Submitted to Fourth Workshop Iberoamerican on Requirements Engineering and Software Environments (IDEAS'2001), San José, Costa Rica, April, 2001.

Using Non-functional Requirements to Select Components: A Formal Approach

Carina F. Alves, Nelson S. Rosa, Paulo R. F. Cunha, Jaelson F. B. Castro
Centro de Informática
Universidade Federal de Pernambuco
50732-970 Recife, Pernambuco - Brazil
Phone: +55 81 3271 8430 Fax: +55 81 3271 8438
E-mail: {cfa,nsr,prfc,jbc}@cin.ufpe.br

George R. R. Justo
Centre for Parallel Computing
University of Westminster
115 New Cavendish Street London - W1M 8JS
Phone: +44 171 911 5000 Fax: +44 171 911 5143
E-mail: justog@cpc.wmin.ac.uk

Abstract

The increasing in size and complexity of software systems has motivated their development based on the reuse of already built COTS (Commercial Off-The-Shelf) components. An important task of this approach addresses the selection process of suitable COTS to make up the final product. We argue that the selection process may be improved if it is performed both considering non-functional requirements and adopting a formal approach. All the elements that take part in the selection process, namely non-functional requirements, software architecture concepts and COTS, are specified using Z. In order to illustrate our approach, we apply it to an Internet information system.

1 Introduction

The increasing in the size and complexity of software systems has motivated their development through planned and deliberated reuse of previsouly constructed components or COTS (Commercial Off-The-Shelf) [25]. This approach, known as component-based development, has potential advantages such as improve productivity, shorter the time to market, lower prices and higher quality solutions. On the other hand, the use of COTS components introduces new problems and risks, including difficulty in selecting suitable components and insufficient requirements analysis.

Non-functional requirements[1] (NFRs) [3] play an important role during the component-based development [4, 5]. In particular, the analysis of quality requirements such as performance and security can improve the selection process through discrimination between competing COTS products. For instance, if two components execute the same task (i.e., they have similar functionality), non-functional attributes may be used in the selection process as an extra and decisive criteria. In this particular case, a component with a better performance may be choosen rather than a slow one.

Despite the importance of the NFRs, they have not received the same attention as functional ones. There are some reasons that can help us to understand why these requirements are not explicitly dealt with: NFRs are usually very abstract and stated only informally, e.g., "the system must have a satisfactory performance" or "the component is safe" are common descriptions of NFRs; NFRs are rarely supported by languages, methodologies and tools; NFRs are more complex to deal with; and very often NFRs conflict and competing with each other, e.g., availability and performance.

These problems are usually found in the traditional development, in which requirements are elicited from users for a bespoke software product. In a different situation, when requirements need to be elicited from users for a commercially available software component, issues become more complex. The main reason for this is the introduction of a third party i.e., vendor of a software component [4]. As a consequence, it is difficult to assess the product's quality attributes from vendor's descriptions.

[1] Also Refered to global requirements, quality requirements, goals, constraints, software quality attributes, system-wide properties or *ilities*.

In particular, most current methods for COTS selection have not adequately treated the NFRs. Kunda [12] presents a social technical approach to COTS evaluation, while Kontio [10] proposes a method that takes into account functional requirements and cost estimation. Another approach proposed by Ncube [14] covers requirements acquisition for COTS products selection but does not address quality attributes.

In this paper, we define a component selection process in which the treatment of NFRs plays a key role. Components that made up a software architecture are matched against COTS components which are stored in a COTS library. NFRs assigned to software architecture components are used to select the COTS which best satisfy them. As a basis for this selection process, all these elements are formally defined in the Z [22].

This paper is organised as follows. Section 2 introduces the basic concepts of software architecture, NFRs and tasks related to the selection process. Next section, Section 3 presents how to formalise the elements presented in the previous section. An example is presented in Section 4 in order to show how to apply our approach in a practical example. Finally, last section presents the conclusion and future direction of this work.

2 Background

Prior to present the proposed formalisation, it is necessary to introduce basic concepts of NFRs, software architecture and a brief description of the selection process of COTS. These elements are presented in the following subsections.

2.1 NFRs

Functional requirements define *what* a software is expected to do, while non-functional requirements (NFRs) specify global constraints on *how* the software operates or how the functionality is exhibited [2, 3]. Functional requirements usually have localised effects, i.e., they affect only the part of the software addressing the functionality defined by the requirement. During the software development process, functional requirements are usually incorporated into the software artefacts step by step. At the end of the process, all functional requirements must have been implemented in such way that the software satisfies the requirements defined at the early stages. NFRs, however, have a global nature, which means that to satisfy a NFR may affect several design components. Thus, NFRs are not implemented in the same way as functional ones.

NFRs have very a distinctive nature, in which a wide variety of aspects such as modifiability and fault-tolerance are categorised as non-functional properties. The IEEE/ANSI 830-1993, IEEE Recommended Practice for Software Requirements Specifications defines thirteen non-functional requirements that must be included in the software requirements document: performance, interface, operational, resource, verification, acceptance, documentation, security, portability, quality, reliability, maintainability and safety. Kotonya [11] classifies these requirements into three main categories: Product requirements, Process requirements and External requirements. Product requirements specify the desired characteristics that a system or subsystem must possess. Process requirements put constraints on the development process of the system. External requirements are constraints applied to both the product and the process which are derived from the environment where the system is developed.

Another classification of NFRs separates them into consumer-oriented and technically-oriented. Consumer-oriented refers to NFRs observable by the consumers, e.g., performance. Technically-oriented defines the NFRs that are of interest for the system development such as evolvability, openness and so on. A more complete list of NFRs was defined in [23]: *performance*, *affordability*, *usability*, *understandability*, *simplicity*, *stability*, *reliability*, *availability*, *fault tolerance*, *scalability*, *evolvability*, *continuous evolution*, *openness*, *seamlessness*, *flexibility*, *configurable*, *adaptable*, *security*, *safety*, *trust*, *high confidence*, *information assurance*, *survivability*, *timeliness*, *real-time*, *mobility*, *QoS* and *system monitoring*. It is worth noting that the terms QoS and NFRs are sometimes used interchangeably. However, the former is a subset of the latter.

2.2 Software Architecture

The software architecture [17, 21] is the highest abstract description of a software design, which is defined at the initial stages of the software development. Software architectures are commonly described in terms of three basic abstractions: components, connectors and configurations. Components represent a wide range of different elements, from a single client to a database, and have an interface (made up of ports) used to communicate the component with the external environment. Connectors represent communication elements between components. The configuration describes how components and connectors are wired. A traditional view of software architecture is shown in Figure 1.

Essentially, the software architecture presents a description of the software where "computation" (included in components) and "communication" (modelled by connectors) are clearly separated. In order to describe software architectures, languages specially designed for this purpose, namely Architecture Description Languages (ADLs), replace box-and-arrows diagrams or natural languages. This class of languages has some key characteristics such as components and connectors as first-class elements and the focus on the structure of the software architecture [13].

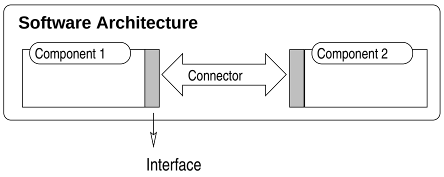

Figure 1: Software Architecture

2.3 COTS Selection

The nature of COTS suggests that the model of component-based software development should be different from the conventional development model. This fact results in a significant paradigm shift from the development-centric toward a procurement-centric approach [24], which is based on the acquisition of components available in the market and integrated into a software solution (as shown in Figure 2).

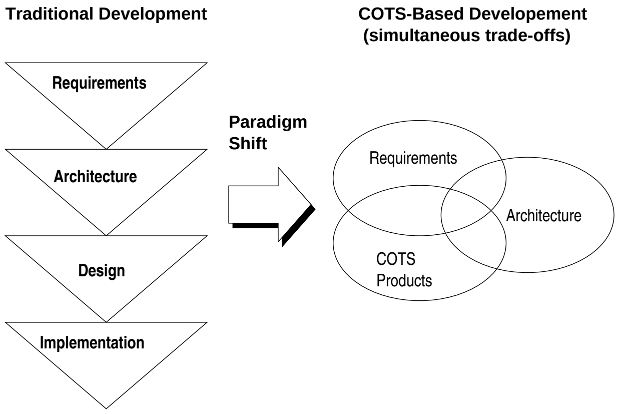

Figure 2: Paradigm Shift

In traditional approaches, the system development often follows an strict sequence of activities. Unlike traditional development, in the COTS-based approach there are simultaneous trade-offs among user requirements, system architecture and COTS products. The selection of adequate components is often a non-trivial task and all these aspects should be carefully considered. As a result, the customers must accept the possibility that the resulting system might be a compromise among these concerns [7].

It has been argued that a well defined selection process is the cornerstone for any effective COTS development [10, 14]. However, the COTS selection process is prone to some potential problems. The black-box nature of the COTS components difficulties the understanding of their internal behaviour and therefore the evaluation process becomes harder. Another problem is concerned with the definition of the evaluation criteria. This process essentially decomposes the requirements for the COTS package into an hierarchical criteria set, which usually include: component's functionality, quality aspects, architecture constrains, and non-technical factors (i.e. vendor guaranties, legal issues). During the evaluation process, the properties of each candidate component are identified and assessed according to the evaluation criteria.

The success of a component-based development process largely depends on the accurate understanding of the capabilities and limitations of the individual candidate products. At a first stage, the products are evaluated from their compliance with functional requirements, since these requirements are easier to elicit and analyse. Then, the evaluation process is performed based on non-functional requirements, which are essential aspects for the negotiation and priorization of competing components. Most COTS methods treat quality aspects only during the latter phases of development. This attitude increases the risks of COTS failure and the costs of the final system. However, Chung [3] emphasises that non-functional requirements should be addressed as early as possible in the systems lifecycle.

3 Formal Selection

The selection process is performed from the formal description of the software architecture, COTS and their NFRs. Figure 3 shows an overview of the main elements of our approach. The software architecture contains the description of NFRs in addition to the traditional architectural elements, components, connectors and the configuration (see Figure 1). The COTS library contains a set of COTS available to the selection. The selection process defines, according to the software architecture, which COTS will be used to build the integrated system or final product.

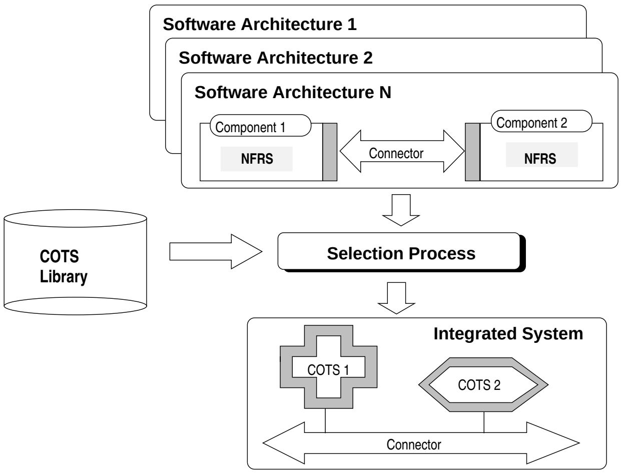

Figure 3: Overview of the Selection Process

Prior to advance on the formalisation, it is necessary to explain some important points about Figure 3. Firstly, a very abstract software architecture is refined into more concrete ones. In terms of development, it means that most concrete software architecture represents the last representation of the software before its realisation into an operational environemtn. Secondly, a component in the architecture has a direct mapping with the COTS in the integrated system [15], according to the selection process. Thirdly, we do not consider connector in the selection process, as they have not usually a one-to-one mapping with the COTS []. Fourthly, the COTS library is obtained by an analisys of products available in the market.

3.1 NFRs Formalisation

In order to fomalise the NFRs, it is adopted the Parmenides framework [20] that explictly considers NFRs within the entire software development process. The framework defines precisely how NFRs are expressed and integrated into an architecture-based development. In Parmenides, NFRs are viewed as constraints on possible design decisions taken for implementing a software system. It means that every decision taken for implementing a functionality in the software must respect the constraints imposed by the non-functional part of the requirements. Furthermore, these constraints are often correlated [3] and their correlation must be also respected.

Parmenides introduces the notion of non-functional information, which is a generic term used to refer to either the non-functional requirement itself or the non-functional properties. Every non-functional information is modelled in the Parmenides by three abstractions: NF-Attribute, NF-Realisation and NF-Requirement. A NF-Attribute models both non-functional characteristics of the software that can be precisely identified (*performance*) and any non-functional feature that cannot be quantified, but may be defined as required in the software in a certain level (*security*). A NF-Realisation models any implementation element or design decision that affects a NF-Attribute. NF-Attributes (NF-Realisations) may be decomposed into sub-attributes (sub-realisations), all its sub-attributes (sub-realisations) are either necessary to achieve it (*and* contribution) or at least one of the sub-attributes is necessary (*or* contribution) to accomplish the NF-Attribute.

Following these definitions, it has been defined given sets (*ID_ATTRIBUTES* and *ID_REQUIREMENTS*) and free types (*TYPE_CONTRIBUTION*, *Range_Value* and *Priority_Degree*) for modelling basic types related to the specification of NFRs.

$$
\begin{array}{l}
[ID\_ATTRIBUTE, ID\_REQUIREMENT] \\
TYPE\_CONTRIBUTION ::= AND \mid OR \mid none \\
Range\_Value ::= level\_1 \mid level\_2 \mid level\_3 \\
Priority\_Degree ::= very\_high \mid high \mid moderate \mid low
\end{array}
$$

The formal specification of a NF-Attribute is defined as the *NF_Attribute* schema

$$
\begin{array}{|l}
\hline
NF\_Attribute \\
\hline
subattributes : \mathbb{P} ID\_ATTRIBUTE \\
contribution : TYPE\_CONTRIBUTION \\
assignment : Range\_Value \\
\hline
subattributes = \emptyset \Rightarrow contribution = none \\
\hline
\end{array}
$$

A NF-Attribute, as mentioned before, may be defined as a composition of sub-attributes that contributes to the attribute being defined. *subattributes* represent these elements, while *contribution* defines the kind of contribution of the sub-attributes to the *NF_Attribute*. *assignment* models the values that can be assigned to the attribute, e.g., *30 transactions per seconds* (*Peformance*) or *Security Level 1* (*for security*). Finally, if the attribute has not subattributes, it means that there is not any kind of contribution (modelled by *none*).

All the NF-Attributes are stored in a library of attributes (*Attributes_Library* schema) for future reuse.

$$\begin{array}{|l} \textit{Attribute\_Library} \\ \hline attributes : ID\_ATTRIBUTE \nrightarrow NF\_Attribute \\ \hline \end{array}$$

A NF-Requirement is modelled by the *NF_Requirement* schema.

$$\begin{array}{|l} \textit{NF\_Requirement} \\ \hline \Xi Attribute\_Library \\ id : ID\_REQUIREMENT \\ atts : ID\_ATTRIBUTE \nrightarrow NF\_Attribute \\ predicate : ID\_ATTRIBUTE \nrightarrow Range\_Value \\ priority : Priority\_Degree \\ \hline \quad \mathrm{dom}\ atts \subseteq \ \mathrm{dom}\ attributes \\ \quad \mathrm{dom}\ predicate \subseteq \ \mathrm{dom}\ attributes \\ \hline \end{array}$$

Every NF-Requirement has an identification, *id* (e.g., *High_performance*). It is necessary to define the set of attributes (*atts*) being constrained by the NF-Requirement, e.g., the NF-Requirement *secure* constrains the NF-Attribute *security*, while the NF-Requirement *High_Performance* is a constraint over the NF-Attribute *performance*. The most important part of the NF-Requirement definition is the *predicate*, which assigns a value in the range of the possible values assigned to the attribute. Finally, in the selection process is necessary to define the priority (*priority*) of the NF-Requirement. NF-Requirements with high priority are mandatory to be satisfied.

3.2 Software Architecture Formalisation

Different formal notations and models have been used to for describing software architectures, such as Z [1, 16], LOTOS [19] and CHAM (CHemistry Abstract Machine) [8]. In our approach, we adopt the Z notation for describing the software architecture, like Garlan. In particular, the Garlan approach is extended in order to enable us to assign NFRs to architectural elements. Architectural elements, namely components, connectors and configuration, are modelled as Z schemas.

A component is modelled by the *Component* schema which consists of a *description*, an *interface* (set of ports) and the NF-Requirements (*nfrs*).

$$\begin{array}{|l} \textit{Interface} \\ \hline ports : \mathbb{P} ID\_PORT \\ \hline ports \neq \emptyset \\ \hline \end{array}$$

$$\begin{array}{|l} \textit{Component} \\ \hline comp\_description : DESCRIPTION \\ interface : Interface \\ nfrs : NF\_Requirement \\ \hline \end{array}$$

Like components, a connector is specified as a Z schema, namely *Connector* schema. As the connector links two components, the role (*roles*) specifies the behaviour of a single component in the interaction. It is worth noting that there are many questions related to the connector's roles, but they are not in the scope of this paper.

$$\begin{array}{|l} \textit{Connector} \\ \hline connect\_desc : DESCRIPTION \\ roles : \mathbb{P} ROLE \\ \hline \end{array}$$

Finally, the software architecture is defined as a set of *components*, *connectors* and their connections (*attachments*).

$$
\begin{array}{|l}
\hline
\textit{Software\_Architecture} \\
components : ID\_COMPONENT \rightarrow\!\!\!\!\!\!+ \; Component \\
connectors : ID\_CONNECTOR \rightarrow\!\!\!\!\!\!+ \; Connector \\
attachment : (ID\_COMPONENT \rightarrow\!\!\!\!\!\!+ \; Component) \rightarrow\!\!\!\!\!\!+ \; (ID\_CONNECTOR \rightarrow\!\!\!\!\!\!+ \; Connector) \\
\hline
\mathsf{dom}\ (\bigcup(\ \mathsf{dom}\ attachment)) \subseteq\ \mathsf{dom}\ components \\
\mathsf{dom}\ (\bigcup(\ \mathsf{ran}\ attachment)) \subseteq\ \mathsf{dom}\ connectors \\
\hline
\end{array}
$$

3.3 COTS Formalisation

The COTS is specified by the *Cots* schema. The *Cots* includes the *version*, the *company* which develops the COTS, the operational systems under which the COTS runs (*operating_system*), an *interface* (similar to the components of the previous section) and the set of NFRs (*nfrs*).

$$
\begin{array}{|l}
\hline
\textit{Cots} \\
version : VERSION \\
company : ID\_COMPANY \\
operating\_system : ID\_OS \\
license : License \\
interface : Interface \\
nfrs : NF\_Requirement \\
\hline
\end{array}
$$

An additional element presents in the COTS specification is the *License*. The *License* has important information that refers to legal aspects of the comercialisation of the COTS: if the COTS has a *warranty*, the support (*support*) provided by the company, the intelectual (*ownership*) owner of the COTS, the degree of reliability (*reliability*) of the system, the confidentiality (*confidentiality*) and the cost (*payment*) of the COTS.

$$
\begin{array}{|l}
\hline
\textit{License} \\
warranty : WARRANTY \\
support : SUPPORT \\
ownership : ID\_OWNER \\
reliability : RELIABILITY \\
confidentiality : CONFIDENTIALITY \\
payment : PAYMENT \\
\hline
\end{array}
$$

All the COTS are stored in the *Cots_Library*, which is accessed during the selection process.

$$
\begin{array}{|l}
\hline
\textit{Cots\_Library} \\
cots : ID\_COTS \rightarrow\!\!\!\!\!\!+ \; Cots \\
\hline
\end{array}
$$

3.4 Selection Formalisation

The proposed approach aims to assist the assessment of COTS products considering their quality aspects and evaluating the components alternatives against the overall set of non-functional requirements specified by the users. The first step of the selection process attempts to identify and find all potential candidates. The search is driven by the evaluation criteria which initially takes as input essential and critical requirements. During this stage, it is reasonable to consider only functional requirements because they are easier to acquire and satisfactory to start the COTS search. The typical sources are Internet, magazines, conferences, vendors catalogues. Besides, there are some search mechanisms for identifying products available in the market such as Agora [18] and JCentral.

We consider the library of all possible COTS candidates already exists. This library may contains the product's name, version and a brief description of product's capabilities and limitations. Until this stage, non-functional requirements were not considered, excepting in cases which they are part of essential requirements set.

The COTS evaluation process is performed by examining the products specification with regards to the requirements they need to satisfy. This phase uses a more detailed evaluation criteria including non-technical aspects, other functional requirements and a formal description of non-functional requirements. During the components evaluation, non-functional requirements plays an important role to discrimination between products.

The selection process is performed as shown in Figure 3. Formal specification of the software architecture, COTS and their NFRs are used in the selection process. In order to carry out the selection, it is defined the Selection_Operation Z schema as follows:

$$Checking\_Result ::= ok \mid no\_ok$$

$$
\begin{array}{l}
\text{\_\_ } Selection\_Operation \text{ \_\_\_\_\_\_\_\_\_\_\_\_} \\
cots\_library? : Cots\_Library \\
software\_architecture? : Software\_Architecture \\
selected\_cots! : ID\_COTS \nrightarrow Cots \\
\hline
\forall comp : ID\_COMPONENT \mid comp \in \operatorname{dom} software\_architecture?.components \\
\bullet \exists c : ID\_COTS \mid c \in \operatorname{dom} cots\_library?.cots \\
\bullet (cots\_library?.cots(c)).interface = (software\_architecture?.components(comp)).interface \land \\
(cots\_library?.cots(c)).nfrs = (software\_architecture?.components(comp)).nfrs
\end{array}
$$

4 Example

The World-Wide Web (WWW) is a distributed hypermidia system organised as a loosely connected set of client and servers[2] that share a common set of communication and markup protocols. Servers make internet resources available to a community of clients that communicate using the HTTP (HyperText Transfer Protocol) [9].

Basic elements on the behalf of the server are a path resolver that allows to determine which file an HTTP message is requesting, a stream manager which manages the raw communication with the network and a HTTP server which speaks at least the HTTP protocol. On the behalf of the client, it must provide a user interface manager to display the results of the user requests, an access manager to formulate user requests, a protocol manager to map between the various protocols that WWW supports and the access manager. Additionally, it also must have a stream manager for handling network communications and the cache manager that keeps a local cache of retrieved information (see Figure 4).

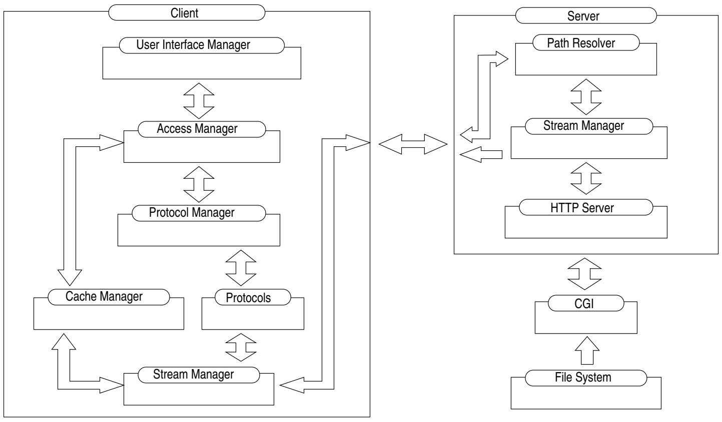

Figure 4: WWW Client-Server

In order to adaptate this example for our specific purposes, we add an additional characteristic of the server: the server must be secure. This kind of requirement, non-functional one, has been often demanded by commercial applications. In practial terms, the server must have to satisfy a certain security level when accessed by clients.

5 Conclusion

This paper has illustrated how NFRs can be formally specified and used to select COTS. From the Z specification of software architecture concepts, NFRs and COTS, it has been defined the formal selection operation which select COTS stored in a COTS library that best match the requirements defined in the software architecture. Unlikely the traditional approach in which only functional elements are used to select COTS, our approach also covers the non-functional characteristics of the software.

Fundamental for our approach is the formalisation of all elements present in the selection process. Through a widely used formal notation, proofs can be carried out using tools such as Z/EVES.

[2] it follows the Client-server architectural style [6].

In terms of future work, the COTS library shown in Figure 4 must be dinamically updated by a search engine. This engine must looking for components in the Internet, which may be made available by companies.

References

[1] G. D. Abowd, R. Allen, and D. Garlan. Formalizing style to understand descriptions of software architecture. *ACM Transactions on Software Engineering and Methodology*, 4(4):319–364, Oct. 1995.

[2] L. Chung. Representation and utilization of non-functional requirements for information system design. In *3rd International Conference on Advanced Information System Engineering - CAiSE'91*, Trondheim, Norway, May 1991.

[3] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-functional requirements in Software Engineering*. Kluwer Academic Publishers, 1999.

[4] L. Dukic. Non-functional requirements for COTS software components. In *Workshop on Ensuring Successful COTS Development*, Los Angeles, CA, USA, May 1999.

[5] J. Foreman. On the front lines of COTS - lessons learned, speculation for the future. Technical report, Software Engineering Institute, Mar. 1999.

[6] D. Garlan. What is style? In *Dagshtul Workshop on Software Architecture*, Feb. 1995.

[7] D. Garlan, R. Allen, and J. Ockerbloom. Architectural mismatch or why it's hard to build systems out of existing parts. In *International Conference on Software Engineering (ICSE'95)*, Seattle, USA, Apr. 1995.

[8] P. Inverardi and A. L. Wolf. Formal specification and analysis of software architectures using the Chemical Abstract Machine Model. *IEEE Transactions on Software Engineering*, 21(4):373–386, Apr. 1995.

[9] R. Kaznman, L. Bass, G. Abowd, and P. Clements. An architectural analysis case study: Internet information systems. In *First International Workshop on Architectures for Software Systems*, pages 148–165, Seattle, WA, USA, 1995.

[10] J. Kontio. A case study in applying a systematic method for COTS selection. In *18th International Conference on Software Engineering*, Berlin, 1996, Mar. 1996.

[11] G. Kotonya and I. Sommerville. *Requirements engineering: process and techniques*, chapter 8, pages 190–213. John Wiley & Sons, Inc, 1998.

[12] D. Kunda and L. Brooks. Applying social-technical approach for COTS selection. In *4th UKAIS Conference*, University of New York, Apr. 1999.

[13] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, Jan. 2000.

[14] C. Ncube. *A Requirment Engineering Method for COTS-Based Systems Development*. PhD thesis, City University, Dec. 1999.

[15] P. Oreizy, N. Medvidovic, R. N. Taylor, and D. S. Rosenblum. Software architecture and component technologies: bridging the gap. In *OMG-DARPA Workshop on Compositional Software Architectures*, Monterey, CA, USA, Jan. 1998.

[16] V. C. C. Paula, G. R. R. Justo, and P. R. F. Cunha. ZCL: a formal framework for specifying dynamic distributed software architectures. In *Ninth European Workshop on Dependable Computing (EWDC'9)*, Gdansk, Poland, May 1998.

[17] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *Software Engineering Notes*, 17(4):40–52, Oct. 1992.

[18] C. S. Robert, A. H. Scott, and C. W. Kurt. Agora: A search engine for software components. Technical report, Software Engineering Institute, Carnegie Mellon University, USA, 1998.

[19] N. S. Rosa, P. R. F. Cunha, and G. R. R. Justo. Uso de E-LOTOS na descrição de arquiteturas de software de sistemas distribuídos. In *Workshop em Sistemas Distribuídos (WoSiD'98)*, Curitiba, Paraná, June 1998.

[20] N. S. Rosa, G. R. R. Justo, and P. R. F. Cunha. A Framework for Building Non-functional Software architectures. In *16th ACM Symposium on Applied Computing*, Las Vegas, USA, 2001. (To Appear).

[21] M. Shaw and D. Garlan. *Software architecture : perspectives on an emerging discipline*. Prentice Hall, 1996.

[22] J. Spivey. *Understanding Z: a specification language and its formal semantics.* Cambridge University Press, 1988.

[23] C. Thompson. Workshop on Compositional Software Architectures. Workshop Report, Feb. 1998. Monterey, California.

[24] V. Tran and T. Liu. A procurement-centric model for engineering component-based software systemsosftw. In *Fifth International Symposium on Assessment of Software Tools*, June 1997.

[25] K. C. Wallnau, D. Carney, and B. Pollak. How COTS software affects the design of COTS-intensive systems. Technical report, Software Engineering Institute, Carnegie Mellon University, USA, June 1998.