2002-12

High-confidence Development and Evolution for System of Embedded Systems

Luqi; Qiao, Y.; Luqi; Qiao, Y.

Naval Postgraduate School

Luqi and Y. Qiao, "High-confidence Development and Evolution for System of Embedded Systems", Technical Report, NPS-SW-015, December 2002.
https://hdl.handle.net/10945/65206

NPS-SW-02-015

NAVAL POSTGRADUATE SCHOOL
Monterey, California

High-confidence Development and Evolution for Systems of Embedded Systems

Technical Report

By

Luqi, Ying Qiao

December 2002

Prepared for: NSF

PROJECT SUMMARY

The objectives of the proposed research is to present a set of systematic technologies to accomplish high-confidence development and evolution of systems of embedded systems (SoES). These systematic technologies include the study of quantitative measures for high confidence of SoES, the modeling of SoES with high-confidence properties, SoES evolution method and robust high confidence analysis for SoES prototype systems.

High confidence is a perception of how a system should behave. It is critical for embedded systems since they are widely used in the fields in which the consequences of systems failure are very serious. Systems of embedded systems (SoES) have been receiving a great amount of attention because of their capability of combining individual embedded systems to accomplish broad and common objectives. Although many approaches have been proposed to construct high-confidence embedded systems, most of them are proposed for the development of monolithic embedded systems. Constructing high-confidence SoES is still a great challenge. Further efforts on studying novel systematic technology for the development of SoES are needed.

As the start point, in this proposed research, we plan to construct the quantitative measure for high confidence of SoES. For this purpose, we propose to present the quantitative metrics for high confidence of SoES and suppose to derive the high confidence measure basing on this metrics. Moreover, we will find the design parameters that have the impact on the high confidence metrics for SoES. By these efforts, the relationship between the high confidence requirements of SoES and the design parameters will be established so that the high confidence can be ensured during the design phase.

Basing on the study of distinguish properties of SoES and the quantitative measure for high confidence of SoES, we propose to develop a computational model for SoES so that it can describe the SoES in the real-world context and the high-confidence properties can be captured. This computational model has two views, i.e., external view and internal view. The external view model focuses on the requirements view and describes the requirements of SoES by representing its emergent properties while the internal view model focuses on the design view and decomposes the emergent properties into the constraints imposed on individual component systems and interactions between component systems for design phase. The computational model is the key point for the systematic development of SoES. It has three purposes: a) generating high-confidence prototyping code for SoES; b) generating code for run-time monitoring during the reliability assessment and testing; c) enabling possible static verification for some metrics.

Moreover, software evolution must be considered during the system design process since the requirements of systems of embedded systems typically change in response to changes in the real-world environments of these systems. However, software evolution is a complicated process in systems of embedded systems because it is difficult to maintain the consistency of constraints during software evolution in such a complex system. To solve the above problems, in this proposed research, we propose to study the evolutional model for SoES from the structure point of view and the behavior point of view. This model builds the basis for evolution of high-confidence SoES. Furthermore, in response to the change of circumstance, how to keep the high-confidence properties during the evolution process is also studied.

After the computational model and evolution model are developed, we propose to provide the description languages for them. The description language for the computational model is proposed to be an extension of current PSDL. And we also propose to develop a description language, called SEDL (software evolution description language), to describe the evolution model for systems of embedded systems.

To manage the requirements, robust high confidence analysis (RDA) is studied. By this method, the impact of changing requirement on high confidence issues is revealed so that it can support changing requirements without compromising high confidence.

In conclusion, these systematic development technologies provide an efficient way to construct the SoES basing on a computational model so that the high-confidence properties of SoES can be certified during the development process.

PROJECT DESCRIPTION

1 Objectives and Significance

The main objective of the proposed work is to develop a set of systematic methods to accomplish the development of high-confidence systems of embedded systems (SoES). These systematic technologies include the study of quantitative measures for high confidence of SoES, the modeling of SoES with high-confidence properties, SoES evolution method and robust high confidence analysis for SoES prototype systems.

High confidence is a perception of how a system should behave. In high-confidence systems, the sponsor, designer, implementers and users have a high degree of assurance that the system will not fail or misbehave due to errors in the system, faults in the environment or hostile attempts to comprise the system. This is critical for embedded systems since they have been widely used in many fields such as flight control and avionics, processing control, weapon system control and nuclear plant control, in which the consequences of failure are very serious.

Recently, requirements for combining individual embedded systems to accomplish broad, common objectives have been receiving greatly increased amounts of attention because of large investments in existing isolated embedded systems. Therefore, systems of embedded systems emerge as the promising solution for such urgent requirements. The typical examples include National Aviation Systems (planes, airports, airways, air traffic control), Naval Mine Countermeasures Force (search, sweep, neutralization systems), Naval Surface Fire Support (reconnaissance, targeting, weapon systems), and Theater Ballistic Missile Defense (surveillance, tracking, interceptor systems). Although a lot of works have been done to develop the individual embedded systems, constructing high-confidence SoES is still a great challenge. This is because the individual component systems are independently developed and therefore typically run on different platforms and have different data formats and interaction protocols. Moreover, since the requirements of SoES typically change in order to respond to changes in the real-world environments, evolution in SoES is unavoidable. However, evolution in SoES is much more complicated than in general embedded systems and it is difficult to keep consistency of high-confidence properties during evolution process. Moreover, increasing complexity of SoES increases the uncertainty of requirements in SoES. Failure to deal with this uncertainty will result in software faults.

Thus, novel technologies to construct high-confidence SoES are impendent to overcome these challenges in an effective and sound way. For this purpose, we propose to present systematic development technologies that provide an efficient way to construct the SoES basing on a computational model so that the high-confidence properties of SoES can be certified during the development process.

Five research issues will be attacked in this proposed work:

(1) Quantitative measure for high-confidence of SoES
(2) Computational model for high-confidence SoES
(3) Evolution process for high-confidence SoES
(4) Description language for the computational model of high-confidence SoES
(5) Robust high confidence analysis for SoES prototype systems

1.1 Significance and Benefits

The proposed systematic technologies will enable the following to occur:

(1) High confidence properties become visible during the whole development process so that it can be ensured during the whole development process. This will enhance the high confidence of SoES.
(2) Reliable software developed by construction
(3) Avoid the waste of effort and expense on infeasible requirements
(4) The lag time between discovery of new requirements and building SoES software to meet those requirements can be reduced

(5) Enable high-confidence to be maintained during the evolution of SoES

1.2 Technical Barriers

SoES should be distinguished from large and complex but monolithic embedded systems by the independence of their components, their evolutionary nature, emergent behaviors and a geographic distribution. Constructing a high-confidence SoES is a great challenge. This is because the individual component systems are developed independently so that these component systems are developed to run on different platforms and have different data formats and interaction protocols. However, current approaches to construct high-confidence embedded systems mainly focus on the development of monolithic embedded systems rather than systems of embedded systems (SoES). Since SoES have the distinguished characteristics compared to the monolithic embedded systems, current studies in the development high-confidence embedded systems have following limitations:

1. They can not provide a model to describe SoES, especially their distinguished characteristics.
2. Change of circumstance results in the change of requirement for SoES. In order to respond to these changes, the evolution should be considered. Furthermore, the evolution of SoES is much more complex than for monolithic software systems because of the heterogeneity of component systems in SoES. However, current approaches to deal with software evolution apply to monolithic embedded systems. This implies that we should develop efficient methods to cope with the evolution in SoES.
3. Many approaches assume the existence of an accurate and valid formal specification of the properties that are to hold with high confidence. However, the last thirty years of computing research have shown that this assumption does not hold in practice, and that a majority of software faults can be attributed to requirements and specification errors. Although prototyping has emerged as a preferred method for requirements validation, current approaches do not support the prototyping of SoES.

2 Technical Approaches

The key point of systematic development technologies is to provide a computational model for SoES with high-confidence properties. This model has three purposes: a) generating high-confidence prototyping code for SoES; b) generating code for run-time monitoring during the reliability assessment and testing; c) enable possible static verification for some metrics. Furthermore, robustness analysis is explored for the prototype so that it is ensured that the prototype can capture the high-confidence properties under a bounded uncertainty. Also, in response to the change of circumstance, how to keep the high-confidence properties during the evolution process is studied in the proposed research.

2.1 High-confidence Measure for SoES

To construct high-confidence SoES and present the computational model of SoES with high-confidence properties, it is necessary to build the relationship between the high confidence requirements of SoES and the design parameters so that the high confidence can be ensured during the design phase. Thus, we should study the high-confidence measure first. For this purpose, in this research issue, we should solve the following problems: 1) Find the relevant quantitative metrics for high confidence of SoES; 2) Find the design parameters that have the impact on the high confidence metrics for SoES; 3) Construct quantitative measure for high confidence of SoES.

High confidence is a perception of how a system should behave. In high-confidence systems, the user designer, implementers and users have a high degree of assurance that the system will not fail or misbehave due to errors in the system, faults in the environment or hostile attempts to comprise the system. Furthermore, high confidence can only be measured by measurable attributes. Some attributes for high-confidence embedded systems are availability reliability, safety, security, integrity, robustness and maintainability etc. The definitions of these attributes are follows:

Availability:	The property of readiness for correct service;

Reliability: The property of continuity of correct service; Reliability is denoted by *R* (*t*), which is the probability that no failure occurs in the time period [0, *t*]

Safety: The property of non-occurrence of catastrophic consequences due to a software failure.

Security: The property of absence of unauthorized access to, or handling of, system state;

Integrity: The property of non-occurrence of the improper system state alterations

Robustness: The property of maintaining rationality in an unexpected environment. In fact, robustness is high confidence with respect to erroneous inputs

Maintainability: The ability to undergo repairs and modifications and improve and revise the system in a changing environment.

In this case, we will use the attributes contributing to high confidence to construct quantitative metrics for high confidence of SoES. Thus, we firstly introduce the high confidence vector that reflects high confidence attributes involved in the specific applications. This high confidence vector is represented as follows: $H = (h_1, h_2, \ldots, h_z)$, where $h_i (i \in [1, z])$ is the set of attributes for high confidence. There are one or more corresponding metrics for each attribute. The typical metrics for high confidence are availability, reliability, maximum time between safety violations and security level. Some more quantitative metrics should be further developed.

Basing on the high confidence vector, we propose to develop a high confidence measure for SoES. This measure can be expressed as a function $g(\ )$ of the high confidence vector. Thus, the high confidence measure H_m for SoES can be represented as follows: $H_m = g(H) = g(h_1, h_2, \ldots, h_z)$.

The key point to derive the high confidence measure is to decide the threshold values of each attributes and the function $g(\ )$. For this purpose, we firstly define a safe range for any individual attribute of high confidence as the interval [*a*, *b*] of two real numbers *a* and *b* within which the value for that attribute is acceptable. Furthermore, we intend to find empirically a safe range of values for each of the h_i parameters and then to formulate the function $g(\ )$ based on this information and the result of the experiments. The SoES remains high confidence if all the individual attributes for high confidence are maintained within respective safe ranges. The threshold values of the attributes are decided by the empirical values that depend on the properties of the applications. For example, availability is the most important contribution for the high confidence in telecommunication systems; reliability should be mostly emphasized in space probe systems and safety is the first-class factor to affect high confidence for the on-board control in transportation systems, etc. Thus, the threshold value of the attribute increases with the importance of the attribution's contribution for the high confidence of SoES.

To map the high confidence measure for SoES into the design parameters, we should firstly develop a methodology to quantitatively express each attribute that contributes towards the high confidence of SoES. Although there has already been considerable research effort to effectively measure some of the high confidence attributes, we propose to extend the research to cover design aspects of high-confidence computing. Following that, we plan to analyze the quantitative expression of the high confidence attributes to recognize the factors that impact the high confidence. According to these factors, we can use empirical experiments or theoretic analysis such as theorem proof to establish the relationship between the design parameters and the factors that impact the high confidence of the SoES.

2.2 Computational Model for Embedded Systems of Systems (SoES)

To construct high-confidence SoES, it is necessary to explore a mathematical model that is close to the designer's view of SoES. Thus, this proposed research plans to study a computational model for SoES so that it can be use to a) generate high-confidence prototyping code; b) generate code for run-time monitoring during the reliability assessment and testing and c) perform possible static verification for some metrics. Moreover, this computational model proposes to satisfy the follows:

(1) Representing the requirements for whole SoES, especially capturing the required high-confidence properties of SoES.
(2) Reflecting the static structure and dynamic behavior of SoES.
(3) Revealing the hierarchical design process for SoES.

2.2.1 Framework of computational model for SoES

The computational model supported by PSDL contains OPERTATORS that communicate via DATA STREAMS[11]. Each data stream carries values of a fixed abstract data type. Although this computational model can describe monolithic and centralized embedded systems and reflect some real-time features of embedded systems, it can not describe distinguished properties of SoES and also can not capture the high confidence properties of SoES. Thus, we propose to extend the computational model supported by traditional PSDL to the case of SoES, especially introducing some high-confidence constraints to depict the high confidence requirements.

In order to satisfy three demands above, we propose that the framework of this computational model have two views --- external view and internal view. The external view model focuses on the requirements view and describes the requirements of SoES by representing its emergent properties. On the other hand, the internal view model focuses on the design view. It decomposes the emergent properties into the constraints imposed on individual component systems and interactions between component systems for design phase. In this case, emergent properties refer to the properties of entire SoES that can not be localized in any individual embedded system. They characterize the emergent behaviors [16] that arise out of the interactions of individual embedded systems and represent the value added by the interaction, and are typically not satisfied when the component systems are all operated as isolated systems. We plan to formally represent the external view model as follows:

$$\zeta' = (G, H)$$

G is a functional emergent property vector which represents the functional requirements of SoES such as timing properties while H denotes a non-functional emergent property vector which reveals the non-functional requirements of SoES, especially the high-confidence aspect. The detailed information of H is discussed in 2.1. Since G and H describe the requirement for whole SoES, the external view model contributes to establish the SoES prototype. Consequently, requirements will be validated by executing the prototype system.

The internal view model is derived by mapping emergent properties into local constraints on component system set S and interaction sets E. Thus, the internal view computational model can be formally described as below:

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

S is the component system set, $S = \{s_i \mid i \in [1,n]\}$; s_i denotes the component system constituting SoES (n is the number of component systems in the whole SoES); E denotes the interaction sets between component systems, $E = \{e_{jk} \mid j,k \in [1,n]\}$, Where e_{jk} denotes the set of interactions from component system s_j to component system s_k; e_{jk} is a set of Γ_{jk} elements, $\Gamma_{jk} \geq 0$, $e_{jk} = \{e_{jk}^{\gamma} \mid \gamma \in [1, \Gamma_{jk}]\}$. C denotes constraint sets on how the component systems are used in the given environment, $C = \{c_i \mid i \in [1,n]\}$. Furthermore, c_i is a set of constraints with P_i elements, $P_i \geq 0$, $c_i = \{c_i^p \mid p \in [1, P_i]\}$; c_i^p denotes the pth constraint for s_i. D denotes constraint sets on interactions between component systems, $D = \{d_{jk} \mid j,k \in [1,n]\}$, where d_{jk} is a set of constraints with Q_{jk} elements each of which applies to interactions in e_{jk}, $Q_{jk} \geq 0$, $d_{jk} = \{d_{jk}^q \mid q \in [1, Q_{jk}]\}$, d_{jk}^q denotes the qth constraint for e_{jk}.

Constraint sets C and D are determined by emergent properties in external view, $C = F_1(G,H)$; $D = F_2(G,H)$, where F_1 and F_2 are two maps that map emergent properties into local constraint sets on component systems and local constraint sets on interactions between component systems respectively. However, how to decide F_1 and F_2 will be further studied in this proposed research.

In the internal view computational model, s_i and e_{jk} describe the static structures of SoES while c_i and d_{jk} describe the dynamic behavior of SoES. Furthermore, the mappings F_1 and F_2 specify what must be assessed to ensure that the SoES 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 an abstract design for the systems integration, which will be concretely realized by wrappers around the s_i.

2.2.2 Component systems

Component systems are a set of independent complex embedded systems. As a matter of fact, a component system is a slot rather than a single fixed subsystem. The slot can be filled by any subsystem that meets the constraints specified at the level above. The analysis that leads to the assurance of high confidence should depend only on the constraints associated with the slots. In this way, if we have a set of concrete subsystems for each slot that have been certified to meet the constraints associated with that slot and vary systematically with respect to other properties, then we can reconfigure SoES by plugging in different certified subsystems into corresponding slots without need for further re-certification for each reconfiguration. Such a structure is needed to support rapid and low cost evolution without compromising high confidence. One of the purposes of the proposed extension to the computational model is to support such a structure.

In this computational model, each component system is either atomic or composite. Composite component systems can be described by interaction-link networks of lower level component systems while atomic component systems can not be decomposed in terms of the SoES computational model.

In order to represent the hierarchical structure of the composite component system, we introduce the *layer* for each component system. Each composite component system in *a higher layer* is composed of several sub-component systems in *the next lower layer*. Based on *layers*, we plan to describe the hierarchical structure of the composite component system by giving the description of decomposing a high layer component system into several lower layer component systems from the external view and internal view. Furthermore, during this decomposition process, the functional and non-functional emergent properties in higher layer component systems should be mapped into the constraints on component systems and interactions between component systems in the lower layer. In this context, high layer functional emergent properties can be used for (a) generating code for monitoring failure events during the reliability assessment and testing and (b) verification. Non-functional emergent properties related to high confidence in high layer can be used for (a) experimental assessment of high-confidence attributes and (b) possible static verification for some metrics, such as Maximum Execution Time (MET).

Moreover, we also need to study how to model component systems themselves. In fact, each component system has three parts --- functional part, interaction part and interface part. We plan to use the operator model in traditional PSDL to describe the functional part so that it can be represented as a functional abstraction or a state machine abstraction with a thread of control. For the interaction part, it will be described by a set of *ports*. Each port is related to an interaction and is responsible to contain the interaction information involved in this interaction. The property of the port depends on the property of involved interaction information. Furthermore, interface part is described by the wrapper which is a software layer around the component system.

2.2.3 Interactions between component systems

Interactions between component systems are complex especially in the case of SoES. There are two types of interaction between two specific component systems, i.e., directed interaction and non-directed interaction. In this proposed research, we will study how to describe and analyze these two kinds of interaction in SoES.

• *Directional Interaction:*
In directional interaction, there are explicit source component system and destination component system. *Source* component system is responsible to send or generate the information and *destination* component system is responsible to receive or response to the information. We propose to describe the directional interaction by interaction style, interaction information involved in the interaction and related connector for the interaction.

Since SoES is composed of heterogeneous component systems, there exist several types of interaction styles in SoES. In this proposed research, we should describe some typical interaction styles such as pipeline, event invocation, explicit invocation and inter-distribution [17,18]. Furthermore, each interaction style has corresponding interaction information. Four types of interaction information supposed to be supported by this computational model are data stream, event stream, parameter and data gram. This enables not only static interactions but also dynamic interactions to be described so that makes the computational model more closer to the real-world SoES.

Another problem is the decomposition of the directional interaction. Since the component systems is either composite or atomic, there are four types of directional interactions, i.e., the interaction between two composite component systems, the interaction from composite component systems to atomic component systems, the interaction from atomic component systems to composite component systems, and the interaction between two atomic component systems. Therefore, in this computational model, we should model hierarchical decomposition of the interaction to support the different kinds of directional interaction from the structure point of view.

• *Non-directional Interaction:*
In non-directed interaction, there is no distinction between *source* and *destination* component systems. All the component systems involved in the interaction are counterparts. They interacts each other not by the explicit interaction information but by accessing to the shared resource. Non-directional interaction is very popular in SoES, especially when the system is very complex. Since non-directional interaction will result in difficulties to determine the timing constraints for the component systems, it is necessary for us to study how to describe it. The important aspects of non-directional interaction are shared resources and strategy that is used to decide which component system has authority to access the exclusive resource. Therefore, we propose to use these two factors to describe a non-directional interaction in computational model.

2.2.4 Constraints on component systems and interaction between component systems

In the framework, low level constraint sets C imposed on component systems and constraint sets D imposed on interactions between component systems are refined from high-level emergent property vector G and high confidence constraints vector H by mappings F_1 and F_2. Accordingly, the constraints imposed on component systems and interactions between component systems can be divided into two catalogs, i.e., the constraints for the real-time features of SoES and the constraints for high-confidence features of SoES.

• *Constraints for real-time features*
In case of constraints for real-time features, a lot of works have been done for timing constraints and control constraints in the computational model supported by PSDL. Thus, in this proposed research, we propose to extend these timing constraints and control constraints to the case of SoES. For contral constraints, like in individual embedded systems, whether the component system is *PERIODIC* or *SPORADIC*, triggering methods, triggering conditions, and output guards are the major aspects to be specified.

Timing constraints are also critical for SoES. In this case, we can make use of the timing constraints specified in the computational model supported by PSDL [11]. In this case, *Maximum Execution Time* is one of the typical timing constraints for SoES. It can be imposed on the component system. Furthermore, for the periodic component system, the timing constraints include *Period* and *Finish Within*. For the sporadic component system, the timing constraints include *Maximum Response Time* and the *Minimum Calling Period*. Moreover, *Latency* that is defined in PSDL can be taken as the timing constraint imposed on the interactions between component systems.

Furthermore, we propose to study more constraints for real-time feature besides the real-time constraints and control constraints identified above, as needed to adequately model and design SoES. One typical example is synchronization constraints. They are very important for embedded systems, especially for SoES which is interaction extensive. In this case, synchronization in SoES refers to the controlled access to the shared resource among the sub-component systems. Further studies are needed to present some detailed synchronization constraints.

• *Constraints for high-confidence features*

To this proposed research, we plan to study some constraints for high-confidence SoES. In the framework, the emergent properties related to high confidence are included in the external view model. Refined from these emergent properties, some constraints for high-confidence features are derived on the high-level of internal model, and consequently are localized to the lower level. In fact, these constraints reflect the design parameters for SoES.

At present, since reliability is a one of the most typical emergent property related to the high-confidence and a lot of research works have been done on this aspect [15,29], we take it as the start point and plan to refine reliability into some constraints of fault tolerance mechanism so that the high level emergent property --- reliability can be mapped into the constraints during the design phase.

2.3 Evolution model for systems of embedded systems

The requirements of SoES should be changeable to accommodate changing circumstances. Thus, software evolution must be considered during the system design process. However, compared to other aspects of software development, the evolution of software systems accounts for bulk of their cost. Since component systems of SoES are developed one by one and originally in different architecture, the evolution of SoES is much more complex than for general software systems. The complexity of SoES makes it difficult to keep the consistency of the high-confidence property during the evolution process. In this research work, we propose to describe the process of evolution of SoES from the structure point of view and the behavior point of view.

The following model is presented to describe the evolution of SoES from the structure point of view:

$$\zeta^{v+1} = \zeta^{v} + \Delta\zeta^{v}$$
$$\Delta\zeta^{v} = \Delta S^{v} + \Delta E^{v} + \Delta C^{v} + \Delta D^{v}$$
$$\Delta S^{v} = \sum_{i\in[1,n]} \Delta s_i^{v} + \sum_{i\in(n,m]} s_i^{v}$$
$$\Delta E^{v} = \sum_{j,k\in[1,n]} \Delta e_{jk}^{v} + \sum_{j\in[1,n],k\in(n,m]} e_{jk}^{v} + \sum_{k\in[1,n],j\in(n,m]} e_{jk}^{v} + \sum_{j,k\in(n,m]} e_{jk}^{v}$$
$$\Delta C^{v} = \Sigma_{i\in[1,n]} \Delta c_i^{v} + \Sigma_{i\in(n,m]} c_i^{v}$$
$$\Delta D^{v} = \Sigma_{j,k\in[1,n]} \Delta d_{jk}^{v} + \Sigma_{j\in[1,n],k\in(n,m]} d_{jk}^{v} + \sum_{k\in[1,n],j\in(n,m]} d_{jk}^{v} + \Sigma_{j,k\in(n,m]} d_{jk}^{v}$$

Where $m = n + \Delta n$, Δn refers to the number of new component systems; v is the software version number of SoES; $\Delta s_i^{v}(i \in [1,n])$ denotes the modification of each exist component system; $s_i^{v}(i \in (n,m])$ denotes the new component system added into SoES; $\Delta e_{jk}^{v}(j,k \in [1,n])$ denotes the modification of each exist interaction; $e_{jk}^{v}(j \in [1,n], k \in (n,m])$, $e_{jk}^{v}(k \in [1,n], j \in (n,m])$ and $e_{jk}^{v}(j,k \in (n,m])$ denote the new interaction added into SoES; $\Delta c_i^{v}(i \in [1,n])$ denotes the modification of each exist constraint set on the component system; $c_i^{v}(i \in (n,m])$ denotes the new constraint set on the component system added into SoES; $\Delta d_{jk}^{v}(j,k \in [1,n])$ denotes the modification of each exist constraint set on the interaction; $e_{jk}^{v}(j \in [1,n], k \in (n,m])$, $e_{jk}^{v}(k \in [1,n], j \in (n,m])$ and $e_{jk}^{v}(j,k \in (n,m])$ denote the new constraint set on the interaction added to SoES.

Furthermore, to ensure the high-confidence during the evolution, it is necessary to keep all the constraints on the component systems and the interaction between the component systems during the evolution process. Thus, we need to study the evolution model for SoES from the behavior point of view.

A graph model for software evolution has been presented in [19]. The main objective of this graph model is to provide a framework that integrates software evolution activities with configuration control. This model formalized the objects and activities involved in software evolution in sufficient detail to enable automatic assistance for maintaining the consistency and integrity of an evolving software system. In this research work, we plan to introduce the behavior description into the structure model for SoES evolution. For this purpose, we propose to make use of the proposed graph model [19] for software evolution of the monolithic embedded system. We plan to use hyper-graphs to describe the software evolution behavior. In this case, the nodes in the hyper-graph represent the versions of software objects, and the software evolution steps are taken as the edges. Furthermore, each evolution step has constraints and these constraints are mapped into the constraints on component systems and the interaction between the component systems so that these constraints are kept. Proposed research will work out details of the needed behavioral annotations for the hyper-graphs and the processed for using those annotations to ensure that high confidence is maintained.

2.4 Description Language for the Computational Model and Evolution Model

To represent the computational model of the SoES, a set of description languages will be studied. These include the description language for the basic computational model and the description language for the evolution model.

PSDL (prototype system description language) [11] is a language for describing prototypes of real-time software systems. However, it was designed for describing the monolithic embedded systems rather than the systems of embedded systems. Thus, in this proposed research, we propose to extend PSDL to describe the basic computational model for the systems of embedded systems. We call this new description language SoES- PSDL. This description language should have facilities for recording and enforcing the non-functional constraints such as timing constraints, resource constraints and synchronization constraints, and for modeling the control aspects of systems of embedded systems using nonprocedural control constraints, component systems abstraction and interaction abstraction.

Like PSDL, SoES-PSDL is proposed to have two parts, i.e., specification and implementation. The specification contains attributes describing the form of the interface, the timing characteristics, and both formal and informal descriptions of the observable behavior of the component systems. The implementation part determines whether the component system is atomic or composite and the architecture of composite subsystems.

To represent software evolution for SoES, we propose to development a description language, called SEDL (software evolution description language), to describe the evolution model for systems of embedded systems. This description language should have the facilities to represent the node, i.e. the version of the component systems, the version of the interaction between component systems, and changes, i.e., the evolution step and the constraints on the evolution step. Like the SoES-PSDL, ESDL also has two parts for specification and implementation. In the specification part, the version of the component systems and interaction between the component systems are revealed. And in the implementation part, the evolution step and the constraints on the evolution steps can be represented. This language will be used to support analysis of impact of changes on high confidence requirements.

2.5 Prototyping Based High-confidence Assurance for SoES

2.5.1 Analyses of Robust high confidence

The concept 'robust high confidence' here refers to the ability of software systems maintain high confidence under uncertainties.

The uncertainties mainly come from the following unpredictable performance changes of the hardware on which the software executes. Especially for embedded systems, many constraints for the software come from the hardware. The performance changes of the hardware will directly influence the performance of the software embedded. These uncertainties stem from many different causes:

- Inside changes: the parts' wearing away
- Outside changes: the changes of circumstance, e.g. temperature, humidity, magnetic field, etc
- Measurement errors: the measured properties of the hardware generally have errors with the real ones.
- Modeling errors: It's difficult to get the accurate model of a hardware system. So the differences between the model and the corresponding real system are always inevitable.
- Hardware upgrades: parts are replaced, new parts are different possibly with incompletely known properties

These uncertainties will result in uncertainties of some metrics related to high confidence of the applied software. The aim of the robust high confidence analysis is to analyze the robustness of the high confidence, that is, the capacity that the software system keeps high confidence under uncertainties.

'Robust high confidence analysis (RDA)' is different from 'sensitivity analysis of high confidence'. Current sensitivity analyses methods are based on classical sensitivity theory that considers the uncertainties of the parameter as sufficiently small. Actually, in many cases, the parameter uncertainties vary within a bounded interval. Robustness analysis is just to handle this kind of problems. The problem can be formalized as follows.

Suppose the computational model of a nominal software system is: $\zeta = \zeta(\alpha)$, where α is parameter vector that compose the system. The perturbation of the system can be described as $\partial\zeta = \partial\zeta(\partial\alpha)$, where $\partial\alpha$ is the perturbation parameter vector, and $\|\partial\alpha\| \le \mu$, $\|\bullet\|$ is a certain *norm* of the vector, μ is a finite constant.

Then the real system is $\overline{\zeta} = \overline{\zeta}(\overline{\alpha}) = \zeta + \partial\zeta$, where $\overline{\alpha} = \alpha + \partial\alpha$. The high confidence function H_f of the system ζ will be in the form of $H_f = H_f(\zeta(\alpha))$. We say ζ is high-confidence if the high confidence function satisfies the following inequality $H_f \ge \nu$, where ν is a given function. Then ζ is called robust high-confidence if $\overline{H}_f = \overline{H}_f(\overline{\zeta}(\overline{\alpha})) \ge \nu$.

The robustness analysis problem is to analyze if a nominal system is robust high-confidence with respect to the given perturbation bound. The key is to find the sufficient and necessary high confidence condition $\mathrm{CON}(\zeta,\nu)$. That means, if and only if $\mathrm{CON}(\zeta,\nu)$ holds, ζ is robust high-confidence. The condition $\mathrm{CON}(\zeta,\nu)$ only depends on the nominal system ζ and the perturbation bound ν.

There are some available approaches and techniques in control theory that can deal with large-scale perturbations of system parameters. We will develop the robust analysis methods to high confidence by using the most effective robust analysis methodology from the area of control theory.

2.5.2 Robust prototyping based development methodology to high-confidence SoES

The final target of the proposal is to develop the high-confidence software system in the most efficient way. Prototyping based method can reduce the cost of development and evolution of the software. Consequently, it should be less costly to construct a high-confidence prototype system than to construct a production one. We can guarantee the high confidence of the product system by making the prototype system be robust high confidence. The robust high confidence analysis of the prototype system plays a key role in the process.

PROJECT DESCRIPTION

A prototype is a concrete executable model of a selected aspect of a proposed system. A prototype system is a partial representation of the system, including only those attributes necessary for meeting the requirements. It serves as an aid in analysis and design rather than as production software[11]. Consequently, the end product system will exist certain differences which can be taken as uncertainties upon the prototype system. For example, the prototype

- might not include all aspects of the intended system.
- might have been implemented using resources that will not be available in the actual operating environment.
- might not be able to handle the full workload of the intended system, or
- might meet its timing constraints only with respect to linearly scaled simulated time.

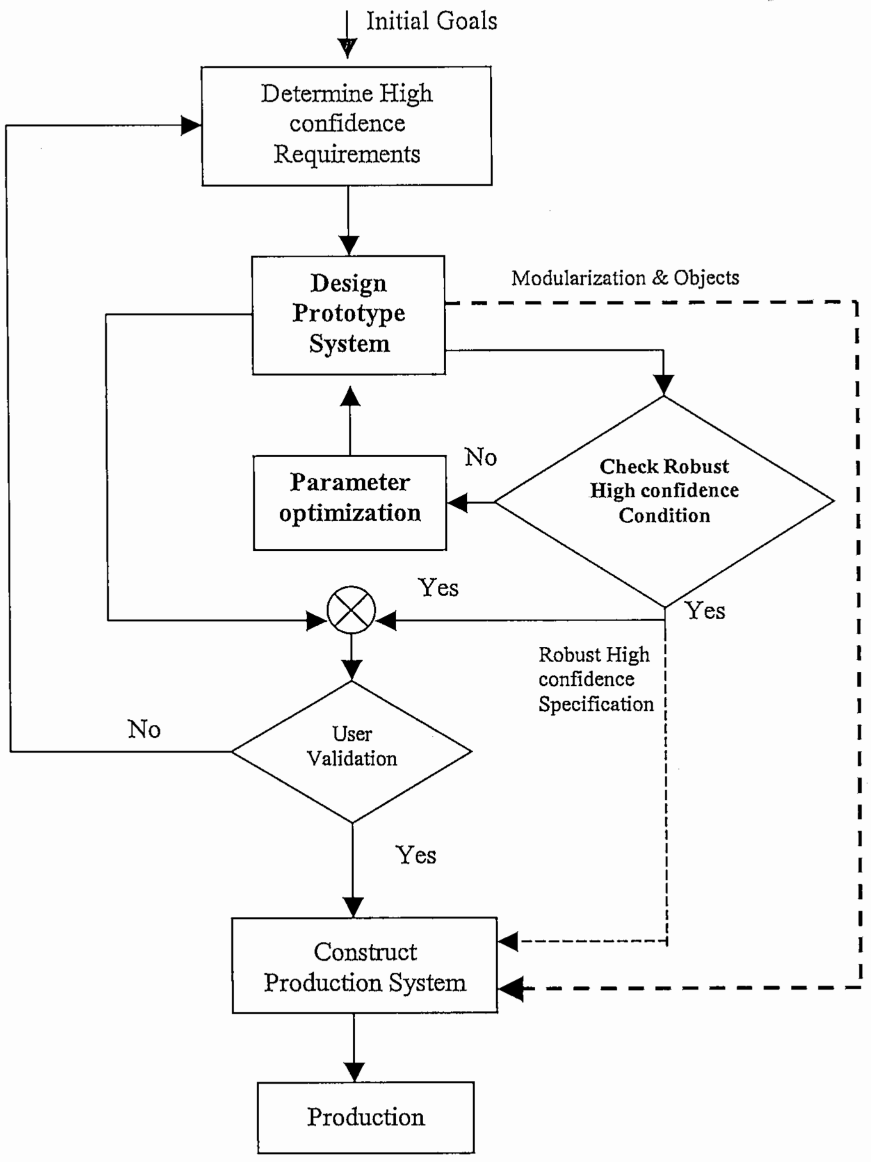
Initial Goals
Determine High confidence Requirements
Design Prototype System
Modularization & Objects
Check Robust High confidence Condition
No
Parameter optimization
Yes
Yes
Robust High confidence Specification
No
User Validation
Yes
Construct Production System
Production

Fig.1 Robust development methodology of high-confidence SoES

These differences listed above can be considered as the perturbation for the production system with respect to the corresponding prototype system. We use $P = P(\alpha)$ to represent the prototype system model, use $\partial P = \partial P(\partial\alpha)$ to represent the difference model between the prototype system and the production system, where $\partial\alpha$ is the parameter difference vector.

To implement robust high confidence analysis, we have to estimate the upper bound of $\partial\alpha$. The estimation will base on the requirements and the constraints of the envisaged system. The estimation will not be necessarily exact. But conservative estimation of the errors will lead to the conservative results about the robust high confidence.

Based on the robust high confidence analysis, the design of the high-confidence product system can be transformed into the design of a robust high-confidence prototype system. That is to find best parameter α^* so that high confidence condition $\mathrm{CON}(\zeta(\alpha^*), \nu)$ holds. Some optimization techniques will be used in searching the parameter vector α^*.

We will develop a rapid prototype based robust development methodology. The scheme of the methodology is illustrated in Fig.1.

The users and designer works together to define the high confidence requirements according to initial goals of the envisaged system. Use the extended PSDL language to give the specification of the requirements. Then construct a prototype that fits the high confidence specification. Estimate the upper bound of the difference between the prototype system and the end production system based on the initial requirement and the prototype system. Use robust high confidence analysis (RDA) to check if the prototype retains the high confidence properties under a bounded uncertainty. If not, the parameter optimization method will be used to find the suitable parameter vector α^*, and the prototype will be modified. After that, users evaluate the resulting prototype's behavior against its expected behavior. If the prototype fails to execute properly, the user identifies problems and works with the designer to redefine the requirements. This process continues until the user determines that the prototype successfully captures the critical aspects of the envisioned systems.

C3 General Plan of Work

C3.1 Plan Overview

We envision the computational model supported by PSDL that we currently have as the starting point for our proposed research. This computational model can only describe the monolithic embedded systems and fail to describe the distinguish properties of SoES. Thus, basing on the computational model supported by traditional PSDL, we will develop a new computational model for SoES. This computational model will describe the distinguish properties of SoES. Furthermore, in order to capture the high-confidence properties during whole development process, the high-confidence metrics will be introduced into the new computational model.

Since high-confidence metric is an important factor in the computational model, we should study the quantitative measure for the high confidence. How to map the high-confidence metrics into the design parameters will be also studied.

To develop the evolution model for SoES, firstly, we will give the evolution model from the structure's point of view basing on the computational model for SoES,. Then, by extending the graph model for evolution, the evolution behavior is introduced into the evolution structure model so that the completed evolution model for SoES is built. Furthermore, we should study how to keep the consistency of the high-confidence properties during the evolution process. The robust high confidence analysis approach will be used.

C3.2 Broad Design Activity

The steps in constructing high-confidence SoES by systemic development technologies are:

(1) Study the quantitative measures for high confidence of SoES.
(2) Study the method to map the high confidence metrics to the design parameter during the design phase.
(3) Extend the computational model supported by PSDL to the case of SoES so that this computational model can describe the static structure and dynamic behavior of SoES; Introduce the high-confidence metrics into the computational model.
(4) Develop the evolution model for SoES from the structure point of view and by extending the graph model for the evolution of monolithic embedded systems to describe the evolution behavior for SoES, the evolution behavior is introduced in to the evolution model for SoES from the structure point of view so that the whole evolution model is developed.
(5) Extend the PSDL and specify the description languages for the computational model and evolution model for SoES.
(6) Study the method to do robust high confidence analysis of prototype system

C3.3 Deliverables

(1) Quantitative measure for high confidence of SoES.
(2) A computational model for high-confidence SoES
(3) An evolution model for high-confidence SoES
(4) Description languages to specify the computational model and evolution model
(5) Approach to accomplish robust high confidence analysis of the prototype system

C3.4 Description of Procedure

To achieve the work described above, we will take the following steps.

(1) Study the distinguish properties of SoES, especially the emergent properties.
(2) Determine the high-confidence metrics for SoES.
(3) Take availability and reliability as the start point to map these two high-confidence metrics to the constraints during the design phase.
(4) Extend the computational model supported by PSDL to describe the distinguish properties of SoES and introduce the high-confidence metrics in the computational model.
(5) Study the evolution in SoES; Extend the graph model which describes the behavior of evolution to the case of SoES, integrate it with evolution structure model
(6) Create extensions to PSDL to specify the computational model for SoES

C3.5 Evaluation Factors

C3.5.1 Reduction in Development Cost

Since a majority of software faults can be attributed to requirements and specification errors, it is necessary to study the feasibility of requirements before large amounts of effort and expense are committed to the project. Thus, prototyping has emerged as a preferred method for requirements validation. In our proposed research, we explore support for prototyping of SoES as a complement to approaches for certification. This proposed research is to ensure that the specified properties, to be certified by other methods, are valid with respect to the real-world context of the SoES and the real-world needs of its stakeholders. This will greatly reduce the development cost since it avoids to put large amount of effort and expense basing on an infeasible requirements.

Furthermore, since the robust high confidence analysis is able to reveal the impact of changing requirements on the high confidence of SoES, this can prevent the change of requirements from comprising the high confidence. This also reduces the development cost because the invalidation of requirement changes can be recognized in advance, and the robustness of architectures for high confidence SoES will be improved.

C3.5.2 Reduction of Development and Maintenance Time

In our proposed research, the evolution of SoES is considered and the evolution model for SoES will be developed. Basing on this evolution model, the new prototype of SoES can be quickly generated according to the change of the requirements. This will reduce lag time between discovery of new requirements and building SoES software to meet those requirements. Therefore the development and maintenance time will be decreased.

C3.6 Schedule

(1) Study the quantitative measure for high confidence of SoES (6 months)
(2) Study the method to map the high confidence metrics to the design parameter during the design phase.(10 months)
(3) Extend the computational model supported by PSDL to the case of SoES so that this computational model can describe the static structure and dynamic behavior of SoES; Introduce the high-confidence metrics into the computational model. (10 months)
(4) Develop the evolution model for SoES from the structure point of view and by extending the graph model for the evolution of monolithic embedded systems to describe the evolution behavior for SoES, the evolution behavior is introduced in to the evolution model for SoES from the structure point of view so that the whole evolution model is developed.(8 months)
(5) Extend the PSDL and specify the description languages for the computational model and evolution model for SoES. (6 months)
(6) Study the method to do robust high confidence analysis of prototype system (8 months)

C3.7 Comparison with Other Research

In recent years, many approaches have been proposed to construct high-confidence embedded systems. Those approaches include verification, run-time testing and monitoring and component-based composition and rapid prototyping.

[1], [2], [3] and [4] are some typical research works for the approach based on verification. These works use model checking to verify the satisfaction of some properties such as survivability properties. [1] uses a model checker based on NuSMV to verify the survivability properties in embedded systems. [2] focuses on the specification and analysis of publish-subscribe software architecture. It specifies the properties in linear temporal logic and uses the SMV model checker to complete the verification. [3] gives an efficient procedure for verifying that a finite-state concurrent system meets a specification expressed in a (propositional, branching-time) temporal logic. In [4], the requirements are formalized in temporal logic and the system model is abstracted from the source codes so that some real-time properties are verified. [5] focuses on the verification of real-time systems. It presents a modular framework for providing temporal properties of real-time systems basing on clocked transition systems and liner temporal logic. In this framework, the properties of real-time systems can be established by use of deductive verification rules, verification diagrams and automatic invariant generation.

For run-time testing, both [6] and [7] attempt to automatically generate the test suite from the formal requirement specification. Furthermore, [6] uses model checking while [7] uses a test derivation schema to achieve this purpose. [8] presents a comprehensive method for run-time monitoring. It uses a monitor script to generate a filter and event recognizer and generates a run-time checker basing on a formal specification so that a set of resource-specific, safety and real-time properties are monitored and checked at run-time.

For research on component-based composition, [9] presents a method for automatic integration of reusable embedded systems. This research uses a component model, a component behavior model and a control plan to compose reusable components in embedded systems. Timing constraints and resource constraints are considered during the component composition.

However, all approaches mentioned above focus on building high-confidence monolithic embedded systems rather than the high-confidence SoES. Since they fail to capture the distinguished characteristics of SoES, it is hard to construct high-confidence SoES by these approaches. Furthermore, all these approaches assume the existence of an accurate and valid formal specification of the properties that are to hold with high confidence. The last thirty years of computing research have shown that this assumption does not hold in practice, and that a majority of software faults can be attributed to requirements and specification errors. Thus, rapid prototyping has emerged as a preferred method for requirements validation. It is a complement to approaches for certification.

A lot of works have been done for constructing the rapid prototyping of embedded systems. [11,12, 23-27] are some typical works on this aspect. They use operators and the data streams between the operators to model the embedded systems and capture the timing constraints and control constraints of embedded systems. [30,31] are also based on the rapid prototyping. They present a rapid prototyping technique that focuses on transition management in hybrid systems. This approach integrates hybrid modeling and simulation with a generic transition management pattern built on a Real-Time CORBA-based platform for reconfigurable control systems. [32] proposes an automated approach to communication architecture synthesis in rapid prototyping of real-time embedded systems.

However, these approaches basing on the rapid prototyping also focus on the individual embedded systems. Besides failure to abstract the distinguish features of SoES, these approaches fail to capture the high-confidence requirements so that the high confidence properties can not be certified during the development process. It is difficult to use these approaches directly for the construction of high-confidence SoES.

Therefore, further efforts on the systematic development technologies for high-confidence SoES should be made. Moreover, development of large software systems causes a sequence of modeling tasks. It requires the modeling and description of the application domain, software requirements, software architecture, software components, programs, their internal structure and their implementation --- be it by one or by several modeling concepts [10]. Thus, in this proposed research, we present a computational model for SoES with high-confidence properties. It forms the basis for systematic development of high-confidence SoES.

Moreover, in order to deal with requirements changes, an evolution model is also presented as the basis for the software evolution in SoES. Although some works [19,20] have been done for the evolution of individual embedded systems, the evolution in SoES is much more complicated than the general embedded systems. It is difficult to keep the consistency of the high-confidence property during the evolution process of SoES. Thus, in this proposed research, we study the specific method based on an evolution model to deal with the evolution in SoES.

C4 Broader Impacts

If the systematic development technologies are provided by this proposed research, then the following can take place:

(1) High confidence of SoES will be enhanced as it is constructed by using these technologies.
(2) Decrease the software faults in SoES due to the error of requirements.
(3) Enable feasibility studies for the development of SoES and avoid to put large amounts of effort and expense basing on an infeasible requirements.
(4) Help to estimate the costs of the development of high-confidence SoES.
(5) The lag time between discovery of new requirements and building SoES software to meet those requirements will be reduced.

(6) Reveal the impact of changing requirement on the high confidence of SoES to prevent the change of requirements from comprising the high confidence of SoES.

C4.1 Transition of Technology

Technology transfer will be addressed by integrating the proposed new capabilities with the traditional PSDL. Publishing results in IEEE, ACM, and OMG sponsored conferences and making toolkits readily available can facilitate acceptance.

The software Engineering Group at the Naval Postgraduate School offers M.S. and Ph.D. degrees. The students at NPS will contribute to this research and development effort. We also plan to integrate emerging technologies into the courses we teach.

C4.2 Experimentation and Integration Plan

The work will be performed by the faculty of the Software Engineering Group at the Naval Postgraduate School and their Ph.D. and M.S. Students. The principal investigators will be responsible for coordination of the following plan previously stated in section C.3.6 for schedule.

REFFERENCES CITED

1. Jeannette Wing, Scenario Graph Generation and MDP-Based Analysis, Presentation at *ARO Kickoff Meeting*, University of Pennsylvania, Philadelphia, PA, May 24 - 25, 2001.

2. David Garlan, Model Checking Publish-Subscribe Software Architectures, Presentation at *ARO Kickoff Meeting*, University of Pennsylvania, Philadelphia, PA, May 24 - 25, 2001.

3. E.M. Clark, E.A. Emerson, A.P. Sistla, Automatic Verification of finite state concurrent systems using temporal logic specification, *http://citeseer.nj.nec.com/clarke93verification.html*

4. Matthew Dwyer, John Hatcliff, and George Avrunin, Software Model Checking for Embedded Systems, *www.cis.ksu.edu/~dwyer/projects/HCES-May-01-1.ppt.*

5. Nikolaj S. Bjorner, Zohar Manna, Henny B. Sipma, Deductive Verification of Real-time Systems Using SteP, *Technical Report STAN-CS-TR-98-1616*, Computer Science Department, Stanford University, December 1998.

6. H.S. Hong, I. Lee, O. Sokolsky and S.D. Cha, Automatic Test Generation from Statecharts Using Model Checking, *Proceedings of FATES'01, Workshop on Formal Approaches to Testing of Software*, August 2001.

7. D. Clarke and I. Lee, Automatic Test Generation for the Analysis of a Real-Time System: Case Study, *Proceedings of 3rd IEEE Real-Time Technology and Applications Symposium (RTAS '97)*, Jun 1997.

8. Lee, S. Kannan, M. Kim, O. Sokolsky, M. Viswanathan, Runtime Assurance Based On Formal Specifications, *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications,* Las Vegas, June 28-July1, 1999.

9. Shige Wang and Kang G. Shin, An Architecture for Embedded Software Integration Using Reusable Components, *Proceedings of International Conference on Compilers, Architecture, and Synthesis for Embedded Systems,* San Jose, CA. 2000.

10. Manfred Broy, Specification and Modeling: An Academic Perspective, *Proceedings of the 23rd International Conference on Software Engineering (ICSE'01)*, Toronto, Canada, May 12-19, 2001.

11. Luqi, Valdis Berzins, Raymond T. Yeh, A Prototyping Language for Real-time Software, *IEEE Transactions on Software Engineering*, Vol.14, No. 10, October 1988.

12. Luqi, Real-time Constraints in A Rapid Prototyping Language, Computer Language, vol.18, No.2, 1993.

13. All Dasdan, Timing Analysis of Embedded Real-time Systems, *Thesis*, University of Illinois at Urbana-Champaign, 1999.

14. Edward A. Lee, Embedded Software, *To appear in Advances in Computers*, Vol.56, Academic Press, London, 2002.

15. Miroslaw Malek, High confidence Concepts, Measures and Models, *Technical Report*, *http://www.informatik.hu-berlin.de/rok/zs/zs2_1-4.pdf.*

16. Mark W. Maier, Architecting Principles for Systems-of-Systems, Technical Report, *http://www.infoed.com/Open/PAPERS/systems.htm.*

17. Luqi, Xianzhong Liang. "Dependability-Assured Software Transformation", Technical Report, #NPS-SW-02-008, NPS Monterey, CA: July 2002.

REFFERENCES CITED

18. Luqi, Xianzhong Liang, "Perspective-based Approach for Highly Dependable Software-Intensive Systems", Technical Report, #NPS-SW-02-0012, NPS Monterey, CA: Nov. 2002.

19. Luqi, A graph Model for Software Evolution, IEEE Transactions on Software Engineering, Vol. 16, No. 8, August 1990.

20. Luqi, " Software evolution via rapid prototyping," Computer, Col. 22, no.5, pp.13-25, May 1989.

21. Luqi, V. Berzins, M.Shing, N. Nada and C. Eagle, Software Evolution Approach for the Development of Command and Control Systems, Command and Control Research and Technology Symposium, Monterey, U.S.A, 2000.

22. Luqi, Ying Qiao, Lin Zhang, Computational Model for High-confidence Embedded Systems development, Monterey workshop 2002, Venice, Italy, Oct. 7-11, 2002.

23. Luqi, M.Shing, Real-time Scheduling for Software Prototyping, Journal of Systems Integration, 6, 41-72, Kluwer Academic Publishers, Boston, 1996.

24. Bernd Kramer, Luqi, Compositional Semantics of a Real-time Prototyping Language, IEEE Transactions on Software Engineering, Vol.19, No. 5, May 1993.

25. Luqi, Joseph A. Goguen, Formal Methods: Promises and Problems, IEEE Software, January 1997.

26. Luqi, Valdis Berzins, Rapidly Prototyping Real-time Systems, IEEE Software, September 1998.

27. Luqi, Mohammad ketabchi, A Computer-Aided Prototyping System, IEEE Software, March 1988.

28. G.A. Agha, Abstracting Interaction Patterns: A programming Paradigm for Open Distributed Systems, in Formal Methods for Open Object-based Distributed Systems, IFIP Transactions, E. Najm and J.-B. Stefani, Eds., Chapman & Hall, 1997.

29. Titos Saridakis, Robust Development of Dependable Software Systems, Technical Report, Institut National De Recherche en Informatique et en Automatique (INRIA), June 1999.

30. M. Guler, S. Clements, N. Kejriwal, L. Wills, B. Hech, G. Vachtsevanos, Rapid Prototyping of Transition Management Code for Reconfigurable Control Systems, Prof. of the 13th IEEE International Workshop on Rapid Systems Prototyping (RSP), PP. 76-83, Darmstadt, Germany, July 2002.

31. M. Guler, A. Clements, L. Wills, B. Hech, G. Bachtsevanos, Generic Transition Management for Reconfigurable Hybrid Control Systems, To appear IEEE Control Systems Magazine.

32. Frank-Michael Renner, Jurgen Becher and Manfred Glesner, Automated Communication Synthesis for Architecture-precise Rapid Prototyping of Real-Time Embedded Systems, Prof. of the 11th IEEE International Workshop on Rapid Systems Prototyping (RSP), Paris, France, June 21-23, 2000.

33. Mark W. Maier. The Art of Systems Architecting. Eberhardt Rechtin, CRC Press, London 2000.

34. John Dobson, Dependable Systems of Systems. http://www.csr.ncl.ac.uk/projects.

35. Ronald R. Luman, Quantitative Decision Support for Upgrading Complex Systems of Systems, Ph.D Dissertation, George Washington University. Nov. 1997.

REFFERENCES CITED

36. Scott Selberg, Systems of systems, Technical Report, Agilent Technologies, Nov. 2000

37. J.C. Laprie, Dependable Computing and Fault Tolerance: Concepts and terminology, Fault Tolerant Computing Symposium 15, pp.2-11, Ann Arbor, MI, June 1985. IEEE Computer Society.

38. J. Arlat, K Kanoun, H. Maderia, et al., Dependability Benchmarking: State of the art, Technical Report, European Community under the "Information Society Technology" Programme. Aug. 30, 2001.

39. Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, Fundamental Concepts of Dependability, Technical Report, Computer Science Dept. Univ. of California, Los Angeles, April 2001.

40. J.E. Anderson, Dependability as a Measure of On-time Performance of Personal Rapid Transit Systems, Journal of Advanced Transportation, Vol.26, No.3, 1992, pp.201-212.

41. E. Jonsson, An Integrated Framework for Security and Dependability, Technical Report, Department of Computer Engineering, Chalmers University of Technology, SE-412 96 Goteborg, Sweden.

42. Y. Masuda, S. Yamakawa, A Compound Dependability Measure Arising from Semi-Markov Reliability Model, Journal of Operations Research Society of Japan.

43. Andrea Bondavalli, Ivan Mura, Kishor S. Trivedi, Dependability Modeling and Sensitivity Analysis of Schedule Maintenance System. Proc. DCCA-7-7th IFIP Int. Conference on Dependable Computing for Critical Applications. San Jose, USA, 1999, pp.319-337.

44. Walter J. Gutjahr, Software Dependability Evaluation based on Markov Usage Models, Performance Evaluation. 40, 2000. pp.199-222.

45. H. M. Hinton, Under-Specification, Composition and Emergent Properties, New Security Paradigms Workshop Langdale, Cumbria UK, 1997.

46. D. Matthews, Assessing the Value of a C4ISREW System-of-systems Capability, *www.dodccrp.org/2000ICCRTS/cd/papers/Track4/004.pdf.*

47. V. Kotov, Systems of Systems as Communicating Structures, HP Labs Technical Reports, *http://www.hpl.hp.com/techreports/97/HPL-97-124.html.*

48. A. G. Arnold and W. F. Kujawa, Test and Evaluation of Complex Systems of Systems, *www.terec.gatech.edu/graphics/ EconT&E/Kujawa_paper.pdf.*