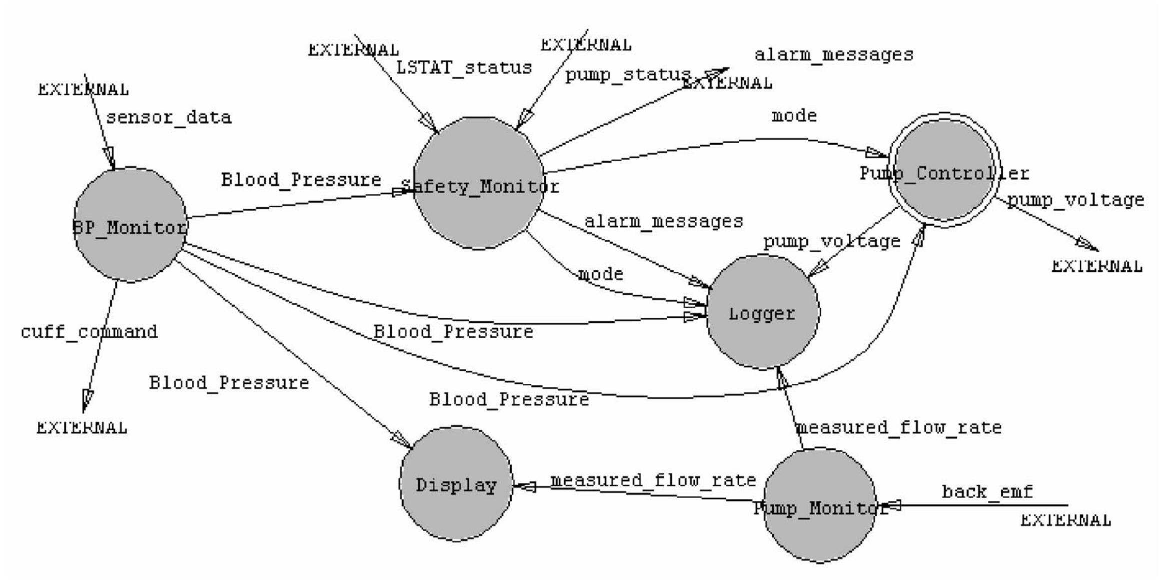
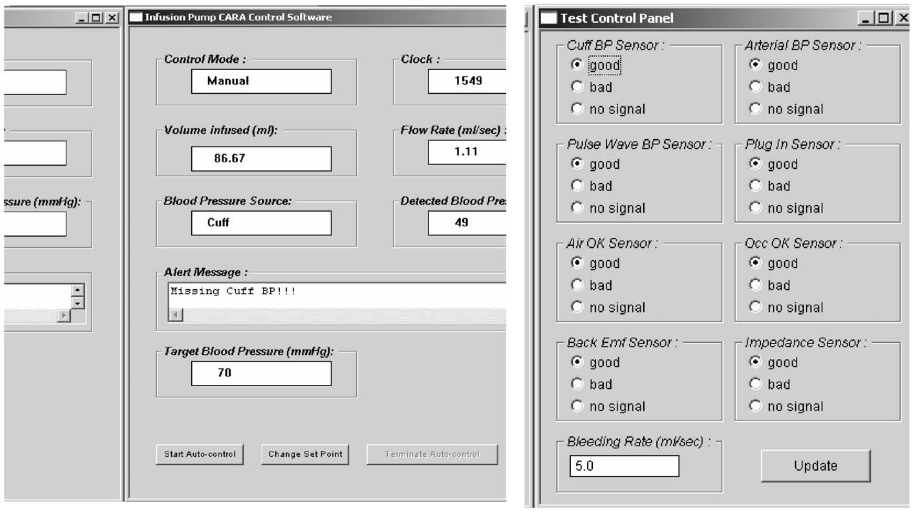


Fig. 6. Decomposition of "CARA" in Fig. 5.


Infusion Pump CARA Control Software

Control Mode : Manual

Clock : 1549

Volume infused (ml): 86.67

Flow Rate (ml/sec) : 1.11

Blood Pressure Source: Cuff

Detected Blood Pre 49

Alert Message : Missing Cuff BP!!!

Target Blood Pressure (mmHg): 70

Test Control Panel

Cuff BP Sensor : [✓ good] bad no signal

Arterial BP Sensor : [✓ good] bad no signal

Pulse Wave BP Sensor : [✓ good] bad no signal

Plug In Sensor : [✓ good] bad no signal

Air OK Sensor : [✓ good] bad no signal

Occ OK Sensor : [✓ good] bad no signal

Back Emf Sensor : [✓ good] bad no signal

Impedance Sensor : [✓ good] bad no signal

Bleeding Rate (ml/sec) : 5.0

Fig. 7. The graphical user interface of the executable prototype.

such as where data is sent and how it is formatted; general functional characteristics, such as periodic execution or sporadic execution; performance constraints, such as Minimum Execution Time; and specific fault-handling approaches can be generated and described in the design documentation. Fourth, during the generation of prototype code, the tool set can generate partial implementation documentation based on the specification structure. The name, type, and purpose of major data structures and variables, simple description of logic flow, expected input and possible output can be derived from the information defined in the model.

Furthermore, CAPS-PC can maintain the version control documentation for requirements specifications and model design. If any changes are made to the requirements during the remaining phases of development, the changes can be tracked from the requirements document, through the design process, all the way to the test procedures.

In our design efforts of the CARA software, the documentation generated or maintained by our tools greatly enhanced the effectiveness of the design procedure and the communication between the model designer, user, and reviewer. The specification documents were used by the tool set to interlink the individual tools. The structured narrative descriptions of system design models are used for detailed discussions or for system reviews.

The use of our tool set to prototype the CARA software based on its requirements documents showed that the development process and the communication between customer and the software developer can be improved by the integration of software development documentation and the enhanced information representation. An executable prototype (Fig. 7) was automatically generated and explicitly represented to designers and users. They can easily compare the prototype with requirements and give feedback to each other so that the design can be refined. Requirements changes will be considered in the process of comparing. Due to the time to get a prototype is short, requirements changes can be responded quickly.

This experiment shows that PSDL can effectively model complex real-time software. PSDL's triggering guards and execution guards provide a convenient means for users to specify state machines explicitly without being concerned with target code. The timer feature is useful in modeling complicated timing policies.

The tool provides an effective means to perform requirements consistency and understandability checking. It also provides some degree of computer-aided inconsistency checking and data entry propagation at the user interface level, and a semantic check via the translator.

8 Conclusions and Future Work

This paper explores a new view of documentation that can better serve development of complex real-time systems. The agility of development will be improved by the following merits of the proposed approach DDD:

1. DDD provides unified information management. All information involved in development process will be recorded in a form that can be manipulated, automatically analyzed. DDD will track changes and help to ensure that information will be transformed consistently from one phase to another.
2. DDD provides an environment of communication between stakeholders and tools, which give project managers, developers, sponsors, maintainers, and end-users the ability to express their opinions or propose requirements changes if needed by adding related documents via a user-friendly interface. This environment will facilitate stakeholder involvement while updating the requirements and consistently providing this information for later use.
3. DDD supports automated software generation by using a computational model, rapid prototyping, and other related techniques.
4. DDD provides a method to monitor frequent changes in requirements and assess the effort and

success possibility of the project with respect to changes in requirements.

By using the DDD approach in every phase of development, even the automated processes, it should become practically feasible to record, compile, and present information to different stakeholders and tools in a clear, understandable way at a level of complexity required to meet the stakeholders' needs. By having these different views available at various stages of development, stakeholders will be able to effectively monitor the development process and communicate with each other. This improved transparency provides valuable information needed for quality control and overall process improvement.

DDD also provides comprehensive support for software maintenance and evolution. In DDD, all the activities and information used by the development processes are accurately recorded and organized in a well-formulated documentation system that drives the system development and build processes. This will ensure that overall system properties are precisely documented and consistently updated and transferred throughout successive phases and available after system release. The documentation will retain sufficient detail to provide a sound basis for fault tracing, bug repairing, and overall system improvement. DDD will keep track of system configuration, document dependencies, and system status and enable the software to respond to future changes in requirements thereby supporting maintenance and evolution of the system. Keeping track of accurate dependency information is critical for automatically locating the relevant parts of a maze of documents for resolving a given system evolution issue.

From the viewpoint of long-term system construction, technologies for computer-aided documentation repositories will drive the form of documentation standard needed for more effective regulatory management. Much of the presented infrastructure can be generalized from software development to the entire systems engineering and certification process.

The main purpose of this paper is to introduce a new idea and a framework of a new development approach to realize the idea. There still have many scientific and technical details need to be studied and developed in the future.

Work needs to be done to extend the CPAS-PC to realize the whole DDD approach, such as, to deal with more formats of representations, to develop more templates for different documentation, to develop more precise quantitative metrics for risk assessment and high confidence property assessment for complex real-time systems, to develop tools that can fully realize the converter and the driver, etc. Future enhancements of CAPS-PC also include abstraction for data streams, visual queues for the declaration and use of timers, multiple views for requirements traces, and better facilities for constructing user define types.

DDD provides an integrated environment to conduct all activities involved in the whole development process, in which the documentation management system provides a mechanism to integrate different methods and tools for use by different activities. However, how to integrate current methods and tools to our environment is still an open problem. We have studied how to establish software development tool ontologies to improve interoperability in heterogeneous software development [44]. Tool ontologies contain the information needed to bridge semantic differences between different tools and to guide tool evolution to reduce the gaps and enable deeper integration. Work needs to be done to document and manipulate tool ontologies to integrate tools to support development processes. On the other hand, the DDD approach can be integrated into existing methods and tools. For example, the abstracted and well-organized information in the documentation repository can be used by formal verification and validation tools. Further efforts need to be applied to these issues.

References

[1] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile Software Development Methods-Review and Analysis," technical report, ESPOO, 2002.

[2] A. Albrecht and J. Gaffney, "Software Function Source Lines of Code and Development Effort prediction," *IEEE Trans. Software Eng.*, vol. 9, 1983.

[3] R. Alur, D. Arney, E. Gunter, I. Lee, W. Nam, and J. Zhou, "Formal Specifications and Analysis of the Computer Assisted Resuscitation Algorithm (CARA) Infusion Pump Control System," *Proc. Conf. Integrated Design and Process Technology (IDPT)*, 2002.

[4] K. Beck, *Extreme Programming Explained: Embrace Change.* Addison-Wesley, 2000.

[5] K. Beck et al., "Manifesto for Agile Software Development," www.agilemenifesto.org, Feb. 2001.

[6] N. Bjorner, Z. Manna, H. Sipma, and T. Uribe, "Deductive Verification of Real-Time Systems Using STeP," *Theoretical Computer Science*, vol. 253, no. 1, pp. 27-60, 2001.

[7] B. Boehm, *Software Engineering Economics.* Prentice Hall, 1981.

[8] B. Boehm, "Software Risk Management: Overview and Recent Developments," *Proc. 17th Int'l Forum on COCOMO and Software Cost Modeling,* http://sunset.usc.edu/events/2002/cocomo17, Oct. 2002.

[9] M. Broy, "Toward a Mathematical Foundation of Software Engineering Methods," *IEEE Trans. Software Eng.*, vol. 27, no. 1, pp. 42-53, Jan. 2001.

[10] M. Broy and O. Slotosch, "From Requirements to Validated Embedded Systems," *Proc. First Int'l Workshop Embedded Software (EMSOFT 2001)*, pp. 51-65, Oct. 2001.

[11] R. Burton and B. Obel, *Strategic Organizational Diagnosis and Design. Developing Theory for Application.* Kluwer Academic, 1998.

[12] S. Campos, E. Clarke, W. Marrero, and M. Minea, "Verus: A Tool for Quantitative Analysis of Finite-State Real-Time Systems," *Proc. Workshop Languages, Compilers, and Tools for Real-Time Systems*, pp. 70-78, June 1995.

[13] D. Clarke and I. Lee, "Automatic Test Generation for the Analysis of a Real-Time System: Case Study," *Proc. Third IEEE Real-Time Technology and Applications Symp. (RTAS '97)*, pp. 112-124, June 1997.

[14] A. Cockburn, *Agile Software Development.* Addison-Wesley, 2001.

[15] J. Corbett, M. Dwyer, J. Hatcliff, and Robby, "Expressing Checkable Properties of Dynamic Systems: The Bandera Specification Language," *Int'l J. Software Tools for Technology Transfer*, 2002.

[16] T. DeMarco and B. Boehm, "The Agile Methods Fray," *Computer*, vol. 36, no. 6, pp. 90-92, June 2003.

[17] X. Deng, M. Dwyer, J. Hatcliff, and M. Mizuno, "Invariant-Based Specification, Synthesis and Verification of Synchronization in Concurrent Programs," *Proc. 24th Int'l Conf. Software Eng.*, pp. 442-452, May 2002.

[18] P. Devanbu, P. Selfridge, R. Branchman, and B. Ballard, "LaSSIE: A Knowledge-Based Software Information System," *Proc. IEEE 12th Int'l Conf. Software Eng.*, pp. 249-261, 1990.

[19] J.P. Dupont, "Complexity Measure for the Prototype System Description Language (PSDL)," master's thesis, Naval Postgraduate School, Mar. 2002.

[20] J. French, J. Knight, and A. Powell, "Applying Hypertext Structures to Software Documentation," www.cs.virginia.edu/cyberia/papers/IPM97.pdf, 1997.

[21] D. Garlan, S. Khersonsky, and J. Kim, "Model Checking Publish-Subscribe Systems," *Proc. 10th Int'l SPIN Workshop Model Checking of Software (SPIN 03),* pp. 166-180, May 2003.

[22] L. Goldin and D. Berry, "AbstFinder: A Prototype Abstraction Finder for Natural Language Text for Use in Requirement Elicitation," *Automated Software Eng.*, no. 4, pp. 375-412, 1997.

[23] E. Hall, *Managing Risk Methods for Software Systems Development.* Addison Wesley, 1997.

[24] J.A. Highsmith, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems.* Dorset House, 2000.

[25] D. Hoffman and P. Strooper, "API Documentation with Executable Examples," *The J. Systems and Software,* vol. 66, pp. 143-156, 2003.

[26] H. Hong, I. Lee, O. Sokolsky, and H. Ural, "A Temporal Logic Based Theory of Test Coverage and Generation," *Proc. Int'l Conf. Tools and Algorithms for Construction and Analysis of Systems (TACAS2002),* pp. 327-341, Apr. 2002.

[27] http://www.comlab.ox.ac.uk/archive/formal-methods/, Oct. 2004.

[28] N. Johnson and S. Kotz, and N. Balakrishnan, *Continuous Univariate Distributions,* vol. 1. Wiley & Sons, 1994.

[29] D. Karolak, *Software Engineering Management.* IEEE CS Press, 1996.

[30] M. Kim, I. Lee, U. Sammapun, J. Shin, and O. Sokolsky, "Monitoring, Checking, and Steering of Real-Time Systems," *Proc. Second Int'l Workshop Runtime Verification,* July 2002, available at www.cis.upenn.edu/rtg/mac/.

[31] X. Liang, J. Puett, and Luqi, "Perspective-Based Architectural Approach for Dependable Systems," *Proc. ICSE 2003 Workshop Software Architectures for Depenable Systems,* pp. 1-6, May 2003.

[32] Luqi, "Real-Time Constraints in a Rapid Prototyping Language," *Computer Languages,* vol. 18, pp. 77-103, 1993.

[33] Luqi, V. Berzins, and R. Yeh, "A Prototyping Language for Real Time Software," *IEEE Trans. Software Eng.*, vol. 14, no. 10, pp. 1409-1423, Oct. 1988.

[34] Luqi and Z. Guan, "A Software Engineering Tools for Requirement Document Based Prototyping," *Proc. Seventh World Multiconf. Systemics, Cybernetics and Infromatics,* vol. VI, pp. 237-243, July 2003.

[35] Luqi, Y. Qiao, and L. Zhang, "Computational Model for High-confidence Embedded System Development," *Proc. Monterey Workshop—Radical Innovations of Software and Systems Eng. in the Future,* pp. 265-303, Oct. 2002.

[36] Luqi, X. Liang, M. Brown, and C. Williamson, "Formal Approach for Software Safety Analysis and Risk Assessment via an Instantiated Activity Model," *Proc. 21st Int'l System Safety Conf.*, pp. 1060-1069, Aug. 2003.

[37] M. Lyu, *Software Reliability Engineering.* IEEE CS Press, 1995.

[38] J. Martin, *Rapid Application Development.* Prentice Hall, May 1991.

[39] M. Murrah, "Enhancements and Extensions of Formal Models for Risk Assessment in Software Projects," PhD dissertation, Naval Postgraduate School, Sept. 2002.

[40] J.C. Nogueira, "A Formal Model for Risk Assessment in Software Projects," PhD dissertation, Naval Postgraduate School, Sept. 2000.

[41] S.R. Palmer and J.M. Felsing, *A Practical Guide to Feature-Driven Development.* Pearson Education, Feb. 2002.

[42] C. Paris and K. Linden, "Building Knowledge Bases for the Generation of Software Documentation," *Proc. Int'l Conf. Computational Linguisitics,* http://acl.ldc.upenn.edu/C/C96/C96-2124.pdf, 1996.

[43] M. Poppendieck and T. Poppendieck, *Lean Software Development: An Agile Toolkit for Software Development Managers.* Addison-Wesley, 2003.

[44] J. Puett, "Holistic Framework for Establishing Interoperability of Heterogeneous Software Development Tools," PhD dissertation, Naval Postgraduate School, June 2003.

[45] M. Saboe, "A Software Technology Transition Entropy Based Engineering Model," PhD dissertation, Naval Postgraduate School, Mar. 2002.

[46] K. Schwaber and M. Beedle, *Agile Software Development with SCRUM.* Prentice Hall, Oct. 2001.

[47] O. Sheyner, S. Jha, and J. Wing, "Automated Generation and Analysis of Attack Graphs," *Proc. IEEE Symp. Security and Privacy,* pp. 273-284, May 2002.

[48] Software Engineering Institute, "Software Risk Management," Technical Report CMU/SEI-96-TR-012, Carnegie Mellon Univ., June 1996.

[49] J. Stapleton, *DSDM: Business Focused Development.* Addison-Wesley, 1997.

[50] L. Williams and A. Cockburn, "Agile Software Development: It's about Feedback and Change," *Computer,* vol. 36, no. 6, pp. 39-43, June 2003.

[51] L. Wills, S. Sander, S. Kannan, A. Kahn, J. Prasad, and D. Schrage, "An Open Control Platform for Reconfigurable, Distributed, Hierarchical Control Systems," *Proc. Digital Avionics Systems Conf.*, Oct. 2000, http://controls.ae.gatech.edu/papers/kannan_dasc_00 .pdf.

[52] J. Wing, "Scenario Graph Generation and MDP-Based Analysis," Presentation at ARO Kickoff Meeting, Univ. of Pennsylvania, Philadelphia, May 2001, http://www-2.cs.cmu.edu/svc/talks/html/wing_files/frame.htm.

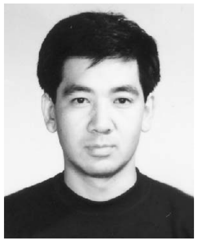
[53] WRAIR Dept. of Resuscitative Medicine, "Narrative Description of the CARA Software," Proprietary Document, WRAIR, Jan. 2001.

[54] WRAIR Dept. of Resuscitative Medicine, "CARA Pump Control Software Questions," Version 6.1, Proprietary Document, WRAIR, Jan. 2001.

[55] WRAIR Dept. of Resuscitative Medicine, "CARA Tagged Requirements, Increment 3," Version 1.2, Proprietary Document, WRAIR, Mar. 2001.

[56] P. Young, V. Berzins, J. Ge, and Luqi "Using an Object Oriented Model for Resolving Representational Differences between Heterogeneous Systems," *Proc. 17th ACM Symp. Applied Computing (SAC),* pp. 976-983, Mar. 2002.

Luqi worked on software-intensive system modeling and automation, computer-aided prototyping for real-time and embedded systems, project and system risk assessment, and control systems engineering using requirements specification languages and software architectures. Since her US National Science Foundation Presidential Young Investigator Award and an IEEE Technical Achievement Award, she founded the Software Engineering Program, has successfully completed more than a hundred projects and supervised hundreds of graduate students including dozens of PhDs, served as a professor and associate chair of computer science, and the director of the Software Engineering Automation Center at the US Naval Postgraduate School. She has also served on many editorial boards, including *IEEE Software*, *IEEE Expert*, and *IEEE Transactions on Software Engineering*, and chaired or served on a hundred program committees of conferences and workshops. She is a fellow of the IEEE.



Lin Zhang received the BS degree in 1986 from the Department of Computer and System Science at Nankai University, China. He received the MS degree and the PhD degree in 1989 and 1992, respectively, from the Department of Automation at Tsinghua University, China, where he worked as an associate professor from 1994. He served as the director of CIMS Office, National 863 Program, China Ministry of Science and Technology, from 1997 to 2001. He is currently working at the US Naval Postgraduate School as a senior research associate of the US National Research Council. His research interests include software engineering, real-time and embedded systems, risk analysis, project management, and system modeling and control.

Valdis Berzins received the PhD degree from MIT in 1979 and worked as a professor at the University of Texas, the University of Minnesota, and the Naval Postgraduate School, served as chair of the Software Engineering PhD Program Committee at Naval Postgraduate School. His research addresses many aspects of software engineering, with the objective of increasing productivity and software quality via automated decision support. His work has been supported by the US National Science Foundation, ARO, and computer industry, resulting in numerous publications, student theses, and software systems. He designed two specification languages, and initiated software merging as a research area by developing the first semantically sound method for automatically combining extensions to software systems, a comprehensive theory of modifications to software, and the first methods for combining changes to software that treat parallel programs and algorithm optimizations. Other research directions include requirements, lightweight inference, software synthesis, risk reduction, software architectures, interoperability, reuse, and reengineering. He is the author of books on software specifications and computer-aided software maintenance.

Ying Qiao received the PhD degree in computer science from the Institute of Software, Chinese Academy of Sciences, in 2001. She is currently a research associate at the US National Research Council. Her research interests include software engineering, embedded systems, and real-time scheduling.