

A Software Prototyping Framework and Methods for Supporting Human's Software Development Activities

Jennifer Z.Guan Luqi
Software Engineering Automation Center
Naval Postgraduate School
833 Dyer Road, Monterey, CA 93940, USA
{zguan, and luqi}@nps.navy.mil

Abstract

Software development environment is a platform for supporting software designer to design software based on the software requirement specification. It is an interactive system with lots of human being involved. Human error, as the main threat for the dependability of the software development system, may greatly harm the quality of the produced software. In this paper, we present a practical framework for software prototyping, addressing two types of threats for the software prototype that will occur during designer's prototype efforts. By illustrating information with multi-level representation, building an iterative prototyping loop, and providing solid project management, the human prototyping efforts is enhanced, refined, and organized. Several approaches to make the specification and prototype of the software requirement to be more accurate, complete, and consistent are proposed. They prevent the occurrence of human omission/slip and help to recovery the system states from human mistake.

1. Introduction

An interactive system is the software whose purpose is to support two-way communication between a computer and its user. Typically, the user will perform an action on the system such as clicking a mouse button, typing text, or moving a joystick. Following this, the system will react accordingly, by invoking an application, displaying information on a screen, or simply waiting for yet another user action. The bi-directional communication between human and the computer supports the information exchange and helps to formulize knowledge and generate the result artifact.

Software development environment, or computer aided software environment is used to support software

designer to design software based on the software requirement description. Its main responsibility is to provide representation of the software models and generated artifacts. It also displays the interface of tools in each of the development stages for user to interaction with. The generation, modification, and validation of the software artifacts, such as software specification, model or final program, can be done by manipulation of the interface elements of the tool in the environment. This kind of software tools or environment, as an intermediate tool for human to manipulate and interact with the computer artifacts, has high priority of requirement to be designed with the features of an interactive system.

The requirements for a software engineering tool, as an interactive system, are becoming broader than before. Not only the quality of the system, but also the usability aspect should be considered in the design of the software engineering tool. For usability issues, providing the people with an easy to understand, easy to use, seldom failure, and a friendly manipulation process can greatly increase the efficiency of the software development and decrease the risk of the project failure. The satisfactory of these properties are important for guarantee the operation of the system and the quality of the designed software.

A system that provides the user a satisfactory feeling of the operation and a satisfactory of the result product is called "dependable" system. Based on the definition of the "dependability" in [1], we extended the concept and defined a dependable interactive system as "a system on which the user's behavior can be performed with high reliability, and user can justifiably put reliance on certain aspects of the quality of service system delivers". By increasing the usability of the software development environment, therefore increasing the dependability, users can put more trust on the system operation, and have more endurance to the occasionally occurred software errors or failures.

In the software development process, the software requirement process has been proved to be the most human intensive process. Lots of human being is involved in the requirement elicitation, specification, and the analysis. The human participation will help the description of the requirement to be more complete, the formulization of the software requirement specification to be more accurate, the depiction of software functionality to be more consistent.

Among lots of approaches for the requirement acquiring and formulization, computer aided software prototyping has been proven to be an economic way to build a small-scale software system [8]. By building an executable version of the envisioned software, the clarified software requirements and the designed software models can be evaluated and adjusted before more efforts have been put into the detailed software implementation.

To ensure the quality of the produced software prototype and improve the prototype dependability, prototyping efforts need to be carried out with quality assurance methods. The correctness of the computing activity performance needs to be ensured. The effectiveness of the interactions between software development tools and human operators needs to be maintained and improved. The robustness of the performance of software design and prototyping tools needs to be sustained.

We had several year experiences and achievements on the computer aided prototyping research and application, the Computer Aided Prototyping System (CAPS) has been successfully used to build a software prototype automatically and clarify software requirements. We extend our foundational works and propose an expanded software prototyping framework based on the consideration of the human factors in the software prototyping process. To enhance the usability of the prototyping efforts, which therefore ensure the resulted prototype, our more research is concerning and focusing on the avoidance of the occurrence of the human errors and the recovery of the occurrence of human mistakes during the iterative prototyping process.

The prototyping framework developed here is an appropriate for the software development process that may include lots of human participations and collaborations. The iterative prototyping process provides the support for gradually accumulation of the software prototype models and the basis for the step-by-step refinement of the software requirement in terms of the feedback from the users about the performance of the software prototype.

The proposed several incorporated approaches, to at most extension, eliminate the potential risk of the error occurrence, is very useful and applicable to be integrated into a wide range of software development environment.

The paper is organized in five sections. In section 2, we briefly describe the related research work in the domain of computer aided software prototyping and human error. Section 3 introduces the framework of software prototyping with targeting at human erroneous efforts. Section 4 shows how we considered about how to resist the human erroneous actions. In this section, several design principles are described and we present the implementation of several approaches in the newly updated Computer Aided Prototyping System (CAPS-PC). In section 5, we draw our conclusions and discuss about future work.

2. Related Works

2.1. Computer Aided Prototyping

With increasing of the software complexity and scale, prototyping tends to be more than a simple set of techniques for software project development. It is called “prototyping methodology”. Several of activities are included in the prototyping methodology. They are modeling of prototype, evaluation of constraints, and automated program generation, which integrated as a whole to be the basis for prototyping methodology. When performing prototyping, several complex aspects are needed give more consideration with carefulness. To support the human prototype effort, a user-friendly prototyping tool is a prerequisite for larger use of prototyping base development [6][7][8].

As [2] indicated, the design and implementation of complex software-intensive system tend to product various errors at every development stages. For example, natural language is used to formulate requirement during requirements and design specifications in the early parts of the development. It would potentially result misunderstanding between users and designer of the future system and would produce errors that will have certain influence to reliability, safety, and cost of the system [8]. The software specification will possibly be interpreted in various ways, leading to “integration problem”[3].

Prototyping, as an economic way to build a small scale of software models and products, has been proved to be an efficient and effective methodology to evaluate proposed systems if acceptance by the customer or the feasibility of developments is in doubt [2].

2.2. Human Error

Since 1880, human error has been studied by lots of psychologists and biologists. It has shown that human error is the first source of damage of system

dependability. But the analysis of human error has always stayed in the statistical analysis and quality description phase. Study on how to apply the result of human error analysis into real system development is very important to the increasing requirement of system dependability, which has rarely been touched before.

2.2.1. Human Error Constraints in Software Development. Peters define human error in the following ways [13]: “any significant deviation from a previously established, required or expected standard of human performance”. Whereas, Sanders and McCormick [11], from the effect of human error point of view, gave a definition of human error as “an inappropriate or undesired human decision or behavior that reduces or has the potential for reducing effectiveness, safety, or system performance”. According to the definition of Sander, a standard for evaluation of action as human error is if this action has undesirable or potential effect on the system criteria, possibly on people’s safety. Also, Sanders thinks an error that is corrected before causing any trouble is still an error because it has the potential of causing an adverse effect on human or system criteria.

Johnson referred to three myths of human error that are often cited as barriers to the practical application of human error analysis [12]. Human error is described to be inevitable, unpredictable, and costly endurance. Following Johnson’s original proposed problems, we gave a further discussion and possible solution for each of them, especially for human activities in software development.

A. Human error is closely related with working situation; from the psychology point of view, the occurrence of human errors depends on the environment and the user’s understanding of situation, which provides the context for the user to derive from the supposed correct actions. Many researches have showed that, with correct and consistent understanding of the operation situation context, the human error can be prevented.

B. Human error can be predicted and removed by get enough knowledge about the interacted system; it is difficult to predict when and where a human will make an error. However, it is possible to predict and remove many of the local conditions that create the opportunity for user’s error to have disastrous consequences, such as reducing inattention and fatigue.

C. Human error can be guard and avoid by providing user satisfactory operation conditions; normally, it is too expensive for companies to employ the analysis and prevention techniques that reduce the human contribution to major accidents. But, a feasible and useful way to reduce the human error is trying to enhance human being’s understanding of system operation logic

and improve the system’s error endurance level. Some typical approaches for error endurance include the error prevention, fault tolerance and fault recovery.

2.2.2. Classification of Human Errors. Classification of human errors can help us to understand the human error within particular environments and special circumstances. Human error normally can be classified according to information processing and discrete actions.

Information processing is conducted coordinately during task performance. Some steps are performed to achieve the task, which include the observation of the system state, hypothesis formulation and testing, goal choosing, procedure selection for achieving the goal, and execution of the procedures. The information processing criteria for human error classification is often used in the analysis of complex systems.

According to classification criteria of discrete action, human error can be catalogued into error of omission, errors of commission, sequence error, and timing error. Error of omission means that the user forgets to do a task. Errors of commission imply performing an action incorrectly. Sequence errors include the errors of performing actions or tasks in the wrong order. Timing errors mean that the user fails to act in the allotted time period, due to performing either too slow or too fast.

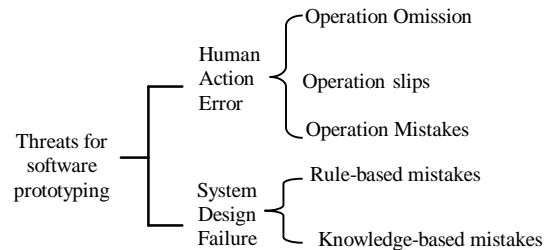


Figure 1. The classification of Human Threats for software prototype development

We define human threats to dependable software development in the context of the design and prototype efforts according to the effect of human discrete actions on the resulted software attributes. There are two types of source that would trigger the occurrence of errors. A human action error is an event where human action affects a part of the system state in a way that may cause a subsequent failure. Normally, the reaching of an error to the service interfaces that produces observable deviations from required service means the occurrence of a failure. A system design failure is an event that occurs when the system function service deviates from correct function service because of incorrect human action. A failure is a transition from correct service to incorrect service, such as not performing the required system function. These distinctions are illustrated in Fig. 1.

3. A software prototyping framework for supporting human design activities

Prototyping is a way of showing the idea behind a design in an inexpensive way. A prototype can be changed easily and the design evolved gradually to an optimal solution before the system is implemented. The idea behind prototyping is to save on the time and cost to develop something that can be tested with real users. Rapid prototyping has been found to be an effective technique for clarifying requirements. The early prototyping effort can provide the user a small-scaled software, which is more conceptual and understandable than that of the traditional software development approach which produce working code only near the end of the process.

Beside the simulation capability of the software prototype, we propose to adopt additional strategies for avoiding the context situation for error occurrence to improve the stability and robustness of the software. We proposed a framework for prototyping software with purpose to minimum the possibilities of error occurrence. Several human error avoidance methods and error recovery methods are designed, which can improve the user's understanding of software prototype and reduce the error of knowledge confusion.

Following our framework, an executable mini-system or a pilot version of an intended system will be built. The prototype can be treated as a partial representation of the target software, and can be used further as an aid in the design, analysis, validation, and verification of the system.

3.1. Prototyping Framework with Supporting of Error Elimination

Prototyping efforts are carried out from the step of "determine requirements", along the step of "construct prototype", to the step of "execute prototype". The resulted software executable prototype is the small scale of the designed software, which can be manipulated, tested, and verified. The real software can be built by performing the evolution of the approved software prototype. For each edges in this triangular diagram, several items are highlighted that act as the facilities to support the activities with duties of elimination of human errors or recovery of system derived state.

The original gathered natural language requirement are clarified and analyzed to provide a correct and consistent software descriptions for the prototype model formulization. By employing specification syntax error checking tool, the incorrect syntax structure can be modified to fit the software functional logics. By

checking the data consistency between the higher-level components description with lower level module description, the omission of software functionalities during the complex system decomposition will be prevented. By providing context aware design hints in the prototype tool, the user can get a clear picture of whole system structure and the timeline of what has been done and what has not.

From the construction of the prototype to the execution prototype, design history retracing is useful for the design backward when it is found that a design effort is fail or the revise of past design efforts. Error recovery is to restore the software status from the mistaken status to formerly verified correct status by retracing the past design history. Run time testing is an execution effort to make sure the result prototype is executable. Several errors would be found when the designer was trying to execute the prototype.

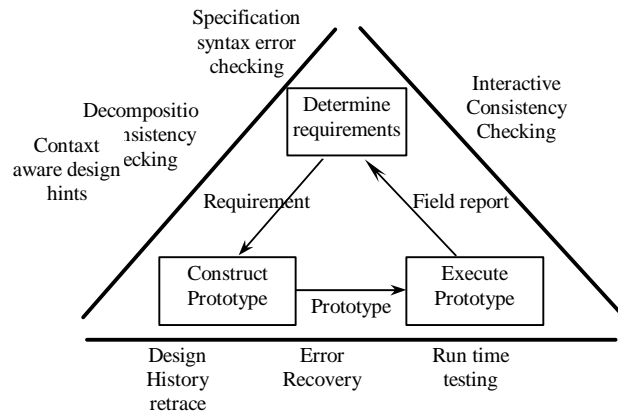


Figure 2. Framework for prototyping with supporting of error preventions

When the prototype is ensured to be executable, the checking of the execution functionalities of the software with the requirement description is very important for the end user's verification. Interactive consistency checking helps users to compare the performance of the prototype by the prototype interface with the imagined software in their brain to make sure what they got is what they want.

To enhance the capability of software prototyping and to make sure the generated prototype fit the requirement needs of the end user, we designed and implemented several methods to reduce the possibilities of the generation of the error, which therefore, will affect the quality of the resulted software. Some methods has implemented into the Computer Aided Prototyping System (CAPS-PC) developed in Naval Postgraduate School. Detailed descriptions and discussions for each method are presented in the following sections.

3.2. Multi-level Information Representation

The display of the too many information with classification and organization would reduce the efficiency of the information representation. The design model of the software prototype, although has been simplified, contains lots of information cohered with each designed module. To simplify the structure of the prototype model without the contained information, the software prototyping model can be displayed via multiple levels of view with different abstraction level and information hidden.

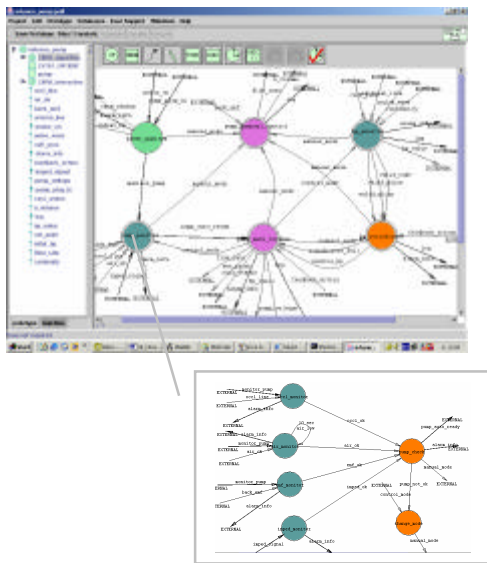


Figure 3. The decomposition of the second level of infusion pump prototype design

CAPS-PC provide multi-level point of view of the whole system design. The construction of the prototype begins with a top-level definition, followed with a number of sublevel refinement based on the functional decomposition. The high level of information illustration and lower level information decomposition prevent the designer from the information explosion and possible occurrence of design omission/slips. The designer can build the prototype with different granularity at different focus

3.3. Iterative Loop for Prototyping of Interactive System

The requirements are firmed up iteratively in a rapid prototyping approach through the examination of executable prototypes as well as by negotiations between

customer and designer. The designer constructs a prototype based on the requirements, and examines the execution of the prototype together with the customer and potential users of the system.

The refinement of requirement is fulfilled by two kinds of approaches. From one hand, the requirements are adjusted based on the feedback from the customer, and the prototype is modified consequently until both the customer and the designer agree on the requirement. On the other hand, during the execution of the prototype, possible user's action can be generated to simulate human fault, which serves as an input to the system functionality or an anticipated feedback for a system function of an acquirement. Both of two methods are helpful to find and locate the software functional errors, on which further modifications can be carried.

4. User-centered Designs with Human Factor Considerations

An integrated software prototyping environment should not only have strong capabilities to building prototype, but also represent high support for usability of the prototyping effort. It is expected to improve the quality of the systems developer, reduce the time and cost of software development, enhance the developer's satisfactory and productivity, and in most extent make the software development a more delight and exciting task [11, 13]. Ease-of-use (usability) and ease-of-learning (Learnability) are two highlighted topic in the design and implement of an integrated development environment.

By considering the human factor in the design of CAPS-PC, we help the CAPS-PC user to correctly understand the functionalities carried on during the software prototyping. Five principles recommended in [11] are implemented in the design of CAPS-PC:

1) Making 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.

2) Minimizing the developer's memory load. Users of CAPS-PC don't need to remember functionalities and program components from one particular dialog or screen when they move to another area. The related artifacts and functionalities can be easily retrieved and made visible whenever they are needed

3) Speaking the developer's language. Although CAPS-PC used LAMPS as its bottom line language, which is a formal specification language as description in section 2, it is easy for user to understand the description of the control components and their supposed triggered action. The words, phrases and concepts displayed in the

CAPS-PC interfaces are illustrated by using common development language, which is comfortable and easy for developer to get acknowledge and familiar with.

4) Keeping the developer informed on the CAPS-PC status and the prototype being designed. During the prototype development process, contextualized feedback and appropriate message are displayed to developer at the time of happening, which efficiently inform the developer about the system status and the hints for possible consequential results.

5) Preventing developers from making mistakes. For a development environment, it would be an effective facility to prevent the problem from occurring in the first place. CAPS-PC used human error resistance strategies to prevent or limit the consequence of human error. Error resistance can be achieved by means of two strategies: error prevention and error handling. Error prevention can result from forcing functions, operator's selection, or training, which will not get rid of all human error occurrences. Then error handling mean to catch the error in the case of error occurrence, and control the evolution of error to minimize the error effect. To increase the usability of specifying and designing of system requirement model, CAPS-PC employed a backward error recovery strategy to reset the system state when an error or mistake occurs. Also, keeping of design histories and making them retraceable can remind the prototype developer the process of their past design steps, which can provide helpful information for developer to enforce most effective recovering approach.

4.1. Design History Retrace and Error Recovery

Backward error recovery is an attempt to restore the system state after an error has occurred. Backward recovery can be considered as the only real "recovery" function. The unexpected effects of errors will be totally removed with step by step backward until to the satisfied state.

Three kinds of backward error recovery commands, undo, cancel and stop, are implied in our prototyping tool, CAPS-PC.

Undo mechanism help user to go back the history of editing of the prototype specification and definitions of the designed software modules, interconnections, and their required properties.

The cancel function is used to abandon commands under specification. For example, a user can cancel the typing of a timing constraints property in a time critical module if it is checked to conflict with other timing constraints. Cancel function always deal with in special situation that the user must know which command are concerned by the recovery.

The stop function is used to terminate a particular execution process. For example, in CAPS-PC, with the completed prototype model, the translation process can be triggered which will translate the prototype specification to generate pre-designed system skeleton and the control relationship between the modules. Once the process is trigger, the output of the execution will be displayed with the ongoing translation process. In the case of finding some mistakes before the process finished, users are always want to make modification right now. It will be necessary to stop the execution process and restore the initial state before translation.

CAPS-PC provides flexible mechanism to support the error recovery on time to prevent the effect of error. By employ multi-thread mechanism, translation, schedule, and compilation of prototype are triggered separately from the main editing of prototype graphic. During runtime execution of program, problems can be found and provided to user directly.

4.2. Automatic Syntax Error Checking

The grammar monitor for software requirement specification provides an excellent debugging support. Before saving the specification description, the syntax can be verified. In the case of syntax error, the grammar error information is displayed and selection suggestion is given on the interface for the user. This enhances the quality of the prototype produced.

When saving the requirement specification, the data flow diagram of the prototype will be checked to ensure that the data flow diagram has a correct input-output structure. Structural error messages are displayed to the user if the input of one module is fully used in its module functionality. An instant checking during each user's input is implemented in the prototype tool, CAPS-PC, to prevent human's operation omission/slips.

4.3. Interaction Consistency Check

During the design of software model, the interaction between the modules will be defined. The relationships between the modules include their data communication, input constraints and output constraints. All of this information would be defined in the connection-edge annotation. In the prototype specification annotation, edges are allowed to have the same name in the same level or in different levels of the tree structure. All the edges with the same name are regards as the same data stream and have the same property. The edge inconsistency problem may occur when the properties of the prototype edge will be changed in one of its instance [12]. We provide automatic consistency maintenance to

check the differences between the interaction edge's name, data type, and related time properties. The edges inconsistency checking include:

1) Dynamically detects the edges' name changed and automatically fill in the edge property if it has the same name with an existing edge;

2) When any edge's property changed and the OK button was clicked, searching all the edges existed to find every edge with the same name but different properties.

Once the prototype tool find the inconsistency between edges, an inconsistency table will display the detail information about the conflict edges, and provide the evaluation hints for user to select a correct property, which is showed in figure 4. When a verified property definition item is selected by the user, all the other edges that are list on the inconsistency table will be changed to be consistent with the verified edges.

4.4. Decomposition Consistency Check

Based on the hierarchical design of software model, the high level model can be decomposed into low level model. The functionalities and properties can be detailed and realized in the decomposed second level. During the decomposition of software model, the high-level component definition should be consistent with lower level modules. Two types of consistencies should be maintained:

1) The number of data input/output of high-level component definition should be consistent with lower level data input/output. All the data input/output defined in high level should be realized into lower level as edges from or to terminal.

2) The properties of component defined in high level should be consistent with lower level properties definition. If the high level component For example, a data manipulation component is defined to have a periodic time constraints, then the lower level component event should also be triggered periodically. If Maximum Execution Time (MET) of the high level component is set to be 500ms, then the triggering condition of the low level component should be within the MET constraints, which means its maximum execution time should be less than 500ms.

4.5. Context Aware Design Hint

During the design of system, we provide several facilities to let the user know the current status of system and the coverage of designed prototype. Give more

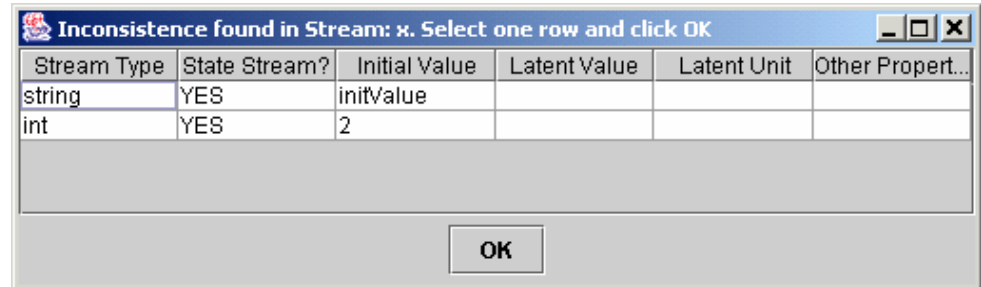


Figure 4. A checkable table for edge inconsistency properties

information for user to keep tracing of the design process will reduce the interruption in the design process. Contextualized feedback and appropriate message are displayed to developer at the time of happening, which efficiently inform the developer about the system status and the hints for possible consequential results.

During the design of software prototype, following information are keep updated and displayed to inform the developer:

- 1) The name of current prototype project and its version;
- 2) The current decomposition component and its located level;
- 3) The necessary to perform the saving of prototype according to the prototype modification actions every after saving the designed prototype;
- 4) Once the focus of mouse is put on a particular menu/icon/button, the related annotation of the interface component will be displayed beside the mouse focus. Its supposed action will be interpreted at the bottom of the interface of the prototype tool.

5. Conclusion

Prototyping is not recognized as a cornerstone of the successful construction of software system as it allows making users at the center of the development process. However, with the involvement of the human, prototyping tends to produce low quality software as no human factors or error resistance is undertaken. We have shown in this paper how methods of error checking and recovery can be integrated into the software prototyping framework and can contribute to the development process of software system through prototyping activities.

While the techniques for prototyping, such as specification, design, and code implementation, has

reached a maturity level allowing moving with real application, the prototype tool with human factor consideration is still under development. A real application has been completely in the field of the CARA [19].

In addition, it has also shown the amount of work that is still required before the environment can be used by wider range of people. Several features is promising to be integrated into our prototype framework such as:

- 1) integration of property verification to ensure the inside accuracy of prototyped component functionality,
- 2) performance analysis in order to support more flexible technique selection, and
- 3) experience recovery with more widely time duration and task scope.

6. References

- [1] J. C. Laprie. Dependable computing and fault tolerance: concepts and terminology. In Fault Tolerance Computing Symposium 15, pp. 2-11, Ann Arbor, MI, IEEE Computer Society, June 1985.
- [2] Kutar, M., Britton, C., & Nehaniv, C. Specifying multiple time granularities in interactive systems. In Proceedings of Design, Specification and Verification of Interactive Systems Workshop, pp. 49-61. 2000
- [3] James Reason, Human Error, Cambridge University Press, 1990
- [4] Luqi, Valdis Berzins, Raymond T. Yeh, A Prototyping Language for Real-Time Software, IEEE Transaction on Software Engineering. Vol. 14, No. 10, Oct. 1988
- [5] Rapid Evaluation of Interactive Systems, LTI-Annual Report 1999/00
- [6] Nico Hamacher, Jory Marrenbach, Karl-Friedrich Kraiss, Formal Usability Evaluation of Interactive Systems, 8th IFAC/IFORS/IEA Symposium on Analysis, Design, and Evaluation of Human Machine System 2001, Sept. 18-20, Kassel, VDI/VDE-GMA, pp577-581, 2001
- [7] Luqi, Real-Time Constraints in a Rapid Prototyping Language, Computer Language, Vol. 18, No.2, pp.77-103, 1993
- [8] Luqi, Mohammad Ketabchi, A Computer-Aided Prototyping System, IEEE Software, pp.66-72, March, 1988
- [9] Pete Broadwell, Naveen Sastry, Jonathan Traupman, FIG: A Prototype Tool for Online Verification of Recovery Mechanisms, ICS SHAMAN Workshop'02 New York, USA, 2002
- [10] Sanders, M.S. & McCormick, E.J. Human Factors in Engineering and Design, 7th Ed., 1993, New York, McGraw-Hill
- [11] Chris Johnson, Why Human Error Analysis Fails to Support Systems Development, Vol. 11, No. 5, Interacting with Computers, pp.517-524, 1999
- [12] Peters G. Human Error: Analysis and Control, Journal of the ASSE, 1966.
- [13] Sarter, N.B. & Woods, D. D., Strong, silent, and 'out-of-the-loop'. CSEL Report 95-TR-01, February, 1995
- [14] Gail E. Kaiser, Cooperative Transactions for Multi-User Environments, Won Kim (ed.) Modern Database Management: Object-Oriented and Multidatabase Technologies, ACM Press, 1994
- [15] Smith, D. E., & Marshall, S. P., Applying hybrid models of cognition in decision aids. In Zsombok, C. & Klein, G. (Eds.), Naturalistic Decision Making, Mahwah, NJ: Erlbaum, pp. 331-343, 1997
- [16] David Navarre, Philippe Palanque, Remi Bastide, & Ousmane, A model-based tool for Interactive Prototyping of Highly Interactive Applications, pp 136-141, 2001
- [17] Sanders, M and McCormick, E Human Factors in Engineering and Design (six edition) McGraw-Hill, New York, NY, USA , 1987
- [18] Luqi, Ying Q., Lin Z., Computational Model for High-Confidence Embedded System Development, Monterey Workshop 2002, Italy
- [19] Luqi, Z. Guan, Requirements Document Based Prototyping of CARA software, to appear at International Journal on Software Tools for Technology Transfer