Integrating Dependability Analysis into the Real-time System Design Process

Naruemon Wattanapongsakorn • University of Pittsburgh • Pittsburgh
Steven Levitan • University of Pittsburgh • Pittsburgh

Key Words: Design technique, Dependability, Bathtub curve, Failure rate, Real-time system, Distributed system

SUMMARY & CONCLUSIONS

In this research, we are developing a design framework for integrating dependability analysis into the distributed, heterogeneous, fault-tolerant real-time system design process. We focus on two dependability attributes: *reliability* and *availability*. We are implementing this framework on top of existing systems for the design of distributed, real-time systems such as TimeWiz (Ref. 4). This will allow system designers to evaluate system dependability, while other system evaluation concerns, such as system performance and design cost, are analyzed during every step in the system design process. Our system dependability analysis provides choices of system design based on the dependability results. In addition, we perform system dependability evaluation, or optimization, early in the system design process, without needing complete design information. In other words, with incomplete design information, we are able to predict the behavior of system dependability. This will significantly reduce the time and costs of real-time system design.

1. INTRODUCTION

Real-time computing technology has recently become one of the most demanding and challenging areas of research in computing. Distributed, heterogeneous, fault-tolerant real-time systems are required to handle complex real-time tasks in modern life such as flight control systems, industrial process control, the world-wide stock market exchange, and computer games. Therefore, the dependability of these systems must be analyzed. Dependability is a quality of service (QoS) having the attributes: reliability, availability, maintainability, testability, integrity, and safety.

During the last fifteen years, a great number of tools and techniques for dependability analysis have emerged. These tools are based on well-known techniques, such as combinatorial analysis, static and dynamic fault-trees, stochastic Petri Nets (Refs. 1, 20), as well as Markov (Refs. 14, 25, 29, 30) and queuing models (Refs. 12–14, 26). In order to handle more complex modern systems the models for these tools have been continuously enhanced. A representative set of these tools is ADAPT (Refs. 9, 23), SHARP (Ref. 31), RPM (Ref. 28), DEPEND (Ref. 17), DIFtree (Ref. 11), HARP (Ref. 2), MEADEP (Ref. 35), SAVE (Refs. 15, 18), SMART (Ref. 6), SURE (Ref. 15), and UltraSAN (Ref. 32).

Among these tools, the first three tools provide both performance and dependability evaluation. Among these three, ADAPT and SHARPE, provide only system (physical) level analysis, and RPM provides only a non-distributed (parallel) processing model. None of the tools offer composite performance modeling during synthesis (design and analysis) of distributed real-time operating systems and applications at all design levels.

In addition, current dependability evaluation tools need significant effort to build a suitable dependability model to complement the existing performance model for each tool. For existing tools, errors in designing the dependability model can cause differences between predicted performance and real system performance. Further, during system dependability analysis, small modifications may require a remodeling of the entire system (Ref. 1).

To solve this problem, a design framework that integrates dependability analysis into the system design process must be implemented. To date, there are very few such system design frameworks, and none of them support design for all design levels in the system design process, including evaluations of system redundancy, and dependency in the failure behavior of components. Furthermore, they do not offer dependability analysis of system/application "snapshots". This capability allows the system designer to observe the dependability behavior of any component in operation at any point in the design hierarchy (Ref. 2– 3).

In our framework, we have developed a technique to capture the dependability properties of hardware and software components. We provide appropriate constant/non-constant failure rate models (Refs. 5–7) for dependability analysis, evaluate functional dependencies, active (hot spares) and passive (cold spares) redundancies which commonly exist in any complex system. We also capture dependability analysis of application snapshots, essentially at the physical level of the system design process. This functionality allows the designer to observe the dependability behavior of any event in the system under design.

0-7803-5848-1/00/$10.00 © 2000 IEEE

The rest of this paper is organized as follows. In the next section, we describe our system design framework. Then, we use an example application/system to present our dependability analysis research model, both the system-modeling framework on which we base our work, and the dependability analysis framework, which spans multiple levels of the design process. Finally, in section 3, we present our plans for future work.

2. THE SYSTEM DESIGN PROCESS FRAMEWORK

Although our methodology for integrating dependability into the design process is general, the two system design frameworks upon which we are basing our work are the Systems Engineer Workbench (SEW2.0) (Refs. 4–5, 34) and the TimeWiz design assistant tools (Ref. 36).

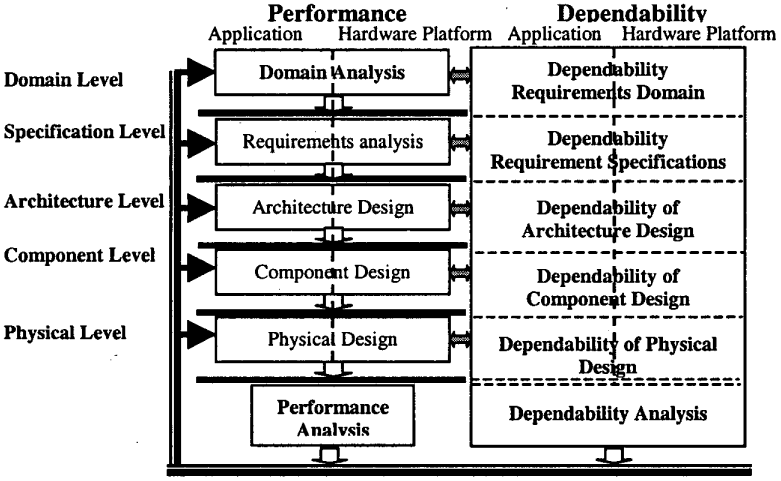


Figure 1: Dependability and System Design Process

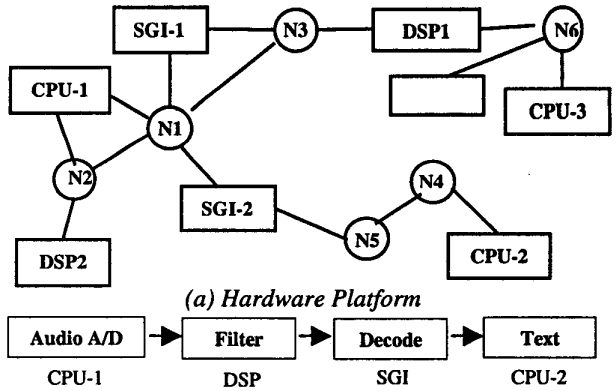
Without going into a detailed explanation (Refs. 5, 37), in a generic design framework shown on the left hand side of Figure 1, the domain analysis classifies the performance requirements for both the software applications and the hardware platform. The specification design process creates a functional performance specification from the domain requirements. The architecture design process partitions the application into a set of components. The component/physical design process maps applications onto hardware components. The application component/physical design process takes care of the binding, routing, and scheduling of applications on hardware resources creating a set of links or connections between the two models (Refs. 5, 34). We assume this generic model in the rest of this paper.

As shown in Figure 1, in our integrated tool, for each level of system design process (from the domain level down to the physical level) there is a corresponding dependability model. Input parameters for each level of dependability analysis are taken from the system design process of the application or the target-platform from the left-hand side of the figure. The dependability analysis evaluates each dependability level and reports back to each step of the design process. Then, the design process may repeat to meet the dependability and system performance requirements. Any change in the design process is automatically propagated from the higher levels down to the lower levels.

2.1. *An Example Application*

To illustrate the way dependability analysis can be integrated into the system design process, a simple example of an application is presented as it is designed, starting at the specification level (which includes the information of the domain level) down to the architecture, and the component/physical level. In this discussion, we focus mainly on the dependability analysis of the application, assuming that the application design process is taken care of by another group of application developers from which the inputs of the dependability analysis and evaluation are acquired. The dependability analysis and evaluation would be carried out incrementally in parallel with the system/application design process.

Our example is a simple speech recognition application running on a hardware platform as shown in Figure 2. The system is comprised of a computer system infrastructure, as shown in Figure 2(a), executing a set of real-time applications, including the speech recognition application. Note that the figure shows the potential hardware resources available for this application which are not all necessary. The speech recognition application, as shown in Figure 2(b), samples human speech, translates it into a set of commands that the computer can understand by performing analog to digital conversion of the speech signal, filtering or processing it, decoding the signal, and converting it to a text format. Since the speech application is CPU-intensive (Ref. 8), DSP (digital signal processing) and SGI (a high performance workstation such as a silicon graphics) servers are required to execute this algorithm in real-time.



(a) Hardware Platform

(b) Application: A Simple Speech Recognition Application

*Figure 2:*Example Hardware Platform & Application (Ref.8)

We now discuss how we integrate dependability analysis into the software application, and the hardware platform models performing the modeling and analysis, level by level. The following subsections describe each of the levels.

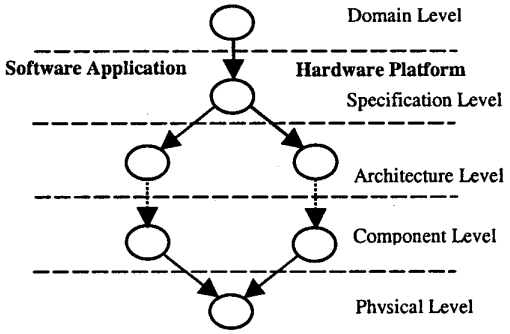


Figure 3: A Simplified Framework Model

Figure 3 shows a simplified picture of our example application in the design framework. At the top level, all the dependability requirements are not yet decomposed. At the middle levels, extra abstraction levels can be added, as shown by the dashed lines. All the physical mappings and bindings are done last at the final stage, the physical level.

Figure 4 shows the speech recognition application as it is represented in our system design process framework, level by level. The mapping, binding, and scheduling is performed and shown at the software application model.

2.1.1 *Domain Dependability Requirement Definition*

This level models the high level description of the domain dependability requirements from the user's or customer's point of view. These are the dependability requirements of a set of applications which may (or may not) run concurrently on a specified hardware platform. This requirement definition is decomposed and propagated down to the next level in the hierarchy. For brevity, we do not show details at this level.

2.1.2 *Dependability Requirement Specification*

This model is a detailed structure document, also called a functional specification. The functional specification indicates what the system/applications should do and how well they must work in terms of reliability and availability.

At this level, all dependability requirements of the application are captured at both the software application and hardware platform models, as shown in Figure 4 and Table 1. In the figure and the table, the dependability requirements are mean time to falure, *MTTF,* (hours), time range of operation, *t*, reliability, *R(t)*, and availability, *A(t)*. The other properties shown are obtained from the lower levels of the framework.

The dependability requirements of the application have constant values. However, the dependability offered from the lower levels in the hardware platform and application models can be captured either in a constant failure rate mode, or in a time-dependent failure rate mode using the Weibull distribution parameters. The Weibull distribution is selected to capture the dependability characteristics of a system/ component because it can model any stage in the "bathtub" curve (Ref. 12), which is the most common failure behavior of a component or system (Ref. 16). At this level, both of the hardware platform and application specification models, again, are decomposed and inherited in the next lower level.

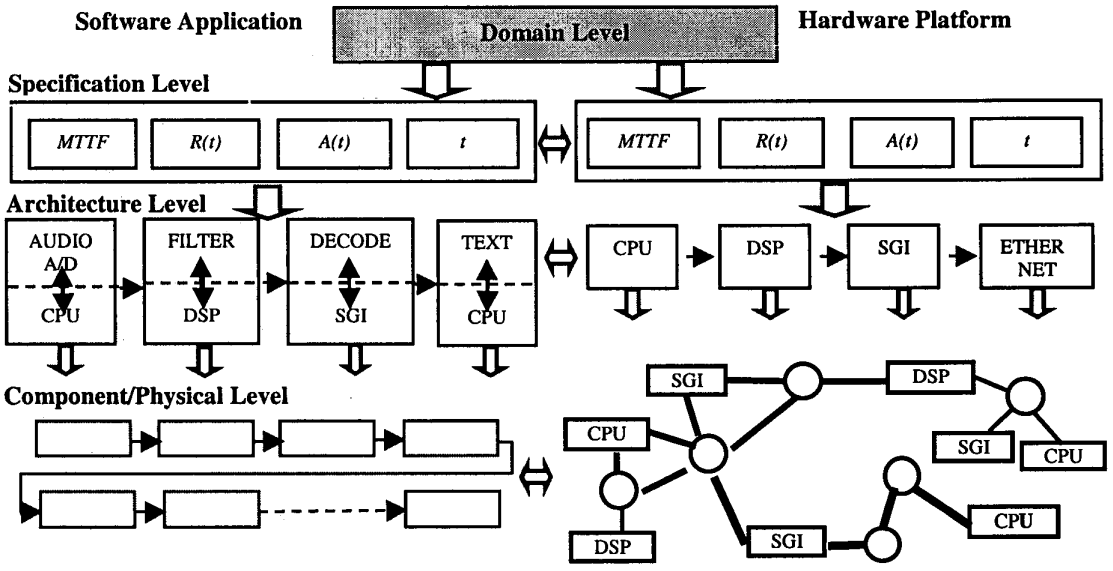


Figure 4: Example Design in the Framework

Specification Properties	Description	M	Auto	R/W	I/O
Type	Application or Hardware Platform		x	R	
Name	Object name		x	R	
Mean Time to Failures (hrs)	MTTF			W	I
MTTF LLevel Offered (Const)	MTTF offered from lower level	C	x	R	O
MTTF LLevel Offered (Weibull)	MTTF offered from lower level	T	x	R	O
Time Range of Operation, t (hrs)	Total hours of operation, t			W	I
Reliability Requirement, R(t) (scale 0-1)	R(t)	C		W	I
Reliability LLevel Offered (Const)	R(t) offered from the lower level	C	x	R	O
Reliability LLevel Offered (Weibull)	R(t) offered from the lower level	T	x	R	O
Availability Requirement (scale 0-1)	A(t)			W	I

where M = Mode which is either constant (C) or time-dependent parameter (T),Auto = Automatically generated, R/W = Read/Write, and I/O = Input/ Output for dependability analysis

Table 1: Dependability Property Description at the Specification Level

2.1.3 *Dependability of Architecture*

At this level, the dependability properties for both application software architecture and the hardware platform architecture are observed. As shown in Figure 4, the software application architecture consists of Audio A/D, Filter, Decode and Text Conversion, and the hardware platform architecture is made from CPU, DSP, SGI, and some Ethernet networks.

Architecture Property	Description	A/H	M	Auto	R/W	I/O
Type	Application or Hardware Platform	A/H		x	R	
Name	Object name	A/H		x	R	
Time Range of Operation, t (hrs)	Total hours of operation	A/H		x	W	I
Reliability Requirement (scale 0-1)	Obtained from higher level	A/H	C	x	W	I
Failure Rate Offered	Failure rate	A/H	C		W	I
Reliability Offered (scale 0-1)	Default value or input from user	A/H	C	x	W	I
Amount of Redundancy	Redundancy if greater than 1	A			W	I
Reliability Offered with Redundancy	Reliability offered	A	C	x	R	O
LLevel Reliability Offered	Reported from lower level	A/H	C	x	R	O
Reliability Offered (Weibull parameters)	Input = Weibull(shape,scale)	A/H	T		W	I
Calculated Failure Rate Offered (Weibull)	Failure rate	A/H	T	x	R	I
Calculated R(t) Offered (Weibull)	Resulted Reliability	A/H	T	x	R	O
Reliability Offered with Redundancy (Weibull)	Reliability offered	A	T	x	R	O
LLevel Reliability Offered (Weibull)	Reported from lower level	A/H	T	x	R	O

where A/H = Application/Hardware Platform, M = Mode which is either constant (C) or time-dependent parameter (T), Auto = Automatically generated, R/W = Read/ Write, and I/O = Input/ Output for dependability analysis

Table 2: Dependability Property Description at the Architecture Level

The dependability requirement of each architecture unit in the hardware platform and application models are again captured in either of two modes, a constant failure rate, or a time-dependent failure rate using the Weibull failure parameters, which provide a rough idea of the lower bound (minimum) of the dependability characteristics of each architecture unit. The dependability properties of an architecture unit are described in detail in Table 2.

From the table, the dependability attributes consist of requirements obtained from the higher levels, and dependability offered at the architecture level, which are the approximated failure rate and amount of redundancy of the units. The other attributes shown are automatically monitored and obtained from the lower levels. Note that the "amount of redundancy" attribute is shown only at the software application model, because binding and mapping of applications onto hardware resources is done at the software application model.

Dependability analysis can take place at this level by comparing the dependability requirements of both the software and hardware to see if the requirements are matched, since the software is supported by the hardware resources. Also, the application or hardware requirements must be lower than what the application or hardware platform architecture can offer, or else a more highly reliable architecture (or resource redundancy) must be used.

2.1.4 *Dependability of Component/ Physical Design*

At this level of the design process, the speech recognition architecture model is decomposed into the speech recognition component/physical model having component/physical dependability requirements as inputs to the dependability analysis of both the hardware platform and the speech recognition application dependability of component/physical models. All the required types of hardware and software components for the speech recognition processing system are specified with known dependability properties and requirements.

The software application components in the application model are supported by the hardware platform component/physical model. Thus the dependability properties of the allocated resources from the specified hardware platform components must meet the requirements of the application component/physical model. The dependability analysis is obtained at this component level both at the hardware platform and the application models to check if all the dependability requirements from the higher levels are still fullfilled under the assumption that each component meets the specified dependability requirements.

The dependability attributes of each component in this level consist of requirements obtained from the higher level, dependability offered at this level which are the failure rate and active/passive redundancy information. The other attributes shown are automatically obtained. Specifically these are R(t) offered from the component, and R(t) offered from software-hardware binding. Table 3 describes the component/physical dependability properties in detail. The

properties that have the same description as in the architecture level are not listed here.

Component Property	Description	M	Auto	R/W	I/O
Active Redundancy?	Default = Yes. Passive redundancy = No		x	W	I
Reliability of Imperfect Switch	For passive redundancy. Default = 0.995		x	W	I
N Redundancy (N out of M)	Active redundancy (N)			W	I
M Redundancy (N out of M)	Active redundancy (M)			W	I
Reliability Offered with Redundancy	Calculated passive or active redundancy	C		R	O
Reliability Offered with Redundancy (Weibull)	Calculated passive or active redundancy	T		R	O
Calculated Reliability Offered (sw&hw)	Result of software and hardware binding	C		R	O
Calculated Reliability Offered (sw&hw) (Weibull)	Result of software and hardware binding	T		R	O

Note: Other attributes that are not discussed belong to the system design performance analysis tool

Table 3: Dependability Property Description at the Application Component/Physical Level

At this component/physical level, there exists the physical routing, binding and scheduling of the application to the hardware platform components, which results in a physical modeling graph. The application architecture model and application component/physical model appear in this graph as well as additional nodes representing the dependability of communication or routing components such as busses and networks in the hardware platform component/ physical model. Component redundancy or fault-tolerance is displayed. Complete dependability analysis and evaluation can take place at this physical level, since all the required parameters are available. Figure 5 shows a snapshot at the physical level of a sub-system, Filter, mapping onto a DSP1 component and its dependability attributes (shown at the right-hand side), as well as the architecture level modeling of the speech recognition application (shown at the top-middle). All the software components for the application are shown at the left-hand side of the figure.

2.2. *Dependability Analysis*

In this section, we show the dependability analysis for the speech recognition application at each level, starting at the Specification level. For simplicity, we choose to discuss only the Const (constant failure rate) mode.

2.2.1 *Specification Level*

At this level, the application requirements are:

Overall system R(t) = 0.90, sub-system Audio A/D & Filter $R_{sub}(t)$= 0.97, MTTF = 1E+5 hours, t (time of operation) = 9600 hours, and A(t) = 1 (non-repairable system/application).

2.2.2. *Architecture Level*

All the requirements from the higher level are obtained. The dependability offered at this level is an approximated (lower bound) failure rate for the application and the hardware-platform architectures. The failure rates for the application architecture Audio A/D, Filter, Decode, and Text Conversion are chosen for this example to be: 2.083E-6, 1.041E-6, 1.041E-6, and 1.562E-6 per hour, respectively, which results in a reliability of: 0.98, 0.99, 0.99, and 0.985, respectively. Similarly, the hardware-platform architecture CPU, DSP, SGI, and a network type each has an approximate

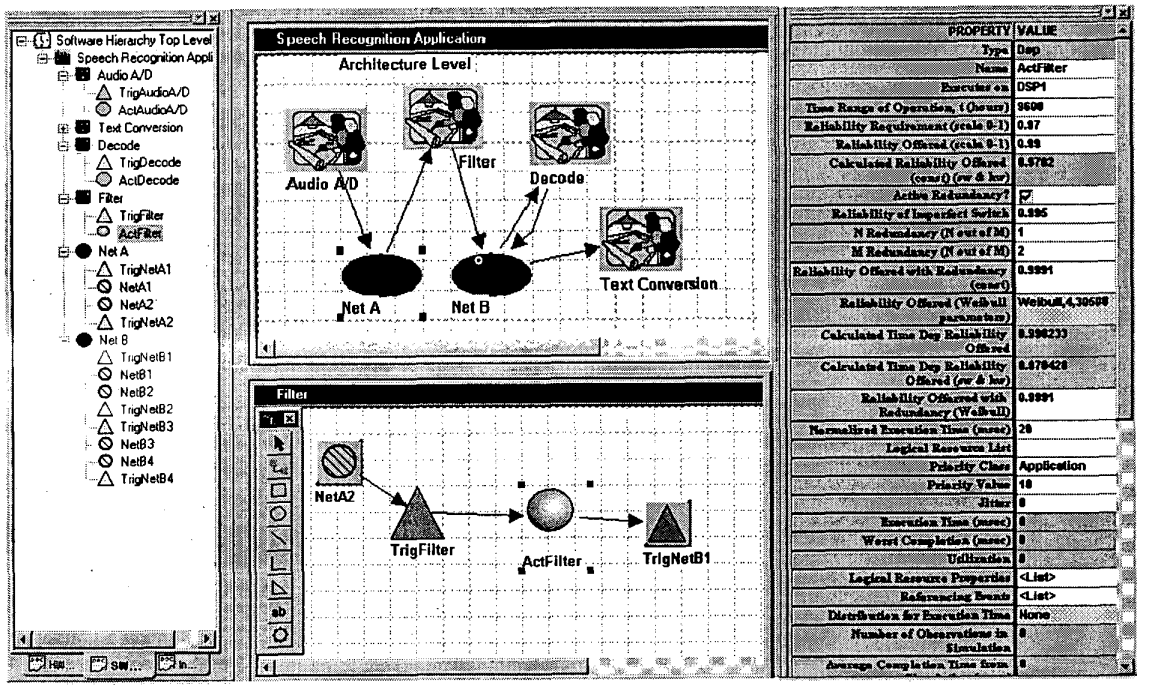


Figure 5: The Architecture and a Snapshot at the Physical Level of the Application

failure rate and a reliability value associated with it. Let us assume that the reliability for CPU, DSP, SGI, and Ethernet Network is 0.97, 0.98, 0.98, and 0.90, respectively.

At this level, the software application architectures are mapped onto the hardware-platform architectures, as shown back in Figure 4. The Audio A/D, Filter, Decode, and Text Conversion architectures are mapped onto the CPU, DSP, SGI, and CPU architectures, which result in a composite reliability of 0.95, 0.97, 0.97, and 0.9545, respectively. Note that each mapping is linked to the others by the network unit. All these mappings (with series connections) result in a system reliability 0.8532 (i.e., 0.95x0.97x0.97x0.9545), and the Audio A/D & Filter sub-system reliability of 0.9215 (i.e., 0.95x0.97), assuming a perfect network connection.

At this point, it is obvious that redundancy is needed in order to obtain the system reliability requirement R(t) = 0.90 and the sub-system requirement $R_{sub}(t)$ =0.97. Therefore, in our framework, we introduce a resource redundancy option into the application architecture level. For example, applying an extra set of resources at the Audio A/D mapping onto the CPU, and at the Filter mapping onto the DSP, as shown in Figure 5. The sub-system and the overall system have the new reliability of $R_{sub}(t)$ = 0.9966 (i.e., $(1-(1-0.95)^2)*(1-(1-0.97)^2)$), and R(t) = 0.9227 (i.e., 0.9975* 0.9991* 0.97* 0.9545), assuming a perfect network connection. This analysis gives an approximate upper bound of the reliability offered at this level.

Without our dependability integration model, this redundancy adjustment information is not available until the final stage of the design process, the physical level. This illustrates one major benefit of our design methodology: to be able to predict system dependability early in the system/application design stage, without needing the complete design information. These decisions are also inherited by the lower level.

2.2.3. *Component/Physical level*

At this level, the mapping of applications to the hardware platform components for system reliability has taken place. To find the best mapping, we perform a reliability analysis (ignoring performance modeling) using a technique called **breadth-first search**; where all possible selected paths are considered. For simplicity of illustration, each mapping unit, which contains system hardware, system software, and application software components is considered to have its reliability value as shown in Figure 6. Without redundancy, the overall analysis at this component/physical level is displayed in Figure 7; the sub-system Audio A/D & Filter and the overall system give 0.8579 and 0.6995 reliability, respectively, which do not meet reliability requirements.

Using the redundancy information from the architecture level, at this level, we provide refinements of the redundancy choices, consisting of active (N out of M), and passive (with perfect/imperfect switch) redundancy. A simple example of a redundancy set is shown in Figure 8. Here, each active redundancy unit has a one-to-one mapping of the application onto a hardware resource. This gives the sub-system and the system 0.9742 and 0.9381 reliability values, respectively. The offered system reliability of 0.9381 means that the system failure is 6.655E-6 per hour, and MTTF is 150,258.26 hours (i.e., 1/failure rate) in the constant failure rate mode, which fulfills all the application dependability requirements. This is also shown in a comparison graph in Figure 9.

The simple mapping algorithm described here made decisions based on constant failure rate for each resource. Time dependent failure rate characteristics for applications and hardware resources can be captured as well, with the Weibull distributions provided at each level in our design framework. The dependability analysis in this time dependent mode can be done using the same techniques as just described.

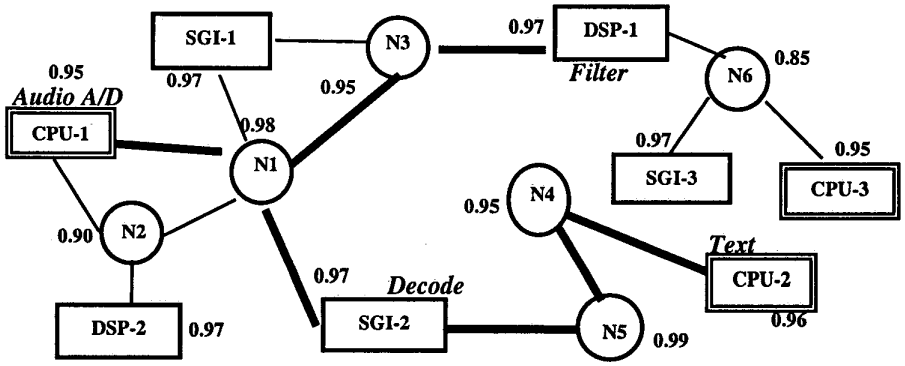


Figure 6: Mapping of the Application onto the hardware platform

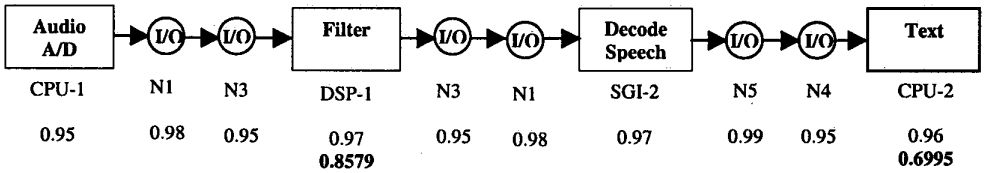


*Figure 7:*The Speech Recognition System after Mapping of the Application

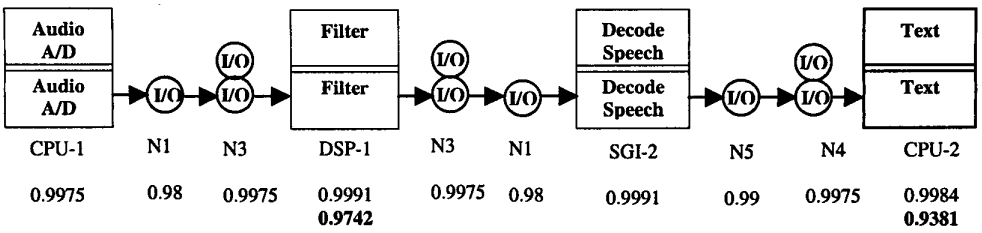


Figure 8: Speech Recognition System with Redundancy

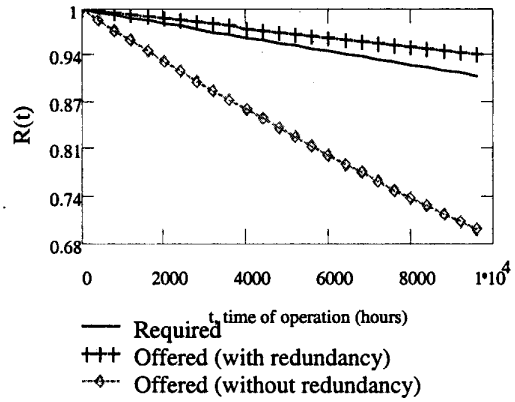


Figure 9: Dependability Requirements and Dependability Offered using Constant Failure Rate Models for the Speech Recognition Application

In this framework, all the decomposition of requirements (in parallel with the decomposition of the design) is performed from the top-down and estimates of reliability of sub-systems are propagated from the bottom-up. This gives the designer insight into the dependability properties of the system as it is being designed so that she can make informed design tradeoffs between cost, dependability and performance. As we have shown, with the redundancy information from the architecture level provided from our dependability integration model, the system designer can make decisions before the final stages of the design process achieving high quality designs in less time.

3. FUTURE WORK

We are working on developing an automated dependability generation method in the framework model. The generation will be fully or selectively automatic depending on the user's choice. Then we will determine consistency between reliability constraints in the design hierarchy of the system/application design process. In order to make sure that our dependability analysis framework captures the composite performance during synthesis of the system at all design levels, consistency and completeness checking algorithms must be developed. The algorithms must simplify the up and down consistency management of the dependability hierarchy, system design completeness verification, and perform dependability specification propagation and traceability. The results of this work will be a fully integrated performance and dependability analysis framework for the design of distributed real-time systems.

REFERENCES

1. Balakrishnan, M., Trivedi, K., "Stochastic Petri Nets for the Reliability Analysis of Communication Network Applications with Alternate-Routing", TR-96/06, March 1996. From:http://www2.ncsu.edu/eos/info/ece_info/www/ccsp/tech_reports/faculty/trivedi.html
2. Bavuso, S. J., et al., "HiRel: Hybrid Automated Reliability Predictor (HARP) Integrated Reliability Tool system (Ver. 7.0), HARP Introduction and User's Guide", Vol. 1, NASA Technical Paper 3452, Nov. 1994.
3. Bowles, J. B., "A Survey of Reliability-Prediction Procedures For Microelectronic Devices, IEEE Transactions on Reliability", Vol. 41, No.1, 1992, pp. 2-12.
4. Bradley K., Setliff D. E., Strosnider J. K., "Supporting Performance-Aware Software Development", Proceedings of the Australian Software Engineering Conference, Oct. 1998.
5. Bradley K., Strosnider J.K., Setliff D. E., "A Formal Framework For Integrated Real-time Analysis Algorithms Into a System Design Process", IEEE Real-time System Symposium, May 1998.
6. Ciardo, G., Miner, A. S., SMART: "Simulation and Markovian Analyzer for Reliability and Timing", Proceedings of the 9th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation and the 7th International Workshop on Petri Net and Performance Models, pp. 41-43, Saint Malo, France, June 1997.
7. Chatterjee, S., Strosnider J. K., "A Generalized Admissions Control Strategy for Heterogeneous", Distributed Multimedia Systems, Proceedings of the ACM Multimedia 95, Nov 1995.
8. Chatterjee, S., "Distributed Pipeline Scheduling: A Framework for Design of Large-Scale, Distributed, Heterogeneous Real-time Systems", Ph.D. Thesis, Carnegie Mellon U., 1996.
9. Choi, C. Y., Johnson, B. W., Dugan, J. B., "Dependable System Codesign Using Data Flow Models", IEEE Proceedings of the Annual Reliability and Maintainability Symposium, 1997, pp. 263-270.
10. Coit, D. W., Smith, A. E., "Reliability Optimization of Series-Parallel Systems Using a Genetic Algorithm", IEEE Transactions on Reliability, Vol. 45, No. 2, June 1996, pp. 254-260.
11. Dugan, J. B., Venkataraman, B., Gulati, R., "DIFtree: A Software Package for the Analysis of Dynamic Fault Tree Models", Proceedings IEEE Annual Reliability and Maintainability Symposium, 1997, pp. 64-70.
12. Ebeling, C. E., An Introduction to Reliability and Maintainability Engineering, The McGraw-Hill Companies,1997.
13. Elsayed, E. A., Reliability Engineering, Addision Wesley Longman, Inc., 1996
14. Fricks, R., Telek, M., Puliafito, A., Trivedi, K., "Markov Renewal Theory Applied to Performability Evaluation", TR-96/11, March 1996. From:http://www2.ncsu.edu/eos/info/ece_info/www/ccsp/tech_reports/faculty/trivedi.html

15. Geist, R., Trivedi K. S., "Reliability Estimation of Fault-Tolerant Systems: Tools and Techniques, Computer", Vol. 23, NO. 7, July 1990, pp. 52-61.
16. Gera, A. E., "The Modified Exponentiated-Weibull Distribution for Life-Time Modeling", IEEE Proceedings of the Annual Reliability and Maintainability Symposium, 1997, pp. 149-152.
17. Goswami, K. K., Iyer, R. K., "DEPEND: A Simulation-Based Environment for System Level Dependability Analysis", IEEE Transactions on Computers, Vol. 46, No. 1, Jan. 1997, pp. 60-74
18. Goyal, A. et al., "The System Availability Estimator", Proceedings of the 16th International Symposium on Fault-Tolerant Computing, CS Press, Los Alamitos, CA, July 1986, pp. 84-89.
19. Hoyland, A., Rausand M., System Reliability Theory: Models and Statistical Methods, Wiley Series in Probability and Mathematical Statistics, John Wiley & Sons, Inc., 1994.
20. Hunter, S., Philip, T., Trivedi, K., "Combined Performance and Availability Analysis of Switched Network Application", TR-96/37, October 1996.
From:http://www2.ncsu.edu/eos/info/ece_info/www/ccsp/tech_reports/faculty/trivedi.html
21. Iyer, R. K., Modulator, "Dependability of Commercial Systems", IEEE Proceedings of Fault-Tolerant Computing Systems, 1996, pp. 537-540.
22. Kececioglu, D., Reliability & Life Testing Handbook, Vol. 1, PTR Prentice Hall, 1993.
23. Klenke, R. H. et al., "An Integration Design Environment for Performance and Dependability Analysis", Design Automation Conference 97, Anaheim, CA, pp. 184-189.
24. Laprie, J.-C., Modurator, "Panel Session on Limits in Dependability", IEEE 23th International Symposium on Fault-tolerant Computing, 1993, pp. 608-613.
25. Logothetis, D., Mainkar, V., Trivedi, K., "Transient Analysis of Non-Markovian Queues via Markov Regenerative Precesses", TR-96/09, March 1996.
From: http://www2.ncsu.edu/eos/info/ece_info /www/ccsp/tech_reports /faculty/trivedi.html
26. Lyu, M. R., Editor in Chief, Handbook of Software Reliability Engineering, IEEE Computer Society Press, McGraw-Hill, 1996.
27. MIL-HDBK-217F, Reliability Prediction of Electrical Equipment, 1995.
28. Nicol, D. M., Palumbo, D. L., Ulrey, M. L., "A Graphical Model-Based Reliability Estimation Tool and Failure Mode & Effects Simulator", IEEE Proceedings Annual Reliability and Maintainability Symposium, 1995, pp. 74-81.
29. Platis, A. N., Limnios, N. E., Le Du, M., "Asymptotic Availability of Systems Modeled by Cyclic Non-Homogeneous Markov Chains", IEEE Proceedings Annual Reliability and Maintainability Symposium, 1997, pp. 293-297.
30. Randell, B., Laprie, J.-C, Kopetz, H., Littlewood, B., (Eds.), Predictably Dependable Computing Systems, ESPRIT Basic Research Series, Springer, 1995.
31. Sahner, R. A., Trivedi, K. S., "Reliability Modeling Using SHARPE, IEEE Transactions on Reliability", Vol. R-36, No. 2, June 1987, pp. 186-193.
32. Sanders, W. H., Obal II, W. D., Qureshi, M. A., Widjanarko, F. K., "The UltraSAN Modeling Environment, Performance Evaluation", Vol. 24, No. 1, Oct-Nov. 1995, pp.89-115.
33. Sommerville, I., Software Engineering, Fifth Edition, Addison-Wesley, 1996.
34. Strosnider J.K., Bradley K., Setliff D.E., Saurav Chatterjee, "An Application/Target-Platform CoDesign Approach", Design Automation Conference, submitted, June 1998, http://www.pitt.edu/~des/setliff.html
35. Tang, D, Hecht, M, Handal, J, Czekalski, L, "MEADEP and Its Applications in Evaluating Dependability for Air Traffic Control Systems", Proceedings of the 1998 Annual Reliability and Maintainability Symposium, Jan. 1998.
36. TimeWiz, "An Integrated Design Environment for Real-time Systems", TimeSys Corporation, http://www.timesys.com
37. Wattanapongsakorn, N., "Integrating Dependability Analysis into Real-time System Design Process", Ph.D. Thesis Proposal, University of Pittsburgh, 1998.

BIOGRAPHIES

Naruemon Wattanapongsakorn, PhD student, EE
267 Benedum Engineering Hall,
University of Pittsburgh, Pittsburgh, PA 15261 USA

Internet (email): naruemon@ee.pitt.edu

Naruemon Wattanapongsakorn is a PhD candidate in electrical engineering at the University of Pittsburgh. She received the B.S. degree in Computer Engineering (1994) and the M.S. degree in Electrical Engineering (1995), both from The George Washington University. Her research interests include distributed system dependability analysis, real-time system modeling, software fault-tolerance, and statistical analysis of system reliability. She is a student member of the IEEE.

Steven P. Levitan, PhD, CS
348 Benedum Engineering Hall,
University of Pittsburgh, Pittsburgh, PA 15261 USA

Internet (email): steve@ee.pitt.edu

Steven P. Levitan is the Wellington C. Carl Professor of Electrical Engineering at the University of Pittsburgh. He received the B.S. degree from Case Western Reserve University (1972) and his M.S. (1979) and Ph.D. (1984), both in Computer Science, from the University of Massachusetts, Amherst. He worked for Xylogic Systems designing hardware for computerized text processing systems and for Digital Equipment Corporation on the Silicon Synthesis project. He was an Assistant Professor from 1984 to 1986 in the Electrical and Computer Engineering Department at the University of Massachusetts. In 1987 he joined the Electrical Engineering faculty at the University of Pittsburgh. His research interests include VLSI architectures, optoelectronic computing systems, parallel algorithm design, and HDL simulation and synthesis for VLSI. He is an Associate Editor of the ACM Transactions on Design Automation of Electronic Systems. He is Chair of the ACM Special Interest Group on Design Automation, a member of OSA, and a senior member of IEEE/CS.