2010 International Conference on Educational and Information Technology (ICEIT 2010)

SA Based Software Deployment Reliability Estimation: Problem Space, Challenges and Strategies

Xihong Su, Hongwei Liu, Zhibo Wu, Decheng Zuo, Xiaozong Yang
School of Computer Science and Technology
Harbin Institute of Technology
Harbin, China
suxihong@sina.com

***Abstract*—Software architecture (SA) has been widely advocated as an effective abstraction for modeling, implementing, and evolving complex software systems such as those in distributed, decentralized, heterogeneous, mobile, and pervasive environments. We intend to investigate two problems related to the domain of those environments: software deployment and reliability. Software deployment is a post-production activity that is performed for or by the customer of a piece of software. Though many approaches for Architecture-based reliability research, little work has been done in incorporating software deployment into architecture-based reliability estimation. Thus, this paper describes which information constitute SA based software deployment reliability model and how to obtain available information for the model. The system reliability estimation at an architectural level includes software deployment, architectural style and component replica.**

Keywords-software architecture, software deployment, reliability, architectural style, component replica

I. INTRODUCTION

Over the past a few decades, we have witnessed an unrelenting pattern of growth in the size and complexity of software systems, which will likely continue well into the foreseeable future. This pattern is further evident in an emerging class of embedded and pervasive software systems. The systems are growing in popularity due to increase in the speed and capacity of hardware, and decrease in its cost, the emergence of wireless ad hoc networks, proliferation of sensors and handheld computing devices, and so on. A number of researches have shown that a promising approach to resolve the challenges is to employ the principles of software architectures [1].

In the domain of pervasive systems, system deployment is a specific facet of software system architecture. System deployment architecture is the allocation of the system software components to its hardware hosts. Deployment architecture is particularly important in pervasive environments, because a system will typically comprise many different, heterogeneous, mobile, and possibly mutable, execution platforms during its lifetime [2].

Reliability is one of the most critical extra-functional properties of a software system. A number of techniques for architecture-based reliability estimation have been proposed. These techniques have tended to oversimplify one or more of (1) the definition of software system reliability, (2) parameters influencing system reliability and (3) the challenges of estimating model parameters by assuming they are available or can easily be obtained. Although a large number of approaches of architecture-based reliability estimation, none of them incorporates software deployment, component replica, architectural style and component dependent into their reliability estimation at the same time.

To define the problem space of SA based software deployment reliability estimation, we must first arrive at a concrete definition of reliability. Reliability is a complex property with different meanings, characteristics, and associated with metrics in different contexts, even within a single application and system. This means that it is not possible or desirable to simply provide "the" reliability estimate. Instead, the global of SA based software deployment reliability estimation should be to provide the architect with a multidimensional description of reliability from different perspectives, for different parameter combinations, reliability definitions, models of system usage, and so on [3].

KrKa [3] defines the problem space of architecture-based reliability estimation. This model includes 13 ingredients, as shown in table I reliability ingredient A, but it neglects the influence of the following ingredients. These ingredients include software deployment, component replica, architectural style and component dependent. Therefore, this paper presents a framework of SA based software deployment reliability estimation incorporating software deployment, architectural style, component replica, component dependent, and as shown in table I reliability ingredient B.

II. TERMINOLOGY AND SCOPE

This section explained the related concepts: software architecture, architectural style, architecture description, architectural middleware, software deployment.

A. Software Architecture

Software architecture provides abstractions for representing the structure, behavior, and key properties of a software system. Those abstractions are described in terms of

978-1-4244-8035-7/10/$26.00 © 2010 IEEE

components (computational elements), connectors (interaction elements), and their configurations (also referred to as topologies) [4].

B. Architectural Style

Software architectural styles (e.g., publish-subscribe) further refine the vocabulary of component and connector types and propose a set of constrains on how instances of those types may be combined in a system [5].

C. Architecture Description

Architecture description specifies an architecture. An architecture can be described according to different viewpoints. Two viewpoints are frequently used in software architecture: the structural viewpoint and the behavioral viewpoint [6].

D. Architectural Middleware

Sam and Chiyoung define architectural middleware as a middleware platform that provides implementation-level constructs for key architectural abstractions: components, connectors, ports, events, styles, and so forth [7].

Prism-MW is developed at USC. It is an architectural platform that provides implementation-level support for architectural constructs in an extensible, efficient, and scalable manner. It gears to distributed, mobile, and pervasive environments [8].

E. Software Deployment

Software deployment is referred to as a collection of activities, which are to make software available for use until uninstalling it from devices. These activities include delivery, installation, configuration, activation, updating, reconfiguration, and un-installation of the software [9].

III. Problem Space of Reliability

The problem space of SA based software deployment reliability estimation is directly determined by the specific definition of reliability being considered. For example, software system reliability is the probability of failure-free operation for a specified time in a specified environment. The definition of reliability is oversimplified and incomplete. Because the definition firstly assumes that a concrete definition of a failure exists, and it is not clear how to define system failure for an arbitrary software system. Secondly, the notion of a computational environment is a complex one, and may include a number of different elements, such as the hardware characteristics of the system [10].

Let us examine some possible definitions of failure in different systems. We can say that a system has failed if it produces incorrect results or can not process requests. Most existing approaches leverage this definition to arrive at a definition of overall system reliability, stating that the system is operational when none of its components has failed. This definition of reliability is convenient, but it is not very useful when applied to complex software systems. The definition of reliability for a particular system is more specific and largely depends on the requirements applied to the system [3].

In software-intensive system environment, there are a number of sources of system failures, including hardware host failures, power source failure, physical network link failure, and so on. A definition of reliability must specify which of these will be examined when analyzing system behavior from a certain perspective. Defining system reliability is a multi-faceted problem that depends on numerous factors, risks, and uncertainties.

IV. Information and Challenges

Defining reliability is only the first step in estimating system reliability. Closely tied with it is the definition of what constitutes: failure information, operational profile and recovery information.

A. Reliability Ingredients

KrKa defines the problem space of architecture-based reliability estimation. This model includes 13 ingredients, as shown in table I reliability ingredient A. However, complex software systems can consist of a large of heterogeneous components distributed over many hosts with different hardware and software characteristics, and have a software deployment process. Therefore, our reliability estimation model discusses 17 ingredients, considering software deployment, as shown in table I reliability ingredient B.

TABLE I. Comparison of Included Reliability Ingredients

Information	No.	Reliability Ingredient A	Reliability Ingredient B
failure information	1	Failure-free	Failure-free
	2	Failure severity	Failure severity
	3	Failure impact	Failure impact
	4	Failure extent	Failure extent
	5	Probability of failure	Probability of failure
operational profile	6	Frequency of execution of different system services and operations	Frequency of execution of different system services and operations
	7	User inputs	User inputs
	8	Operational contexts	Operational contexts
	9	——	Architectural style
	10	——	Component dependent
recovery information	11	Likelihood of recovery	Likelihood of recovery
	12	Time to recovery	Time to recovery
	13	Recovery mechanism	Recovery mechanism
	14	Recovery processes	Recovery processes
	15	Extent of recovery	Extent of recovery
	16	——	Component replica
	17	——	Software deployment

The architectural styles pose constraints on the valid configurations and interaction of components. Architectural styles can be used to determine the component dependencies and reduce the space of valid configurations. In turn, architectural styles help to reduce the complexity of reliability analysis, as well as the process of finding a good architecture [11].

System reliability depends not only on the reliabilities of components in the system but also their interactions, namely,

the dependencies among them. The reliability of individual components may depend on the performance of other components [12].

In a distributed system, which consists of many hardware notes, a failure of a hardware node does not mean that the whole system fails. It can still operate, if software services are independent or if replica of critical software component exists. Then, what's the relationship between a component and its replica, or a component and other component?

With redundant copies, a replicated component can continue to provide a service in spite of the failure of some of its copies, without affecting its clients. If a hardware node fails, how to replicate the right component?

Finally, to assess the reliability of a system, we need a model that will allow all of the reliability ingredients enumerated above to be meaningfully combined. When all 17 of the ingredients enumerated are known with some certainty, reliability analysis of SA based software deployment models can be used with confidence.

B. *Availability of Reliability Ingredients*

The information contained in SA based software deployment models is rich enough to constitute a backbone for a reliability estimation technique. This section examines how 17 reliability ingredients introduced in the previous section can be obtained from architecture and software deployment process.

1) *Failure information*

KrKa describes that the definition of failure-free behavior can be directly extracted from an architectural model specification of desired component and connector behavior. Behavioral specifications within an architectural model must include both functional and non-functional properties to be complete. We think that this definition of failure-free behavior is incomplete. First, this definition does not consider the failure-free behavior of host hardware node. If a host fails, components deployed on the host will fail. Second, in practice many architectural models do not include complete descriptions of non-functional characteristics.

The definition of failure severity must be derived and composed from several sources. First, behavioral description in an architectural model may include a specification of the criticality of system services. It directly follows that the severity of a failure of a critical service is higher than that of a non-critical service. Second, user stakeholder perspectives, when captured in an architectural model, help an architect determine failure severity by defining which system users are affected by a given failure [3].

To determine failure impact of possible system, a precise specification of component interaction and deployment is required. Failure impacts are determined by how failures may propagate within the system and whether failures can be obtained within a limited scope. But KrKa does not consider the influence of component replica on failure impact. For example, if a set of components is deployed on many hardware hosts, a failure of a hardware node does not mean that the whole system fails. It can still operate, if replica of critical software component exists.

Information about failure extent is specified in terms of (1) the part of the system function affected by a failure and (2) the durability of a failure (e.g., permanent vs. recoverable). Failure extent is determined by the degree to which a given failure influences the availability of user-level services, or results in degraded operation from the users' perspective [13].

The probability of failures cannot, in most cases, be directly extracted from an architecture model and architecture deployment result. The probability of a given failure may depend on a number of factors which may not fully determined at architecture design time, such as hardware and network design, or the way in which application logic is implemented. KrKa does not consider the influence of component replica, software deployment, components' logic link on the probability of failures.

2) *Operational profile*

KrKa describes that the frequency of execution of system services is usually not captured in an architectural model. We think that it depends on many factors, such as deployment features, use case scenario specification describing important execution sequences.

The set of user inputs to the system are normally specified in an architectural model, but the frequencies and probabilities of each possible input are not. This information can, in most cases, be only approximated. KrKa can not provide the details of user inputs. The user inputs of our framework include: a set of hardware nodes (hosts) with the associated parameters a set of components with the associated parameters, a set of physical links with the associated parameters, a set of logical interaction links between software components in the distributed system, with the associated parameters, a set of services, a set of parameter constraints and so on.

The operational contexts of different system processes can be determined from a compositional evaluation of component behaviors, concurrency mechanisms, computation resources and so on. For example, an architectural model should specify which services may execute in parallel; this provides information about which other processes may be executing (and consuming computational resources) when a given service is invoked. The availability of computational resources may, in turn, affect service reliability.

Architectural style can be used to distinguish among different architectural elements of a given style and specify the architectural elements' stylistic behaviors. For example, we need to distinguish clients from servers in client-server style. Clients block after sending a request in the client-server style, while C2 Components send requests asynchronously in C2 style. Architectural style also specifies the rules and constraints that govern the architectural elements valid configuration. For example, it disallows Clients from connecting to each other in the client-server style, or allows a Filter to connect only a Pipe in the pipe-and-filter style [8]. Architectural style can not be solely determined from an architectural specification. For example, an architect can choose an appropriate style depends on the

interaction patterns among the application components, or the target environment and so on.

System reliability depends not only on the reliabilities of components in the system but also their interactions. The reliability of individual component may depend on the performance of other components. Component dependent can not also be solely determined from an architectural model. For example, the distribution of components over hardware host node can change after software deployment. Therefore, we can also obtain component dependent information from software deployment.

3) Recovery information

KrKa describes that the likelihood of recovery cannot be solely determined from an architectural model. However, he does not provide the details of factors which influence the likelihood of recovery. Our framework considers some factors: (1) likelihood of component recovery, (2) likelihood of hardware host node recovery, (3) likelihood of service recovery, (4) likelihood of physical network link recovery, and so on.

KrKa describes that time to recovery from a failure can not be ascertained from an architectural model. He also does not consider the details of factors which influence the time to recovery. Our framework considers three factors: (1) time to component recovery, (2) time to hardware host node recovery, (3) software deployment time.

The recovery mechanism employed within a given system may be specified in the system architecture model. Recovery mechanisms describe the steps taken during normal operation and after failure to ensure that recovery is possible. Recovery strategies include redundancy, replication, checkpointing mechanisms.

The recovery processes may be specified in an architectural model. Recovery processes capture the actions performed after failure occurs to mitigate the failure. For example, when a host experiences hardware problems, the system may attempt to automatically redeploy critical component running on that host. In a system utilizing checkpointing, a failed component is restarted and the checkpointed internal state is restored. Similarly, failure of a master replica, in a system employing cold replication, is recovered from by assigning one of the other replicas to become the master, and initializing a fresh replica to maintain a constant number of replicas.

The extent of recovery refers to the fragment of the system that can be returned to its full functionality. An architectural model should clearly specify the extent of recovery in the architectural specification. For example, certain types of failures can not be completely recovered from because the software system is not responsible for managing them (e.g., hardware).

Component replica must be done selectively, considering service criticality, component reliability, available resources, and so on. Service criticality may be obtained from an architectural specification. Generally, the system components have varying degrees of component reliability, the replication of a highly reliably component provides less benefit than the replication of an unreliable component. Information about constraints of components can be distilled from architectural specification that include (1) required memory for each software component, (2) possible restrictions on component locations (e.g., two components may not be allowed to reside on the same host), (3) available memory on each host. Therefore, a trade-off decision about criticality, reliability and costs of software components is needed to find the right set of component replica.

Existing researches [14-16] have shown that software system deployment architecture can have a significant effect on the system non-functional properties. If considered from the perspective of the effects of deployment on the target system, software deployment deals with at least four problems [17]: (1) initial deployment of a system onto a new host (or set of hosts), (2) deployment of a new version of a component to an existing target system, (3) static analysis, prior to the deployment, of the likely effects of the desired modifications on the target system, (4) dynamical analysis, after the deployment, of the effects of the performed modifications on the running target system.

V. STRATEGIES

In this section, we deal with the challenges of the reliability ingredients enumerated in Section 4. The approaches discussed include techniques for analyzing reliability at both the component-level and at the system-level.

A. Failure Information

Naturally, every reliability estimation technique has a definition of failure-free operation, although some are more simplistic than others. In [18], a failure is defined as the failure of any particular service provided by system components, while in [19], a failure is defined as the failure of any individual component. Roshandel [20] allows system failures to be defined as Boolean combinations of individual component failures. These approaches do not explore the relationship of other non-properties to reliability. KrKa considers the inclusion of these non-properties with reliability models to be an important direction for future area. Our reliability model includes some non-properties, such as hardware host node failure, physical network link failure, component logical link failure and so on.

Cheung explicitly considers failure severity in their analyses, and uses multiple failure states to account for failures of different severities. For example, Goseva illustrates how to compute the overall failure severity distribution [21].

Cortellessa explicitly considers failure impact of possible system faults in their analysis. It is assumed that some component failures are independently from others in the system. Cortellessa considers the inclusion of error propagation and component collocation with reliability [22]. However, they do not consider the influence of component replica on failure impact. We consider component replica within reliability models to determine failure impact.

Many approaches do not differentiate between different failure extents. They consider failure of part of the component/system as a complete inability to perform further tasks. Roshandel dose accommodate one aspect of failure

extent by allowing system failures to be defined as Boolean combinations of component failures.

It is naturally to use failure probability to analyze the reliability. The failure probability, Goseva relates the probability of failure to a well-known complexity metric, namely, the number of states and edges in a component state chart model. Reussner [18] derives the failure probability of component services through a combination of the reliabilities of method bodies, method calls and returns, and the environment, but it does not specify how these input values are obtained. These approaches do not consider software deployment in calculating failure probability.

We assume that the failure probability of a component, a host and physical network link can not change during software deployment. The distribution of components over hardware host can change after software deployment. We can calculate failure probability again. For example, the probability of failure of a component is dependent on the host on which it is placed and on the links which it uses to serve a certain service.

B. Operational Profile

The frequency of execution of system services and operations are essential for reliability estimation techniques. Chenug requires the probabilities of transitions between internal component states, while system-level approaches need the probabilities of transfer of control between components and services, or the probabilities of execution of particular execution paths. Redrigues also requires the transfer probabilities of control between execution paths. However, only Chenug proposes a way of obtaining these parameters. These parameters can be derived from a combination of expert knowledge, functionally similar components, and system simulations.

The user inputs provide a modeling notation that account for user inputs in reliability estimation, using annotations of UML Use Case diagrams. However, Cortellessa dose not specify how to obtain the required quantitative data. Prism-MW supports for system monitoring and deployment. Our model uses Prism-MW to obtain the following information: (1) internal software properties, (2) external properties, (3) changes in the structure of the software, (4) for each pair of software components in the system, the number of times these components interact is recorded. Since the monitored data represents the most recent operational, structural, and contextual profile of the system execution, they can be used to assess the system reliability more accurately.

Reliability estimation techniques pay little or no attention to operational contexts. Cortellessa touches on concurrency within the context of a single scenario, but do not account for the degree of concurrency in modern software system. Additionally, a few approaches take sharing of computational resources into account, so this represents another important area for further study.

Few works have suggested the appropriate architectural style for applications in pervasive environment. Prism-MW provides explicit support for individual architectural styles and multiple architectural styles even within a single application [23]. In our model, we estimate system reliability of SA based software deployment considering different styles by Prism-MW. We also estimate the influence of different styles on system reliability, considering software deployment and component replica.

Our model assumes the failing events of each replica are independent. Because all replicas are on different hosts and therefore should have different causing events for hosts and link failures. There can never be the same components deployed on the same host, because a constraint of the framework prohibits this double deployment. Additionally, Dong[24] specifies that the relationship between components may be parallel, loops and so on.

C. Recovery Information

Cheung explicitly models the characteristics of failure recovery. The likelihood of recovery and time to recover from a given failure is accounted for in a limited form, as they assign a recovery probability to each failure state. Cheung mainly considers likelihood of component recovery. He does not consider the recovery likelihood of hardware host node, services and physical network links. Our model includes these factors. For example, we can use MTTF and MTTR to calculate recovery probability of hardware host node.

In general, the time to recover from a failure cannot be ascertained from an architectural model. Firstly, recovering from a failure may require manual intervention, whose duration can be highly variable. Secondly, even when recovery mechanisms are fully automated, the time to execute may depend on other unknown parameters, such as the services requested by users, the state and availability of computational resources. KrKa[3] does not provide the details of factors that influence the time to recovery. Our model describes time to component recovery and time to hardware host node recovery. Additionally, Time to recovery must be taken into account the amount of downtime the system will experience in order to redeploy.

In order to solve component replica multidimensional problem, we need to develop some algorithms that determine a subset of software components. There are two types of replication strategies: active replication and passive replication. On the basis of these two main algorithms, we design component replication decision algorithms. Our replication decision algorithms are implemented in DeSi [15], a visual environment that supports specification, analysis, and implementation of distributed software system deployment architecture.

While the system architects may choose a computationally expensive deployment strategy initially, they may be forced to switch to light-weight system monitoring and fast during the system execution, though less precise, redeployment calculations. Deployment decision algorithms are also implemented in DeSi. Because software deployment architecture is an exponential problem, we need to design approximation algorithms. We can translate software component deployment into multi-way cut problem, and so on.

Most existing approaches do not consider recovery mechanism, recovery processes, or extent of recovery. These

approaches model systems at a higher-level of granularity than is not ideal for a detailed specification of recovery processes and mechanisms as well as their extent, so while they can, in theory, incorporate these elements, it is difficult to do so in an accurate and informative way.

VI. CONCLUSIONS

This paper defines the problem space of SA based software deployment reliability estimation, presents and discusses the reliability ingredients, and describes the challenges in obtaining the necessary information for reliability estimation. We have also studied how to deal with those challenges.

ACKNOWLEDGMENT

The work is supported by High Technology Research and Development Program of China (Project No. 2008AA01A201), National High Technology Research and Development Plan of China (Project No. 2006AA01A103), National Natural Science Funds of China (Project No.60503015).

REFERENCES

[1] C.Seo, S.Malek, G.Edwards, D. Popescu, et al., "Exploring the Role of Software Architecture in Dynamic and Fault Tolerant Pervasive Systems," Proc. IEEE Symp. Software Engineering for Pervasive Computing (SEPCASE'07), IEEE Press, May 2007, pp.9-15, doi: 10.1109/SEPCASE.2007.6.

[2] N. Medvidovic, S.Malek, "Software Deployment Architecture and Quality and Quality-of-Service in Pervasive Environments", Proc. IEEE Symp. Engineering of Software Services for Pervasive Envrioments (ESSPE'07), Dubrovnik, Croatia , Sept. 2007, pp.47-51.

[3] I. Krka, L. Cheung, G. Edwards, L. Golubchik, N. Medvidovic, "Architecture- based Software Reliability Estimation: Problem Space, Challenges, and Strategies", In the Proceedings of the DSN Workshop on Architecting Dependable Systems(WADS'08), Anchorage,AK , Jun 2008, pp. 32-38.

[4] R. Thomas, "Architectural Styles and the Design of Network-based Software Architectures", Ph.D.Dissertation, University of California, Irvine, 2000.

[5] M.Shaw, D. Garlan, "Software Architecture: Perspectives on an Emerging Discipline", Prentice Hall Ordering Inforamtion,1996.

[6] F. Oquendo, "Dynamic Software Architectures: Formally Modeling Structure and Behaviour with $\pi -$ ADL", Proc. IEEE Symp. Software Engineering Advances (ICSEA'08), IEEE Press,Otc.2008, pp.352-359, doi:10.1109/ICSEA.2008.47.

[7] S.Malek, C. Seo, N. Medviovic, "Tailoring an Architectural Middleware Platform to a Heterogeneous Embedded Environment", Proc. IEEE Symp. Software Engineering and Middleware(SEM'06), ACM Press, Nov.2006 , pp.63-70, Portland, Oregon .

[8] S.Malek, "Dealing with the Crosscutting Structure of Software Architectural Styles", Proc. IEEE Symp. Computer Software and Applications(COMPSAC'08) , IEEE Press, July 2008,pp.385-392, doi:10.1109/COMPSAC.2008.124.

[9] C. Vo, T. Torabi, "A Framework of Over the Air Provider-initiated Software Deployment on Mobile Devices", Proc. IEEE Symp. Software Engineering (ASWEC'08), IEEE Press, March 2008, pp.633-638, doi:10.1109/ASWEC.2008.31.

[10] J.D. Musa,"Software Reliability Engineering", McGraw- Hill,1999.

[11] S.Malek, R. Roshandel, D. Kilgore, I. Elhag, "Improving the Reliability of Mobile Software Systems through Continuous Analysis and Proactive Reconfiguration", Proc. IEEE Symp. Software Engineering-Companion Volume (ICSE-Companion'09), IEEE Press, May 2009, pp.275-278, doi:10.1109/ICSE-COMPANION. 2009. 5071000.

[12] D.W.Coit , J.R. English, "System Reliability Modeling Considering the Dependence of Component Environmental Influences", Proc. IEEE Symp. Reliability and Maintainability Symposium (RASM'99), IEEE Press, Jan.1999, pp.214-218, doi: 10.1109/ RAMS. 1999. 744121.

[13] I. Krka, G. Edwards, L.Cheung, L. Golubchik, N. Medvidovic, "A Comprehensive Exploration of Challenges in Architecture-Based Reliability Estimation", Architecting Dependable System VI, vol. 5835/2009, pp. 202-227, Otc. 2009.

[14] G. Hunt, L.Scott, "The Coign Automatic Distributed Partitioning System", Proc. IEEE Symp. Operating System Design and Implementation, ACM Press, Feb. 1999, pp.187-200, New Orleans, Louisiana.

[15] S. Malek, "A User-Centric Approach for Improving a Distributed Software System's Deployment Architecture", Ph.D.Dissertation, USC, May 2007.

[16] M. Mikic-Rakic, S. Malek, N.Medvidovic, "Improving Availability in Large, Distributed, Component-Based Systems via Redeployment", Component Deployment, vol. 3798/2005, pp.83-98, Nov.2005.

[17] M. Mikic-Rakic, N. Medvidovic, "Architecture-Level Support for Software Component Deployment in Resource Constrained Environments", Component Deployment, vol.2370/2002, pp.493-502, Jan. 2002.

[18] R.H.Reussner, H.W.Schmidt, H.Poernomo, "Reliability Prediction for Component-based Software Architectures", J. Systems and Software, vol.66, pp.241-252, May2003.

[19] V. Cortellessa, H. Singh, B. Cukic, "Early Reliability Assessment of UML Based Software Models", Proc. IEEE Symp. Software and Performance (WOSP'02), ACM Press, July 2002,pp:302-309, Rome, Italy.

[20] R. Roshandel, N.Medvidovic, L.Golubchik, "A Bayesian Model for predicting Reliability of Software Systems at the Architectural Level", Software Architectures, Compoents, and Applications, vol. 4880/2007, pp.108-126, Jan. 2008.

[21] L. Cheung, R. Roshandel, N.Medvidovic, L. Golubchik, "Early Prediction of Software Component Reliability", Proc. IEEE Symp. Software Engineering (ICSE'08),IEEE Press, May 2008,pp.111-120, doi:10.1145/1368088.1368104.

[22] Cortellesa, v., Grassi, V., "A Modeling Approach to Analyze the Impact pf Error Propagation on Reliability of Component-Based System", Component-Based Software Engineering, vol. 4608/2007, pp.140-156, July 2007.

[23] S. Malek, M.Mikic-Rakic, N. Medvidovic, "A Style-Aware Architectural Middleware for Resource-Constrained, Distributed Systems", Proc. IEEE Symp. Software Engineering, March 2005,pp. 256-272,doi:10.1109/TSE.2005.29.

[24] W. Dong, H. Ning, Y. Ming, "Reliability Analysis of Component-Based Software Based on Relationships of Components", Proc. IEEE Symp. Web Services (ICWS'08), Sept.2008, pp.814-815, doi: 10.1109/ICWS.2008.83.