

# An NFR-Based Framework for Aligning Software Architectures with System Architectures

Nary Subramanian  
Dept. of Computer Science  
University of Texas at Tyler  
Tyler, TX 75799  
nsubramanian@uttyler.edu

Lawrence Chung  
Dept. of Computer Science  
University of Texas at Dallas  
Richardson, TX 75081  
chung@utdallas.edu

## Abstract

*System architectures (SysArch), usually developed during the requirements analysis phase of the information system development process, consider different ways of allocating the system requirements between hardware, software and the network. The requirements pertaining to software are then used to design the software subsystem and very often the first step in the design happens to be the development of the software architecture (SoftArch) for the system. In general, high quality software architecture is expected to result in the development of a high quality software system. Therefore, selecting appropriate software architecture seems to be extremely important for the organization. Usually the software requirements drive the selection of the software architecture – however, this approach ignores the effect of SysArch on SoftArch – for example, the choice of database software component in the SoftArch may be influenced by the hardware platform in the SysArch. Therefore ensuring alignment between the SoftArch and SysArch could result in an optimal final system. In this paper we use the Propagatory Framework for establishing the alignment between SoftArch and the SysArch. We demonstrate the practicality of the Propagatory framework by applying it to establish the alignment for a Home Appliance Control System (HACS).*

## Keywords

architecture, system, software, alignment, framework, appliance

## 1. Introduction

In a typical information system development process, after approval by the executive sponsors, the initial phases include scope definition, problem analysis and requirements analysis [1]. It is during the requirements analysis phase that the requirements of the new system

are elicited from the stakeholders and analyzed – the analysis includes, among other aspects, development of system architectures (SysArch) which considers allocation of the requirements between hardware, software and the network. Once the optimal system architecture is selected the next step is to develop the hardware, software and the network elements of the system almost independently of each other. Among these elements software development is usually the most important since the development of software seems to consume most resources in terms of budget, schedule, and man power [1]. The requirements pertaining to software are used to design the software subsystem and very often the first step in the design happens to be the development of the software architecture for the system. Software architecture (SoftArch) is the high level viewpoint of the software to be developed and at the least, a software architecture consists of components, connectors, constraints, styles and patterns [2, 3]. Most of the properties of the final software are considered to have been designed into the software at the software architecture stage itself and therefore, in general, high quality software architecture is expected to result in the development of a high quality software system. Therefore, selecting an appropriate software architecture seems to be important - usually the software requirements drive the selection of the software architecture from among competing architectures but this approach ignores the effects of the SysArch on SoftArch including:

1. ensuring that the components selected for the software architecture can execute on the hardware selected in the SysArch: for example, the database component should work on the hardware server in the SysArch
2. ensuring that the components selected for the software architecture can execute on the software environment determined in the SysArch: for example, the web server may need to run on the chosen operating system
3. ensuring that the connectors between software architecture components can be mapped to the

appropriate network technology – thus if the software components communicate using signaling channel 7 protocol then the network technology in SysArch should not insist on TCP/IP.

Therefore it is in the interests of the project and the organization that there is a close alignment between SoftArch and SysArch. The advantages of this alignment include:

1. ensuring SoftArch satisfies software requirements
2. ensuring SoftArch satisfies the project goals captured by the SysArch
3. ensuring SoftArch satisfies the organization's goals captured by the Enterprise Architecture [4] to which the SysArch is usually traceable.

In this paper we use a framework called the Propagatory Framework [10] to track the alignment between SoftArch and SysArch and includes the following steps:

1. decompose the system requirements
2. decompose the software requirements
3. decompose the candidate software architectures
4. determine the contributions of the software architecture decomposition to the software requirements decomposition
5. determine the contributions of the software architecture decomposition to the system requirements
6. Select the software architecture that best meets the requirements in Steps 4 and 5.

The words “decompose” and “contribution” are derived from the NFR Framework [5] – the Propagatory Framework uses the NFR Framework. The NFR Framework is a process-oriented and goal-oriented framework and captures the goals of the organizations and their realizations by means of various softgoals and their decompositions. Alignment is captured by means of contributions and their justifications in the NFR Framework and the NFR Framework provides rules to propagate the contributions to the topmost softgoal in the decomposition hierarchy. The Propagatory Framework systematically applies the NFR Framework concepts repeatedly to determine the alignment (or traceability) between the SoftArch and SysArch, and derives its name from the fact that the propagation rules of the NFR Framework are used to determine the achievement (or its lack) of the alignment between the SoftArch and the SysArch.

There seem to be few approaches to establish the traceability between SoftArch and SysArch. The

Strategic Alignment Method (SAM) [6] reinforced the importance of alignment for strategic management. Four cross-domain relationships were determined:

1. strategic execution alignment perspective – how business strategy affects organizational infrastructure and IS infrastructure design
2. technology information alignment perspective – how business strategy affects IT strategy and IS infrastructure
3. competitive potential alignment perspective – how IT strategy affects business strategy and organizational infrastructure
4. service level alignment perspective – how IT strategy affects IS infrastructure and organizational infrastructure.

However, SAM does not describe how to actually determine these perspectives. Information Economics [7] provides a framework for determining which projects are worth immediate funding by evaluating projects against a list of ten decision factors that encompasses both the business and IT domains that managers deem important for the business and then determining which project(s) score the highest. However, the reasoning behind the ten decision factors seems somewhat subjective and there is no way to actually establish the traceability between the system architecture and the enterprise architecture in this approach. Business Process Re-engineering (BPR) has attempted to align IT with business process [8] – however, BPR does not attempt to establish alignment between system architecture and enterprise architecture. Business IT Alignment Method (BITAM) [4] is an interesting method that helps to align the system architecture (as represented by the IT architecture) with the enterprise architecture (as represented by the business architecture) – BITAM uses twelve steps to establish mappings between three layers of a business system: business model layer, business architecture layer, and IT architecture layer; any misalignments are due to improper mappings between the layers and need to be restored. However, BITAM does not seem able to capture justifications for decisions and an ability to quickly adapt to changed business goals.

The Propagatory Framework is applied to determine the traceability (or alignment) of the SoftArch to the SysArch of a Home Appliance Control System (HACS). A Home Appliance Control System (HACS) helps to control appliances in the home such as the garage door, stove, refrigerator, air-conditioner, and the like from one place either inside or outside the home [9]. The lessons learned from this application of the Propagatory Framework are documented in detail.

This paper first describes the NFR Framework briefly in Section 2, in Section 3 the Propagatory Framework is applied to the HACS system, Section 4 documents the observations and the lessons learned from using the Propagatory Framework, and Section 5 gives the conclusions of this paper.

## 2. The NFR Framework

The NFR Framework [5] is briefly described in this section. The NFR Framework, where NFR stands for Non-Functional Requirements, requires the following interleaving tasks, which are iterative:

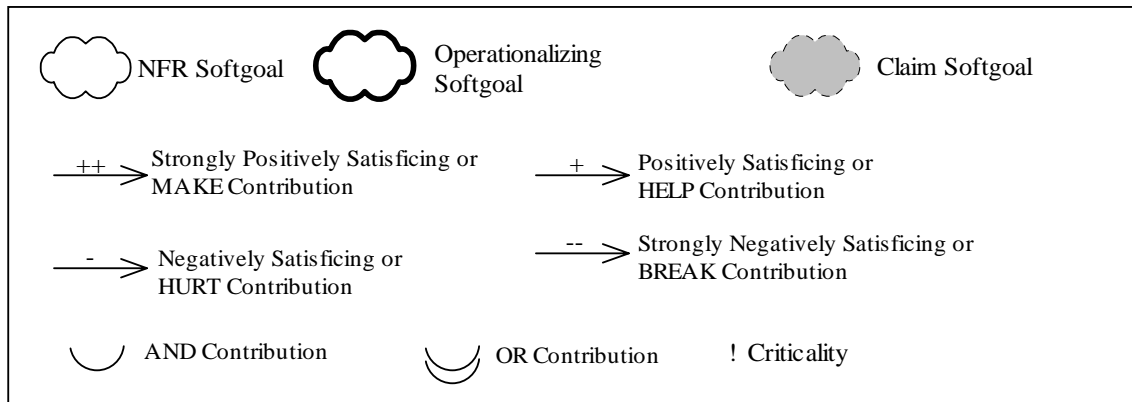
1. Develop the NFR softgoals and their decomposition.
2. Develop operationalizing softgoals and their decomposition.
3. Determine contributions between operationalizing softgoals and NFR softgoals.
4. Develop goal criticalities.
5. Evaluation and analysis.

The graph that results from the application of above steps is called the Softgoal Interdependency Graph (SIG). The NFR Framework uses the concept of goal satisficing. The notion of goal satisficing of the NFR Framework assumes that decisions taken during the development process usually contribute only partially

(or against) a particular goal, rarely “accomplishing” or “satisfying” goals in a clear-cut sense. Consequently any model is expected to satisfy NFRs within acceptable limits, rather than absolutely. There are different degrees of satisficing and the degrees are indicated by arrows annotated with + or – symbols. In the NFR Framework, each requirement (either system requirement or software requirement) is called an NFR softgoal (depicted by a cloud), while each element of the SoftArch or SysArch is called an operationalizing softgoal (depicted by a dark cloud). The rationale for various decisions is captured by yet another softgoal – the claim softgoal (depicted by a cloud with dashed border). The partial ontology of the NFR Framework is given in Figure 1 and the steps to use the NFR Framework are described below.

During the NFR goal decomposition process, the NFR softgoals are decomposed into their constituent softgoals based on the domain – in this paper we treat the requirements for the SysArch and the requirements for the SoftArch as NFR softgoals. This decomposition is not unique and depends on what the people performing the decomposition consider important for the domain at that point in time. The softgoals are named using the following convention:

*Type[Topic1, Topic2, ...]*



**Figure 1. Partial Ontology of the NFR Framework**

where *Type* is an NFR and *Topic* is the system or domain to which the *Type* applies. The NFR softgoals can be related to each other by three contributions: AND-contribution (indicated by single arc), OR-decomposition (indicated by double arc), and a refinement (only one child NFR softgoal). The various elements of the SoftArch and the SysArch are considered as operationalizing

softgoals in the NFR Framework. Some NFR softgoals may be declared critical or priority softgoals by annotating them with ‘!’ marks and this is done during step 3 of the NFR Framework application process. During NFR softgoal satisficing determination we determine the contributions made by the operationalizing softgoals to the various NFR softgoals – these contributions can be of four types: MAKE, HELP, HURT, and BREAK. These contributions have the following ranking:

MAKE > HELP > HURT > BREAK.

During the final step – the evaluation and analysis step – we apply the propagation rules of the NFR Framework to determine to what extent the models satisfy the NFR softgoals. While detailed propagation rules may be seen in [5], in this paper we will be using the following simplified propagation rules:

- R1. If most of the contributions received by a leaf NFR softgoal are positive (MAKE or HELP) then that leaf NFR softgoal is considered satisfied.
- R2. If most of the contributions received by a leaf NFR softgoal are negative (BREAK or HURT) then that leaf NFR softgoal is considered denied or not satisfied.
- R3. In the case of priority softgoals, or when there is a tie between positive and negative contributions, the system architect or the developer can take the decision based on or a variation of R1 and R2
- R4. In the case of AND-contribution, if all the child softgoals are satisfied then the parent NFR softgoal is satisfied; else the parent softgoal is denied.
- R5. In the case of OR-contribution, if at least one child softgoal is satisfied then the parent NFR softgoal is satisfied; else the parent softgoal is denied.
- R6. In the case of refinement (only one child) the parent is satisfied if the child is satisfied; and the parent is denied if the child is denied.

Upon applying these propagation rules, if the root NFR softgoals are satisfied then the goals of that SoftArch or SysArch have been met to a large extent. Throughout the SIG development, the rationales for the various contributions are captured by claim softgoals.

### 3. Application of the Propagatory Framework

The steps of the Propagatory Framework listed in the Introduction will be applied to the HACS system. The system requirement for HACS are given in Figure 2.

#### 3.1 Decomposition of System Requirements

The decomposition of the system requirements for HACS is given in Box 1 on the upper left part of Figure 5. The decomposition is based on the system requirements given in Figure 2. The requirements for HACS system represented by the NFR softgoal System Requirements[HACS] is AND-decomposed (indicated by the single arc) into two NFR child softgoals – Non-Functional Requirements [System, HACS] and Functional Requirements[System, HACS], which represent, respectively, the non-functional requirements and the functional requirements of the

HACS system. The NFR softgoal Non-Functional Requirements[System, HACS] is AND-decomposed into Extensible[HACS], Reliable [HACS], Performance[HACS], and Secure[HACS], which, respectively, refer to the four main non-functional requirements (requirements 8, 9, 10, and 11 of Figure 2) of the HACS system. The NFR-softgoal Functional Requirements[HACS] is AND-decomposed into Access[Appliance Status], Ethernet Connection, Configure[Appliance], Control[Appliance], and Support Apache Web Server, that refer, respectively, to the system requirements 1, 6, 2, 3, and 7.

#### 3.2 Decomposition of Software Requirements, Software Architectures, and Contributions

Steps 2, 3 and 4 of the Propagatory Framework are described here. The software requirements for a typical HACS system are given in Figure 3. Based on the software requirements, the software design takes place. The first step in the design process is the development of the software architecture which is a high-level viewpoint of the software system. Many of the properties of the final software system are usually decided at the software architecture stage itself. The constituents of a software architecture include components, connections, constraints, patterns, and styles [2,3]. Components are the elements from which systems are built; connections are the interactions between the elements; patterns describe the layout of the components and connections; constraints are on the components, connections and patterns; and styles are an abstraction of architectural components from various architectures. We consider two alternative architectures developed for meeting the requirements of HACS software including:

1. Web-based Java Architecture
2. Web-based ASP.NET Application.

The components and connections of these architectures are given in Figure 4. The web-based Java software architecture has Java classes for components, uses MySQL database, interfaces with the database using JDBC (Javabean DataBase Connectivity), and provides browser access using applets. The connections include messages between objects, SQL over TCP/IP connection to the database, and TCP/IP connections to the appliances. The constraints on this architecture include the requirement for a JRE (Java Runtime Environment), the pattern in this architecture is that of communicating objects, while the style is object-oriented.

1. The system should permit remote access of each of the appliances in the HACS.
  - 1.1 Access should include real-time status access of each appliance in the HACS.
  - 1.2 Access should be possible from both inside and outside the home.
  - 1.3 External access should be possible through the internet.
2. The system should permit remote configuration of each of the appliances in the HACS.
  - 2.1 Configuration should be possible based on prior settings.
  - 2.2 Configuration should be possible from both inside and outside the home.
  - 2.3 External configuration should be possible through the internet.
3. The system should permit remote control of each of the appliances in the HACS.
  - 3.1 Control should include all aspects of the appliance including starting and stopping.
  - 3.2 Control should be possible from both inside and outside the home.
  - 3.3 External control should be possible through the internet.
4. The system should interconnect the following appliances: microwave, refrigerator, air-conditioner, and the stove.
5. The system should provide a means to access, configure, and control the appliances both locally and remotely.
6. All connections should be over Ethernet.
7. The web server should preferably be Apache server.
8. The system should be secure.
9. The system should have high performance.
10. The system should be reliable.
11. The system should be extensible.

**Figure 2 . System Requirements for HACS**

1. The software should permit local and remote access of each of the appliances in the HACS.
  - 1.1 Access should include real-time status access of each appliance in the HACS.
2. The software should permit local and remote configuration of each of the appliances in the HACS.
  - 2.1 The configuration should be possible based on prior settings.
3. The software should permit local and remote control of each of the appliances in the HACS.
  - 3.1 The control should include all aspects of the appliance including starting and stopping.
4. The software should provide web browser-based remote and local access, configuration and control facility.
5. The software should store all the appliance details and the user details.
6. The software should be secure.
  - 6.1 There should be a means for user authentication.
7. The software should have fast response.
8. The software should be reliable.
  - 8.1 Software should bring to the notice of the user any appliance failure.
  - 8.2 Software should itself be robust - any internal failures should be handled gracefully.
9. The software should be extensible
  - 9.1 The software should be updatable
  - 9.2 The software should be modifiable.
  - 9.3 The software should be reconfigurable.

**Figure 3 . Software Requirements for the HACS Software**

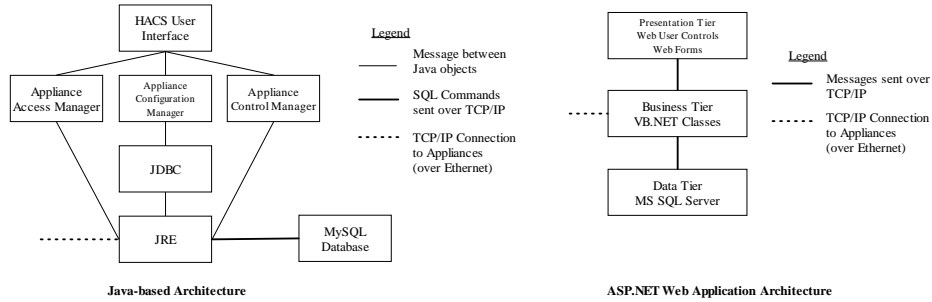
The web-based ASP.NET application uses ASP.NET classes for the presentation layer, Visual Basic.NET classes for the business layer, and Microsoft SQL database for the data layer. The constraint is the need for .NET Framework, the pattern is that of communication between adjacent layers, while styles include both layered and client-server based.

The decomposition of the requirements for HACS is shown in the upper part of Box 2 (the box on the right) in the SIG of Figure 5. The decomposition is based on the software requirements given in Figure 3. The bottom part of Box 2 of Figure 5 gives the decompositions of the two architectures considered in Figure 4. Figure 5 indicates only pertinent negative and positive contributions (to avoid clutter) between the architectural elements and the software requirements and the justifications for these contributions are captured by the claim softgoals (indicated by the shaded cloud shapes) – all the other contributions that

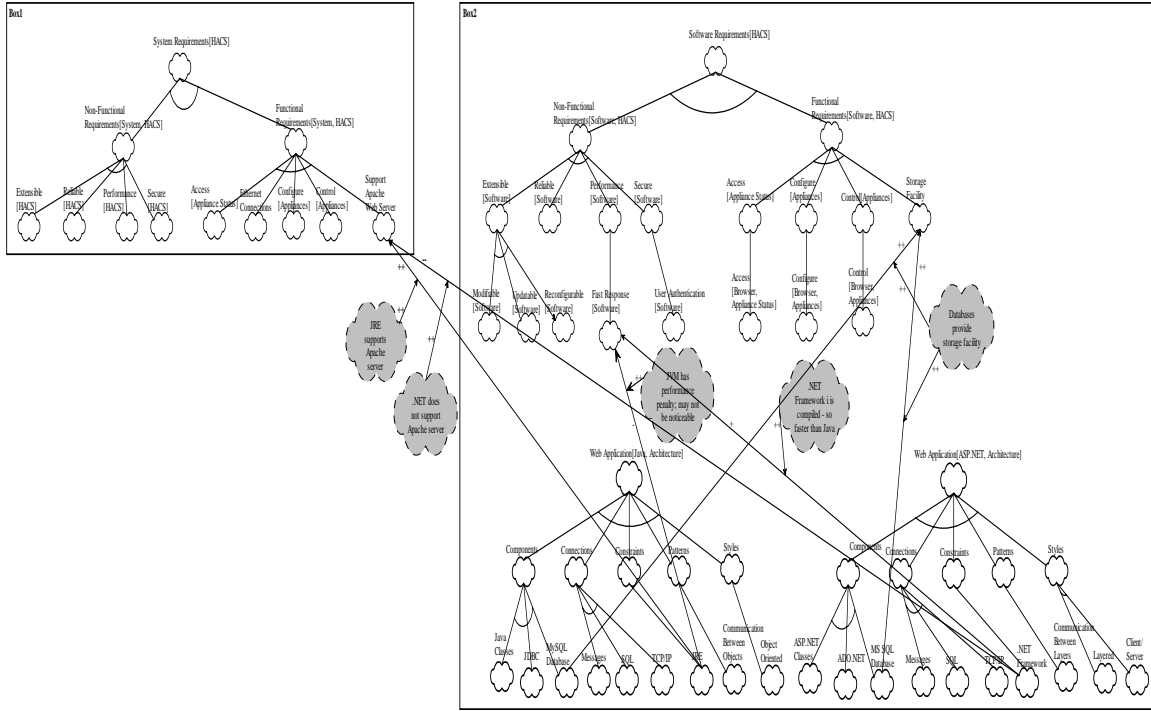
are not shown are all positive (either MAKE or HELP contributions). Applying the propagation rules we can conclude that Java-based architecture does not fully satisfy the software requirements. However, the developer (or analyst) will need to consider the effect of system requirements before choosing the final architecture and this is discussed next. Either way, we now know why (or why not) software architecture meets (or does not meet) the requirements.

### **3.3 Determination of Contributions between Software Architectures and System Requirements**

The contributions between software architectures and system requirements is shown by the contributions (arrows) between Box1 and Box2 in Figure 5 and the justifications for the contributions are given by the claim softgoals.



**Figure 4. Candidate Software Architectures for HACS**



**Figure 5. SIG with System and Software Requirements Decomposition and Contributions**

### 3.4 Selection of Software Architectures

Table 1 gives the options before the analyst based on the SIG of Figure 5. As can be observed, by maintaining close alignment between software architectures and system requirements the analyst has a better choice.

## 4. Observations

To keep our discussions simple we have assumed all softgoals (NFR, operationalizing, and claim softgoals) to be of the same priority in the SIGs. This need not be the case since the NFR Framework allows for consideration of propagations based on variable priorities as well – in practice variable priorities may be the norm and this can be accommodated by small changes to the propagation rules.

The Propagatory Framework captures all justifications using the claim softgoals – this helps preserve a historical record of important decisions. Also any changes in the goals and/or architectures can be captured by quickly updating the SIG and recording the justifications for the changes. To avoid clutter in the decomposition SIGs only the pertinent contributions were shown since they help appreciate the value of the Propagatory Framework. However, this does not mean that positive contributions clutter up the SIG and that they are worthless; in fact, a primary advantage of using the NFR Framework lies in capturing the justifications for the various contributions be they positive or negative by means of the claim softgoals. A software program that captures the various SIG elements will make this job easier for the analyst.

**Table 1. Results of the Evaluation of Alignment Between SoftArch and SysArch**

<b>Requirements Satisficing</b>	<b>Software Architecture Selected</b>
<b>System requirements not considered</b>	ASP.NET Architecture chosen
<b>System requirements more important than software requirements</b>	Java-based Architecture chosen.
<b>System requirements either equally or less important than software requirements</b>	ASP.NET Architecture chosen.

## 5. Conclusions

System requirements drive the development of system architectures (SysArch) that allocates the requirements between software, hardware and network elements. Subsequently the software requirements drive the development of software architectures (SoftArch), which is a high level view of the software system. Usually, the software architectures are selected based on the software requirements – however, consideration of the alignment between software architectures and system architectures allows the developer to make a better choice of software architectures. In this paper we used the Propagatory Framework [10] for establishing alignment between SoftArch and SysArch. The Propagatory Framework is based on the NFR Framework [5] and has six steps to establish the alignment. In order to demonstrate the application of the Propagatory Framework we applied this Framework to the development of the Home Appliance Control System (HACS) [9]. HACS helps to control home appliances from a central location either inside the home or remotely via the internet as well. This application of the Propagatory Framework demonstrated its practicality and some of its advantages.

Further work needs to be done in order to validate the applicability of the Propagatory Framework – we believe that it needs to be applied to determine alignment between SoftArch and SysArch for several different systems. Another area of future work is to develop means of documenting unambiguously both the SysArch and the SoftArch – in this paper we have used box-and-line diagrams to document software architectures: but ADL’s (Architecture Description Languages [2]) may be a better way to document software. Yet another research area would be to develop a software tool to guide the analyst or developer in determining the alignment between SoftArch and SysArch. However, in our opinion, the Propagatory Framework provides a systematic framework for establishing traceability between software architectures and system architectures.

## References

- [1]. J. L. Whitten and L. D. Bentley, *Systems Analysis and Design Methods*, McGraw-Hill/Irwin Publication, New York, 7<sup>th</sup> Edition, 2006.
- [2]. M. Shaw, and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996.
- [3]. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, SEI Series in Software Engineering, Addison-Wesley, 1998.
- [4]. H-M Chen, R. Kazman, and A. Garg, “BITAM: An Engineering-Principled Method for Managing Misalignments Between Business and IT Architectures”, *Journal of Science of Computer Programming*, July 2005, Vol. 57, No. 1, pp. 5-26.
- [5]. L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, Boston, 2000.
- [6]. J. Luftman, P.R. Lewis, and S. H. Oldach, “Transforming the Enterprise: The Alignment of Business and Information Technology Strategies”, *IBM Systems Journal*, Vol. 32, No. 1, 1993.
- [7]. M. Parker, R. Benson, et al, *Information Economics: Linking Business Performance to Information Technology*, Prentice Hall, Englewood Cliffs, NJ, 1998.
- [8]. M. Hammer and J. Champy, *Reengineering the Corporation*, Harper Collins Publishers, New York, 1993.
- [9]. [http://feminity.toshiba.co.jp/feminity/feminity\\_eng/about/index.html](http://feminity.toshiba.co.jp/feminity/feminity_eng/about/index.html)
- [10]. N. Subramanian, L. Chung, and Y-t Song, “An NFR-Based Framework to Establish Traceability Between Application Architectures and System Architectures” to appear in the *Proceedings of the 7<sup>th</sup> ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2006)*, IEEE Computer Society, Las Vegas, June 2006.