Purdue University
Purdue e-Pubs

Department of Computer Science Technical Reports

Department of Computer Science

1999

Towards the Evaluation of Netcentric Scientific applications

Shanhani Markus

Elias N. Houstis
Purdue University, enh@cs.purdue.edu

Report Number:
99-044

Markus, Shanhani and Houstis, Elias N., "Towards the Evaluation of Netcentric Scientific applications" (1999). *Department of Computer Science Technical Reports*. Paper 1474.
https://docs.lib.purdue.edu/cstech/1474

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.

TOWARDS THE EVALUATION OF NETCENTRIC SCIENTIFIC APPLICATIONS

Shahani Marcus
Elias N. Houstis

CSD TR #99-044
December 1999

Towards the Evaluation of Netcentric Scientific Applications

Shahani Markus and Elias N. Houstis

1. Introduction

A complete and rigorous analytical evaluation model for netcentric applications is particularly hard to present due to the indeterministic nature of the underlying network. Since each netcentric application is based on a different set of goals and often attempts to optimize a different set of qualities, it is not possible to encompass the accurate analytical assessment of all such applications under a single set of evaluation criteria. A separate analytic model is needed for each application, which is a costly process in terms of testing model validity and adequacy. Further, a truly useful evaluation framework for such applications would have to incorporate qualitative assessments in addition to quantitative measurements. For instance, in the case of netcentric scientific applications, it is important to consider the performance of the computational components in comparison to the communication components. However, this consideration should also take into account the pragmatic motives in selecting a particular distributed model over a stand-alone system. These motives may include non-quantifiable, yet justifiable reasons such as availability and resource sharing. Furthermore, a particular netcentric application architecture could be governed by a certain principle or goal. For example, some netcentric scientific applications are specifically designed with the concept of legacy code reuse as its primary objective. In such cases, it is imperative for any performance evaluation model to factor in such a design constraint.

A significant amount of research has been done in quantitatively measuring, testing and analyzing distributed application components. These techniques include testing for fault tolerance using fault injection strategies [7], availability analysis using Markov modeling [8], modifiability assessment using predictive coupling measures [2], performance evaluation using real time analysis [13] and informal methods to ensure design quality [3]. However, these measurement techniques are performed in isolation and do not take into account the motivating factors and intricate relationships between various quality attributes [15] of a distributed application. The Software Architecture Analysis Method (SAAM) presented in [9] is a sequence of steps to map architectural quality goals to scenarios that measure the goals, mechanisms that realize the scenarios and analytic models that measure the results. The Architecture Tradeoff Analysis Method (ATAM) [12] extends this scenario-based approach to focus on multiple quality attributes in a risk mitigation method aimed at locating and analyzing tradeoffs in a software architecture. Both these methods are useful in identifying software architecture design decisions and their rationale [10][11]. They help in investigating how an application's quality goals interact with and impact each other in various *usage* scenarios and *system evolution* scenarios.

In this document, we seek to define a unified evaluation framework based on the SAAM and ATAM techniques to qualitatively evaluate netcentric scientific applications. This evaluation framework is designed to be extensible to easily incorporate any application-specific qualitative goals. Further, the

framework would allow construction of analytical models for a quantitative study of specific actions in an application. Use of this evaluation framework would lead to the benefit of rigorously documenting the netcentric scientific application's architectural design goals, characteristics and implementation mechanisms, and would help in systematically assessing the impact of any future architectural additions or modifications. Although this technique cannot be used effectively to compare different netcentric scientific applications, it can be used to determine an implementation's success in achieving its design goals and requirements.

This document is organized as follows: In section 2, we present the background details for the proposed evaluation model. In section 3, we investigate the model's adequacy in assessing netcentric scientific applications. We present the evaluation model in section 4 and in section 5, we define a set of quality attributes to be used as a basis in evaluation process. In section 6, we describe an exercise in the application of this model to evaluate a sample feature of an existing netcentric system. In section 7, we discuss future research work in enhancing this model and we conclude our remarks in section 8.

2. Overview

Scientists accustomed to performance evaluation techniques in parallel programming may be disconcerted at the almost invariable lack of speedup in netcentric applications. In fact, more often than not, distributed applications will display significant latency (i.e.: the time lag between communication over the network and the speed of the processors). However, in such instances, one should bear in mind the motivations behind the distributed computing model and weigh in these advantages against such latency disadvantages.

Netcentric applications are difficult to debug and maintain. They are subject to partial failures that are sometimes hard to detect and could substantially affect the integrity of the overall system. Evaluating the performance of netcentric applications is equally complicated, with the presence of several conflicting underlying factors that affect the analysis. These influences range from low-level prognostics such as network bandwidth, load and TCP-window-size to high-level issues such as functional partitioning, data distribution and service replication. A rigorous analysis should consider factors such as these and also incorporate the *motivations* behind a particular netcentric application architecture. For instance, a particular network-based scientific application may provide a scientist with remote access to a large PSE (Problem Solving Environment) via the Internet. This would eliminate the need for the scientist to install the complex PSE locally. In addition to requiring an inordinate amount of computer storage, the installation of such a PSE may require a disproportionate amount of manpower in comparison to its usage. In such cases, the ability to remotely access and use the capabilities of such an environment far outweighs the potentially high user-response times. An evaluation model for such a network-based application should be

able to reflect these qualitative goals in addition to identifying its potentially high communication latencies. We use a scenario-based analysis technique to elicit and validate such goals and requirements in our proposed evaluation and analysis model.

Scenario-based architectural engineering has been extensively used in software architecture design to understand the ways in which a system meets its operational requirements [1]. It is an iterative method of design and analysis, where design decisions are motivated by scenarios.

A scenario is defined as a brief description of a single end-to-end interaction with a software system from a developer or user ("stakeholder") perspective [9]. This concept is similar to the notion of "use cases" in the object-oriented community. In our evaluation model for netcentric scientific applications, we extend this definition by combining *one or more* interactions in a scenario. This allows us to appropriately capture the typical interactions that occur in netcentric applications.

A stakeholder can be identified as a person with valid interest in the software system such as, a system developer, system administrator, programmer, component developer, potential reuser, software maintainer or end user. The scenario input from these stakeholders leads to an understanding of the relationship between quality attributes such as performance, reliability and scalability and this in turn helps in the architectural analysis process.

Scenarios may be "direct", reflecting uses of the system, or "indirect", reflecting changes to the system. Scenarios may encompass many system requirements and are obtained from stakeholders by brainstorming the uses of and potential modifications to the system. This process frequently results in the creation of new requirements. Thus, scenarios operationalize requirements in order to understand their mapping onto the system architecture, their impact on the system components and their interactions within the system.

3. Adequacy of Evaluation Model

An architectural analysis is only as good as the set of requirements or goals that it seeks to validate [11]. Hence the coverage of the scenarios in anticipating all the uses and potential changes to the system, and their adequacy in determining all the requirements and goals of the system is an important issue. Based on the definition of adequacy given in [6], we can formally state the adequacy of a scenario-based analysis methodology as follows:

> For a software system architecture A, a set of scenarios S, is *adequate* with respect to a requirements coverage domain R, if for each r in R, there exists an s in S such that r is covered when A is evaluated on S.

Measuring such evaluation model adequacy is an open problem. As noted in [6], the popular hypothesis is that adequacy assessment and improving a test set against a sequence of well-defined, increasingly strong coverage domains leads to increased confidence in the software system under review.

A possible approach towards building an adequate set of scenarios would be to formulate scenarios that manifest each desired quality attribute. Though this leads to strong coverage of the set of predetermined qualities and the resultant requirements, it does not guarantee coverage of the entire domain of requirements. However, it assists in building a significantly broad set of relevant scenarios. We refer to such quality-attribute driven scenarios as *features*.

4. Evaluation Model

We begin the process of building the evaluation model for a netcentric scientific application by defining a set of quality attributes for the application. The proposed methodology does not impose any restriction on the number or type of attributes included in this set of qualities. As a guideline, in the next section we describe a set of quality attributes that we have identified to be common across the class of netcentric scientific applications.

The elicitation process is facilitated by the use of a *requirements elicitation table* (figure 1) based on an extension of the generic "scenario elicitation matrix" described in [11]. The requirements elicitation table is designed with the purpose of encouraging the clear specification of an application's requirements. These requirements are obtained by guiding scenario specifications from each relevant stakeholder perspective, across a set of precisely defined quality attributes. This process of synthesizing an application's goals is formalized via their explicit documentation in the requirements elicitation table.

		Quality Attribute 1	**Quality Attribute 2**		**Quality Attribute *n***
Scenario *i*	Stakeholder *j*		Requirement description		Requirement description

***Figure 1:** Requirements Elicitation Table*

Once the requirements elicitation table is generated, it has to be mapped onto the application architecture. In [11], a sequence of steps is presented to translate generic quality goals to specific architectural mechanisms and their associated analyses. We have expanded this process and derived a *feature translation algorithm* (figure 3) to guide the mapping of features onto the application architecture. Based on this algorithm, the mapping process leads to the definition of specific *mechanisms* that realize the requirements. The

requirements elicitation table is used to synthesize these mechanisms in an iterative process until they are deemed acceptable in relation to the stakeholders' perspectives. The selected mechanisms are then used as a basis to elicit *actions* that implement the application's requirements.

The actions determination process is facilitated by the use of an *actions elicitation table* (figure 2). This table allows for the clear specification of actions that instantiate the mechanisms under a set of application constraints. This process highlights the effect of design constraints on the application's architecture and on its desired quality attributes.

		Quality Attribute 1	**Quality Attribute 2**		**Quality Attribute *n***
Mechanism *i*	Constraint *j*		Action description		Action description

***Figure 2:** Actions Elicitation Table*

By synthesizing the actions in the elicitation table, an action(s) is chosen for final analysis. In this last step of the algorithm, an analytical (mathematical) model is constructed for the selected action(s). For an accurate analysis of this mathematical model, it should be considered along with the requirements and actions elicitation tables since they contain the intrinsic knowledge on the effect of constraints and motivations on the underlying application architecture. The analytical model can then be used to assess the actions on the architecture, determine their success in realizing the requirements and pinpoint potential bottlenecks or detrimental characteristics.

This process of mapping features or scenarios onto the application's architecture explicitly integrates "design motivations" and "design constraints" into the analysis and leads to a *realistic* evaluation of the goals and requirements of netcentric scientific applications. A significant side benefit of this model is the rigorous documentation that is generated during the process. Figure 4 illustrates the evaluation model's iterative documentation phases, which are driven by the feature translation algorithm.

The optional abstract requirements, abstract mechanisms and abstract actions listed in the feature translation algorithm are needed when this methodology is used to evaluate application frameworks (reusable software architectures). In such cases, an abstract requirement, mechanism or action for the framework is translated into a specific requirement, mechanism or action respectively, by considering a particular application instance of the reusable architecture. This is the technique adopted in SAAM for the analysis of a family of systems in its sequence of steps to map scenarios onto a shared architecture.

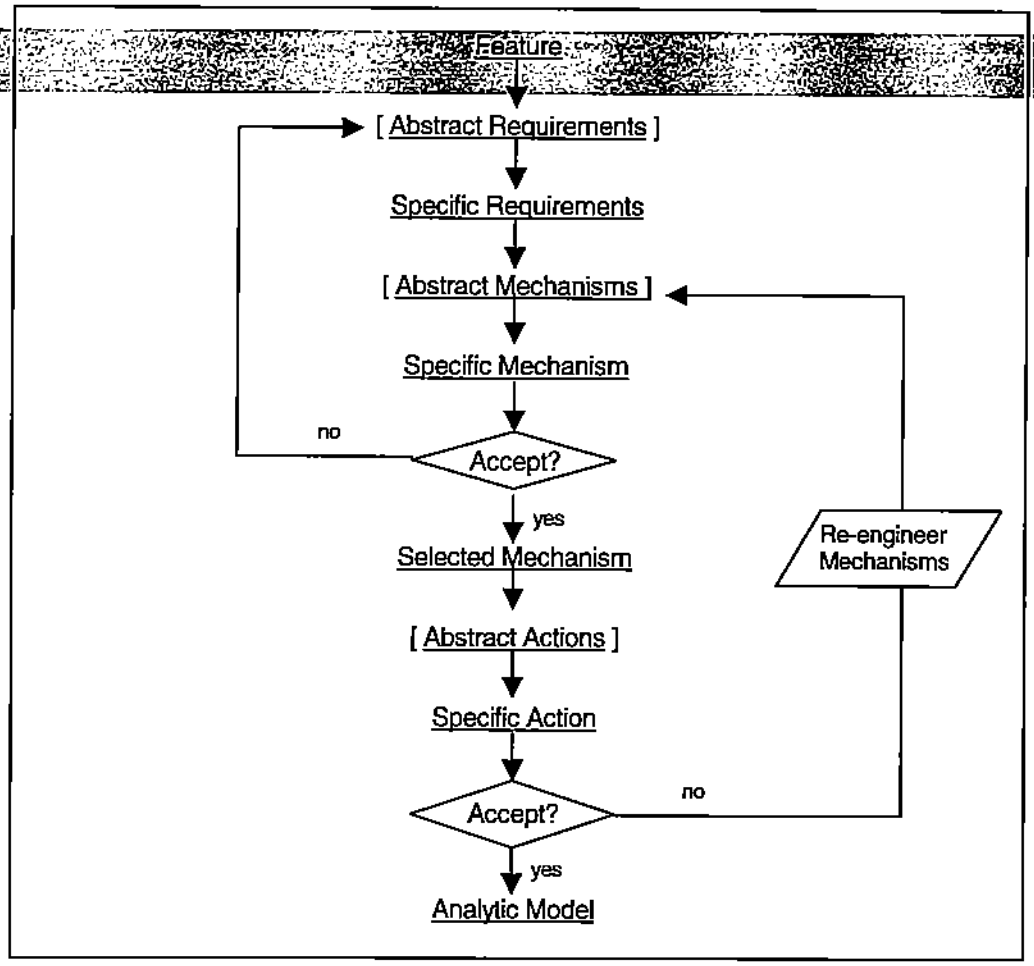

Figure 3: *Feature Translation Algorithm*

5. Quality Attributes for Netcentric Scientific Applications

We have identified six quality attributes that are important in netcentric scientific application design. We recommend these qualities be used as the basis set in the evaluation model. They are defined as follows:

- *Performance*: How well and how fast the system responds to user requests under "normal" execution conditions. It is directly related to the performance of the computational components and the communication latencies.

- *Fault tolerance:* How well the system responds to and recovers from partial failure. This includes important mechanisms for failure detection. Responses may vary (user-level or system-level notification) and recovery

approaches may include strategies such as redundancy and saving computation progress via systematic checkpointing.

- *Security:* How secure the system is against intrusions and malicious attacks that would compromise its integrity. Also includes the application's ability to protect the privacy and integrity of system and user data. Includes the concepts of secure communication and user authentication.

- *Reliability:* How good the system is at completing its task or informing the user if it cannot do so. For estimated long tasks, this may include periodic updates of the computation status. This quality basically reflects the end-user's confidence in the application. It includes the notion of repeatability. That is, if the application solves a particular problem and obtains a solution once, then it should be able to repeatedly solve the same problem and obtain a mathematically approximate solution in the future, or give a precise reason for being unable to do so (such as a crucial computational server failure).

- *Scalability:* How well the system scales with respect to an increase in one or more of the following: the number of simultaneous users, the number of concurrent computations and the complexity of the computations.

- *Accessibility:* The degree of access the application provides to a remote user. This may range from none to full remote user access. This quality does not encompass authentication and instead, pertains to the notion of availability for remote use over a network.

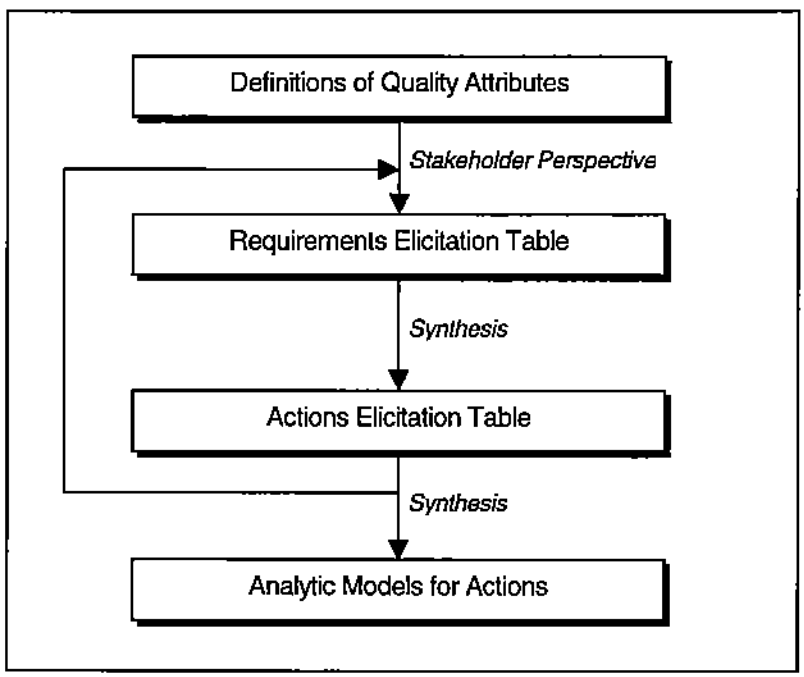

***Figure 4:** Iterative Documentation Phases*

These definitions are meant to be guidelines. Individual application evaluators may need to enhance these descriptions to include specific perspectives related to the application. Furthermore, attributes can be added to this set to reflect intrinsic quality goals that motivate a particular application's design. For instance, a netcentric PSE application may include extensibility and flexibility into this set to reflect the importance of these qualities in the design of a scientific PSE.

The stakeholders that provide input into the requirements elicitation table need to have a good understanding of these quality attributes in order to make an effective contribution. Hence, our evaluation model requires an *a priori* definition of all quality attributes that are included in the requirements elicitation table. Such definitions may be clarified by noting any relationships between quality attributes. For example, the typical quality attributes for netcentric scientific applications described above are related in a hierarchical manner (figure 5). In this hierarchy, the attributes on the same level tend to have a proportional or inversely proportional effect on each other.

In the requirements elicitation and actions elicitation phases, these quality attribute hierarchies can be implicitly assumed to help in the formulations, provided they have been clearly documented in the quality-attribute definition phase. For instance, in addressing a reliability feature, one may implicitly assume that the fault tolerance requirements have been satisfied.

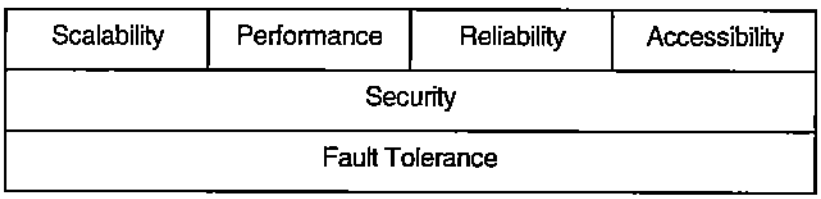

***Figure 5:** Hierarchy of Quality Attributes for Netcentric Scientific Applications*

6. Applying the Feature Translation Algorithm: An Example

In this section, we present a sample evaluation process driven by the feature translation algorithm. For this purpose, we consider an actual netcentric scientific application, the network-based NetPellpack PSE for the solution of PDE (Partial Differential Equation) problems [14].

In this example, we base our evaluation process on the manifestation of a single quality attribute – the accessibility quality. The quality-driven scenario (feature) for this application is "ubiquitous remote access to a PDE solver". The requirements elicitation table pertaining to this feature is given below.

Feature or Scenario	Stakeholders	Fault-tolerance	Security	Scalability	Performance	Reliability	Accessibility
Provide ubiquitous remote access to the PSE server in order to begin a PDE problem solution session.	End User				Access to the PSE server should be setup in a reasonable amount of time.	Must be able to gain access to the PSE server repeatedly, in the same manner.	Must be able to gain access in a convenient manner, at any time from any platform.
	System Developer	Must ensure availability of a PSE server that can solve the PDE problem. If a server is not available, inform remote user.	Should authenticate the remote user.				

The above table operationalizes the "ubiquitous remote access" feature of the NetPellpack application to elicit requirements. The stated feature refers to simply gaining initial access to the PSE server and does not encompass later session oriented interactions related to the actual PDE solution process. To facilitate the ubiquitous access feature, the NetPellpack prototype implementation provides Web-based access to the PSE server in order to establish a session-oriented connection for the actual problem solving process. Thus, NetPellpack provides initial access via a Web server that redirects authenticated requests to a PSE server.

The computational component of the PSE server consists of a complex legacy PSE system. For cost-effective rapid prototyping purposes, the NetPellpack system reuses this legacy software installation. Thus, the current implementation of NetPellpack supports only a single PSE server. This constraint should be considered in evaluating the NetPellpack architecture. The requirements mapping onto the architecture is given below.

Feature: Ubiquitous remote access to PSE server → *Specific Requirement:* Web-based access → *Specific Mechanism:* Client-side HTTP request redirection.

An evaluation of this mechanism seems to reasonably realize the requirements specified in the elicitation table. We now evaluate the effectiveness of the mechanism in relation to the constraints and in terms of its actual instantiation in the actions elicitation table.

Mechanisms	Constraints	Fault Tolerance	Security	Scalability	Performance	Reliability	Accessibility
Client-side HTTP request redirection.	Reuse legacy PSE installation – only a single PSE server available.	Verify PSE server availability. If it is not available, the remote user should be informed *before* the client attempts redirection.		Spawn a separate process on the PSE server to handle each session-oriented connection request.			

Examining these actions reveals a shortcoming in the prototype implementation - the remote user is *not* informed if the PSE server is unavailable. This is a direct consequence of using a client-side HTTP redirection mechanism. Realizing the desired fault tolerance requirement would therefore require a different mechanism. This re-engineering process involves shifting the evaluation back to the actions elicitation table in order to define a new set of mechanisms. For clarity of this presentation, we shall assume the completion of this iterative process and the instantiation of a suitable mechanism that meets the fault tolerance criteria.

The final step in the evaluation is the synthesis of an analytical model. The communication and processing latencies in the sequence of interactions can be modeled as follows:

$$\text{Perceived latency by end user} = T_{Lreq} + T_{Lres} + t_{auth} + t_{chk} + t_{serv},$$

where, T_{Lreq} = communication latency for request, T_{Lres} = communication latency for response, t_{auth} = elapsed time for authentication, t_{chk} = elapsed time to perform the PSE server availability check and t_{serv} = elapsed time to send client information from the Web server to the PSE server and the time to spawn a process that would establish the session-oriented connection.

Typically this latency would be dominated by, $T_{Lreq} + T_{Lres}$. We have succeeded in optimizing this latency by locating and removing the client-side request redirection mechanism that originally resulted in double this quantity.

This example validates the proposed evaluation model's capability in discovering qualitative flaws in application architectures. Thereby, it illustrates the feature translation algorithm's ability to analyze and improve the architectural design or implementation of a netcentric scientific application.

7. Future Work

To develop this preliminary evaluation model for netcentric scientific applications into a comprehensive and useful methodology, detailed case studies are essential. Further, suitable metrics to evaluate the "goodness" of this model and other scenario-based evaluation techniques could be formulated.

The proposed systematic evaluation process could be simplified by a tool that assists in building the elicitation tables and in stepping through the feature translation algorithm. With the addition of a database, such a tool would be valuable as a documentation repository for application architecture designs and modifications.

Application patterns [4] such as master/worker, marketplace and specialist patterns are observable in many netcentric scientific applications. A collection of these application patterns can be identified and classified for the class of netcentric scientific applications in a manner similar to the classification of design patterns in object-oriented software [5].

The evaluation model can be enhanced by overlaying the analysis process with a schema that captures the application pattern governing the architecture design. The schema can be designed as an abstract architectural framework that mimics a particular application pattern. Using a schema, pattern-specific templates can be generated, which identify common features, requirements and actions. This would speed up the evaluation process and provide greater insight into a netcentric application's architecture. The use of such templates could be easily incorporated into an evaluation-enabling tool as described above.

8. Conclusion

The proposed model qualitatively evaluates an architectural design or an implementation of a netcentric application by operationalizing the application's requirements across a desired set of quality attributes, realizing these requirements with a set of mechanisms and instantiating these mechanisms by specific actions. Specific actions are modeled analytically to locate bottlenecks and areas of improvement. Thus, this systematic evaluation methodology helps in analyzing the architectural design and in identifying the impact of any modifications on the overall application.

This rigorous evaluation process is significant in an application's life cycle in terms of enabling comprehensive documentation of historic design decisions, perspectives that motivated these decisions and constraints that influenced the implementation of these decisions. Further, this algorithm-driven evaluation model helps in reviewing different netcentric scientific applications. It enables the application of a standard set of quality attributes

to analyze an application's stated requirements, mechanisms and implementation techniques.

In conclusion, we believe the evaluation model proposed in this paper greatly benefits the software engineering of netcentric scientific applications.

Bibliography

1. G. Abowd, L.Bass, P. Clements, R. Kazman, L. Northrop and A. Zaremski. "Recommended Best Industrial Practice for Software Architecture Evaluation." *Technical Report, Software Engineering Institute, Carnegie Mellon University*. CMU/SEI-96-TR-25. http://www.sei.cmu.edu/publications/documents/96.reports/96.tr.025.html. 1996.
2. L. Briand, J. Daly and J. Wuest. "A Unified Framework for Coupling Measurement in Object-Oriented Systems." *IEEE Transactions on Software Engineering*. 25:1, Jan/Feb, 1999. 91-121.
3. F. Buschmann, R. Meunier, H. Rohnertand and P. Sommerlad. "Pattern-Oriented Software Architecture: A System of Patterns." *Wiley*, New York. 1996. 476 pages.
4. E. Freeman, S. Hupfer and K. Arnold. "JavaSpaces™ Principles, Patterns and Practice." *Addison-Wesley*, Reading, Massachusetts. 1999. 344 pages.
5. E. Gamma, R. Helm, R. Johnson and J. Vlissides. "Design Patterns: Elements of Reusable Object-Oriented Software." *Addison-Wesley*, Reading, Massachusetts. 1995. 395 pages.
6. S. Ghosh and A.P. Mathur. "Issues in Testing Distributed Component-based Systems." *First ICSE Workshop on Testing Distributed Component-Based Systems*. Los Angeles, CA. May 1999. http://www.cs.purdue.edu/homes/ghosh/papers/icse99workshop-ghosh.ps.
7. S. Ghosh and A.P. Mathur. "On Error and Failures in Distributed Systems Built to CORBA and COM Standards." *Proceedings of the International Conference on Software Engineering and its Application*. Dec 1997. http://www.cs.purdue.edu/homes/ghosh/papers/osmania.ps.
8. A. Iannino. "Software Reliability Theory." *Wiley*, New York, 1237-1253.
9. R. Kazman, G. Abowd, L. Bass and P. Clements. "Scenario-Based Analysis of Software Architecture." *IEEE Software*, Nov 1996. 47-55.
10. R. Kazman, M. Barbacci, M. Klein and S.J. Carriere. "Experience with Performing Architecture Tradeoff Analysis." *Proceedings of ICSE99*, Los Angeles, CA. May 1999. 54-63. http://www.sei.cmu.edu/staff/rkazman/icse99.pdf.
11. R. Kazman, S.J. Carriere and S.G. Woods. "Toward a Discipline of Scenario-based Architectural Engineering." *Annals of Software Engineering*, Vol 9. 2000 (to appear). http://www.sei.cmu.edu/staff/rkazman/annals-scenario.pdf.
12. R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson and J. Carriere. "The Architecture Tradeoff Analysis Method." *Proceedings of ICECCS* 1998. 68-78.
13. M. Klein, T. Ralya, B. Pollack, R. Obenza and H. M. Gonzales. "A Practitioner's Handbook for Real-Time Analysis." *Kluwer Academic*. 1993.

14. S. Markus, S. Weerawarana, E.N. Houstis and J.R. Rice. "Scientific Computing via the Web: The Net Pellpack PSE Server." *Computational Science & Engineering*, Vol 4, No. 3, 1997, 43-51.
15. J. McCall. "Quality Factors." In J. Marchiniak, ed., *Encyclopedia of Software Engineering*, Vol 2, Wiley, New York, 1994. 958-969.