A Non-Functional Approach for COTS Components Trading[*]

Luis Iribarne, Antonio Vallecillo, Carina Alves, Jaelson Castro
University of Almería, University of Málaga, Universidade Federal de Pernambuco.
liribarne@ual.es, av@lcc.uma.es, {cfa, jbc}@cin.ufpe.br

Abstract. In CBSD, the possible benefits of COTS software development, such as low cost, low risk, and high quality, cannot be satisfactorily achieved due to inadequate and/or incomplete proposals for component specification. In particular, non-functional aspects play an important role for components description and selection, but are usually not taken into account when documenting components and searching for them. In this paper, non-functional features of commercial components are analyzed, and a template for collecting this kind of information in XML documents is proposed in order to allow effective COTS components trading.

Keywords: COTS components, Non-functional requirements, Trading, Component-Based Software Development (CBSD).

1. Introduction

In COTS-based development process, effective evaluation and selection of COTS software products is one of the key aspects of the system development life cycle. Its success largely depends on the accurate understanding of the capabilities and limitations of the individual candidate products. COTS-based development involves simultaneous trade-offs among user requirements, system architecture and COTS products. The selection of suitable components is often a non-trivial task and all these aspects should be carefully considered [15]. As a result, the customers must accept the possibility that the resulting system might be a compromise among these concerns.

In general, a COTS selection process initially decomposes the requirements for the potential COTS candidates into a hierarchical set of criteria.

These criteria usually include components' functionality, non-functional requirements, architecture constraints, and non-technical factors such as vendor guarantees and legal issues. Then, during the selection activity, the properties of each COTS candidate are identified and assessed according to this set of evaluation criteria.

[*]This work has been supported by CYTED project "WEST: Web-oriented Software Technology" (CYTED subproject VII, code VII.18)

It is worth noting that non-functional requirements play an important role during the assessment of COTS components. The lack of a careful consideration of non-functional requirements increases the risks of COTS failure and the costs of the final system because these requirements often correspond to strategic or business objectives of the organization as a whole [9]. Since non-functional requirements have a global nature, i.e. the satisfaction of a particular requirement can affect several parts of the system, they might have higher priority if conflicting with some of the functional requirements. In addition, the careful analysis of quality attributes can improve the discrimination process between competing COTS products that already meet the core functional requirements. For instance, if two components implement the same task (i.e. they have similar functionality), non-functional attributes may be used in the selection process as further and decisive criteria.

Despite the widely recognised importance of considering non-functional requirements in CBSD, most proposals for documenting COTS components fail to take them into account. Or even if they do, either their treatment is too simplistic (as in the case of the approaches that follow the ODP "property"-based model [14]), or they are not fully integrated within a RE framework that covers the initial phases of the software development process, and that allows to transform the user requirements into the appropriate architectural and component (functional and non-functional) requirements.

On the other hand, trading is the natural mechanism defined in COTS-based development for searching and locating components. A client role that requires a particular component service can query a matchmaking agent (the *trader*) for references to available components that provide the kind of service required. Service advertisements are usually called "exports", while queries are called "imports". The trader provides just the references to possible service providers, but without intervening in the service provision itself, which is a private matter between the client and the server (the one selected by the client from the list of candidates returned by the trader). An example of a COTS components trader for open systems can be found in [13]. However, what we found in most current traders is that they fail (again) to deal with non-functional requirements.

This paper presents a proposal for documenting COTS components in such a way that non-functional requirements can be described, and that traders can use for effectively locating and selecting the appropriate components. One of the major advantages of our proposal is that has been designed within a RE framework, namely NFR [5]. This allows a natural integration with it, as well as other interesting benefits, such as requirements decomposition and traceability.

This paper is organized as follows. In Section 2 we explain a proposal for documenting COTS components using XML templates, that can be used for services trading in commercial environments [13]. After that, in Section 3 we focus on non-functional requirements, identifying and analyzing the importance of these requirements for effective "off-the-shelf" components evaluation and selection processes. Then, in Section 4 we propose a way to describe non-functional requirements in COTS documents, extending the XML templates introduced in Section 2. Finally, we discuss some related work and draw some conclusions in Sections 5 and 6.

2. Documenting commercial components

In CBSD, the process of building COTS-based systems includes some tasks, such as
(a) searching for components that satisfy the requirements of the system architecture; (b) evaluating these components; (c) adapting or extending the selected components to fit into the application architecture; and (d) gluing or integrating these components together [6]. It is very important for these processes to use complete, concise and unambiguous specifications of components in order to guarantee a successful COTS software development. In addition, these specifications could be later registered into well-know repositories by developers or third parties, facilitating the COTS development process.

Most of the existing proposals for documenting components are based on the notion of component interface, which provides a form to control the dependencies that arise between components in a program or system [1]. In that way, most of the programming languages (e.g., Java, C#, Smalltalk, and so on) support some mechanisms to define interfaces by means of Interface Definition Languages (IDLs). However, most IDL proposals are restricted to express component syntactical features, ignoring other relevant aspects, such as protocols, behavioral, or semantic information [19], or non-functional features [5][18].

On the other hand, the proper searching and selection processes of COTS components have become the cornerstone of effective COTS development. However, these processes currently face serious limitations, mainly because the information available about the components is not expressive enough for their effective selection, and also because the search and evaluation criteria are usually too simplistic to provide practical utility.

In [13] we introduced a proposal for documenting and searching COTS components, based on two XML templates, the first one (called *COTScomponent*) for documenting components, and the second one (called *COTSquery*) for querying traders. These templates can be used by several kinds of users (i.e., system architects, designers, developers, and vendors) to export and import components to/from software repositories. These templates are available at http://www.cotstrader.com.

Following our proposal, a component can be defined inside a *COTScomponent* template, which consists of four main parts: *functional*, *properties*, *packaging*, and *marketing* part. Figure 1 shows a COTS document example in XML format, using the COTScomponent template. A detailed description of the example document is beyond the scope of this paper, although a complete description is available in our COTStrader web site. At the beginning of this template we can see the name of the component (OnePlaceBuffer) and two name spaces, first one pointing to the *COTS-XMLSchema* schema (living at the COTStrader site) and the second one pointing to the W3C's XMLShema types (e.g., for parsing processes).

Next, we can see the four parts mentioned previously. The *functional* part describes the computational aspects of the service, including both syntactic (i.e., signature) and semantic information. Our functional definition of a service will host the set of interfaces offered by the service (providedInterfaces), and the set of required interfaces that any instance of the service may require from other components when implementing its supported interfaces (requiredInterfaces). In this example, the component offers two interfaces, one with the basic functionality, and other for "logging-in" into the service. Semantic information can be described with pre/post conditions (inside behavior tag), as well as by means of protocols (serviceAccessProtocol), which specify the relative order in which a component expects their methods to be called, and the way it calls other components' methods. The *properties* part describes the non-functional aspects of the service (e.g., QoS, NFRs, etc.), which are based on "properties", i.e. pairs *(name, value)* following the RM-ODP standard. The *packaging* part contains the packaging information about how to download, deploy and install the COTS component that provides the required service, including implementation details, context and architectural constraints, etc. Finally, the *marketing* information deals with the rest of the non-technical issues of the service, such as licensing and pricing information, vendor details, special offers, etc.

In this paper we will show how this proposal can be enhanced to deal with non-functional requirements in a more effective way, and within a RE approach, namely NFR.

3. Non-Functional description

Non-functional requirements address important issues of quality and restrictions for software systems, although some of their particular characteristics make their specification and analysis difficult [5]:

- Non-functional requirements can be subjective, since they can be interpreted and evaluated differently by different people;
- Non-functional requirements can be relative, since their importance and description may vary depending on the particular domain being considered;
- Non-functional requirements can be interacting, since the satisfaction of a particular non-functional requirement can hurt or help the achievement of other non-functional requirement.

Despite this problematic nature of non-functional requirements, we do insist that they need to be explicitly treated during the evaluation of COTS components. Unlike the conventional development, in COTS-based systems customers do not have control over components capabilities nor access to their internal behavior. Therefore, the careful analysis of non-functional requirements such as interoperability, adaptability, and reliability can improve the evaluation of COTS components and guarantee that components will be properly integrated into the specified software architecture.

In order to effectively deal with non-functional requirements, we use some principles from a qualitative approach called NFR Framework [5]. This approach is based on the explicit representation and analysis of non-functional requirements. Considering the complex nature of non-functional requirements, we cannot always say that non-functional requirements are entirely accomplished or satisfied. Rather, the NFR Framework represents non-functional requirements as *softgoals*, which does not necessarily have a priori, clear-cut criteria of satisfaction.

```
<?xml version="1.0"?>
 <COTScomponent name="OnePlaceBuffer"
                xmlns="http://www.cotstrader.com/COTS-XMLSchema.xsd"
                xmlns:types="http://www.w3.org/2001/XMLSchema">

  <!-- 1: Functional information -->
  <functional>
   <providedInterfaces>
    <interface name="OnePlaceBuffer">
     <description notation="CORBA-IDL">
      interface OnePlaceBuffer {void write(in long x); long
        read();};
     </description>
     <behavior notation="Larch"> ... </behavior>
    </interface>
    <interface name="LoginInterface"> ... </interface>
   </providedInterfaces>
   <requiredInterfaces> ... </requiredInterfaces>
   <serviceAccessProtocol> ... </serviceAccessProtocol>
  </functional>

  <!-- 2: Non functional information -->
  <properties notation="W3C">
    <property name="confidentiality">
      <type>xsd:string</type> <value>CRYPTOGRAPHY[PublicKey]</value>
    </property>
    ...
  </properties>

  <!-- 3: Packaging information -->
  <packaging>
    <description notation="CCM-softpkg"
     href=".../OnePlaceBuffer_Impl.csd"/>
  </packaging>

  <!-- 4: Marketing information -->
  <marketing>
    <license
     href="http://www.cotstrader.com/examples/OPB/license.html"/>
    <expirydate> 05-10-2001 </expirydate>
    <certificate
     href="http://www.cotstrader.com/examples/OPB/lcard.png"/>
    ...
  </marketing>

 </COTScomponent>
```

Figure 1. A COTS document using the COTS-XML Schema template

In addition, non-functional requirements can contribute positively or negatively, and fully or partially, towards achieving other non-functional requirements. Firstly, they are decomposed into more specific non-functional requirements. For instance, the security requirement can be considered quite broad and abstract. To explicitly deal with such a broad requirement, we may need to break it down into smaller parts, so that unambiguous solutions can be found. By treating this high-level requirement as a *softgoal* to be achieved, we can decompose it into more specific subgoals which together satisfy the higher-level softgoal (this is an AND type of contribution). Thus the security softgoal can be decomposed into sub-softgoals: integrity, confidentiality and availability. Another kind of contribution is the OR type, with this relationship the softgoal is satisfied if any of its sub-goals is.

At some point, when the refinement process carried out so far provides more specific descriptions of the stated non-functional requirements, developers may consider that these requirements have been sufficiently refined and possible operationalizations can be found. It is worth noting that operationalizations are related with functionalities that implement initial non-functional requirements. A very important aspect of non-functional requirements decomposition using the NFR Framework is that, as far as NFR softgoals are refined into more detailed ones, it is possible to identify interactions between non-functional requirements. These interactions include positive and negative contributions and have a critical impact on the decision process for achieving other non-functional requirements. A suitable way to deal with such complex interdependencies is to assign priorities to non-functional requirements in order to make appropriate tradeoffs among NFRs. In addition, all design decisions should be supported by well-justified rationales.

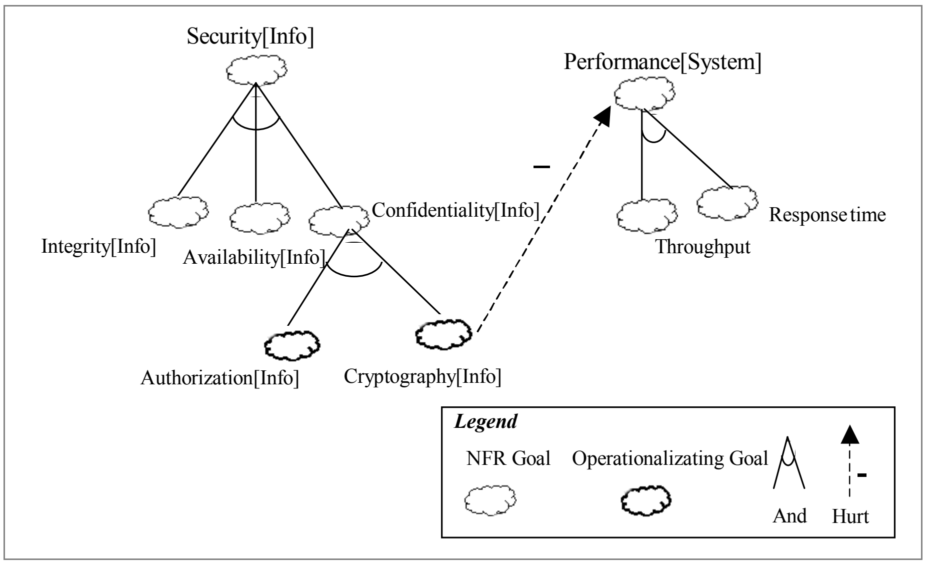

Figure 2. Decomposition of non-functional requirements using the NFR Framework

For instance, figure 2 shows a decomposition of non-functional requirements using the NFR Framework. The goal *security of information* is decomposed into the subgoals *integrity, availability, confidentiality* through an AND type of contribution (i.e. only if all subgoals are met the overall goal is achieved). While the goal *system performance* is decomposed into *throughput* and *response time*. Interestingly, it is necessary to address interactions between different kinds of non-functional requirements even though the non-functional requirements were initially stated as separate requirements. Note that *cryptography* contributes negatively (show as "-") for *system performance*. Since the NFR Framework facilitates the understanding of what a particular non-functional requirement means, this approach can effectively help the description of NF information for COTS documentation.

4. Including Non-Functional information into COTS documents

Once we have identified and analyzed the importance of the non-functional requirements for effective COTS component evaluation process, in this section we propose a mechanism for collecting non-functional information into COTS documents using the COTS-XMLSchema template [13]. For our purposes, we only focus on the *properties* part. At the end of this paper we include two appendices which show the non-functional schemas (i.e., grammar for documenting NF information) developed and used in this paper. In order to explain this section, we will use an instantiation of these schemas in the following examples.
Our main goal is to extend this information in order to enhance the COTS components documentation regarding the non-functional requirements so COTS components can be naturally integrated into the NFR framework. In our XML approach we adopted the ODP style to describe non-functional properties, which are the usual way in which the non-functional aspects of objects, services and components are expressed in the literature. We suggest the use of W3C types for describing properties, although any notation is valid for describing them (e.g. the OMG's CCM style [17] which also uses an XML vocabulary). In figure 3 we show a simple example of use case for documenting non-functional information.

```
<properties notation="W3C">
  <property name="confidentiality">
    <type>xsd:string</type> <value>CRYPTOGRAPHY[Public Key]</value>
  </property>
  <property name="capacity">
    <type>xsd:int</type> <value>1</value>
  </property>
  <property name="isRunningNow">  <!-- dynamic property -->
    <type>xsd:bool</type> <value
     href="http://...:8080/servlet/OPB.running"/>
  </property>
  <property name="keywords">
    <type>xsd:string</type> <value>storage,bounded</value>
  </property>
</properties>
```

Figure 3. An example of some NF information in a COTS description template

Inside the `<properties>` tag there is a collection of properties, each one indicated by a `<property>` tag, and with associated type and value. Keywordbased searches are also allowed, including the special property "keywords". Dynamic properties can also be implemented by declaring the reference to the external program that evaluates their current value. Properties can also be described in a separate file, which can be pointed from the `<properties>` tag as usual:

```
<propertiesnotation="CCM-properties"href=".../OnePlaceBuffer.cpf"/>
```

On the other hand, at the COTS component description level we can enhance properties, which may be either single properties, or composition of properties. Composition can be either *AND-composition* or *OR-composition*. Figure 4 shows an "OR" example.

```
<properties notation="W3C">
  <property name="security" composition="OR">
   <property name="user-authorization">
     <type>xsd:string</type> <value>LOGIN</value>
   </property>
   <property name="manager-authorization">
     <type>xsd:string</type> <value>ADMIN</value>
   </property>
  </property>
 </properties>
```

Figure 4. Including OR composition in a property tag

In addition, we need to express that a given property (no matter whether it is simple or composed) is "*implemented by*" a given functional element (e.g., an interface), or that a given property is "*present*" in a given functional element. The first issue permits NFRs traceability (which functional element provides a NFR), while the second one defines the functional elements that exhibit a certain NFR (by default, the whole component). Thus, as we can see in Figure 5, we propose adding two optional child XML elements to a property: `<implementedBy>` and `<implementedIn>` tags.

```
<properties notation="W3C">
 <property name="user-authorization">
   <type>xsd:string</type><value>LOGIN</value>
   <implementedBy>LoginInterface</implementedBy>
 </property>
 <property name="replication">
   <type>xsd:string</type><value>CONSENSUS</value>
   <implementedIn>OnePlaceBuffer</implementedIn>
   <implementedIn>LoginInterface</implementedIn>
 </property>
</properties>
```

Figure 5. Functional element/s presented/provided by/from a NFR

In these descriptions, the information within the `<implementedBy>` and `<implementedIn>` tags refers to "functional" elements described in any of the `<functional>` tags of the COTS component, that is, inside a *COTScomponent* document (see Section 2), such as in a `<providedInterface>`, a `<behavior>`, or a <serviceAccessProtocol>. A valid XML pointer to a local XML tag refers to the functional element. It is important to note that the reference to a functional element should always be a local reference, i.e., it should point to an element present in the same template, since we are trying to express *how* a NFR is implemented by *that* particular component, or which particular interface provides a given property.

Once we have shown how to describe NFRs in terms of properties, there is only one thing left: how to have them into account when searching for components. In [13], a client will need to provide two XML documents in order to look for components satisfying his requirements. The first one containing the selection criteria to be used by the trader to look for the service (WHAT), and it will point to the second document, which describes the main features of the required service (HOW). That is, queries look for COTS candidates containing a <COTScomponent> XML document with the component features that we wanted in the target component, together with a <COTSquery> document that determines the selection criteria. In the particular case of NFRs, the <COTScomponent> document establishes the precise properties we are looking for (see figure 6) and then in the <COTSquery> the selection criteria are determined (see figure 7).

```
<?xml version="1.0"?>
<COTScomponent name="authorizationStyle"
 xmlns="http://www.cotstrader.com/COTS-XMLSchema.xsd">
 <properties notation=”W3C”>
   <property name=”security” composition=”OR”>
     <property name=”user authorize”><type>xsd:string</type></property>
     <property name=”manager-
       authorize”><type>xsd:string</type></property>
   </property>
   <property name=”isRunningNow”><type>xsd:boolean</type></property>
 </properties>
</COTScomponent>
```

Figure 6. Document A of a query (WHAT). Selection criteria.

```
<?xml version="1.0"?>
<COTSquery name="authorizationStyle"
     xmlns="http://www.cotstrader.com/COTS-XMLSchema.xsd">
  <COTSdescription href="http://.../authorizationStyle.xml"/>
  <propertyMatching>
    <constraints notation="Xquery">
     (//property[name="security"]/property[name="user-
     authorize"]/value="SAFE") and
     (//property[name="isRunningNow"]/value="TRUE")
    </constraints>
    <preferences notation="ODP">first</preferences>
  </propertyMatching>
 </COTSquery>
```

Figure 7. Document B of a query (HOW). Constrains and preferences.

Usually, the property matching is accomplished by ODP traders [14], using constraints and preferences. Constraints are boolean expressions consisting of values, constants, relational operators (`<`, `>=`, `=`, `!=`), logical operators (`not`, `and`, `or`) and parenthesis, that specify the matching criteria for including a component in the trader's list of candidates for the current searching. Constraints are evaluated by the trader by substituting the property names with their actual values, and then evaluating the logical expression. Components whose constraint evaluates to false are discarded. In the example, for writing the expression, we have used the W3C's XML QueryAlgebra notation [21]. Preferences allow sorting the list of candidates according to a given criteria, which is expressed using the terms `first`, `random`, `min(expr)` and `max(expr)`, where `expr` is a mathematical expression involving property names [14].

On the other hand, quality aspects are hardly specified in COTS descriptions, yet they are critical during the selection/matching process. In that sense, it is necessary to describe refinements, identifying interdependencies among different non-functional requirements and assigning priorities that provide rationales useful during the decision-making process. Therefore, we propose to enrich the above XML elements, including priorities when the *target* COTS component is described (see figure 8). Priorities can be assigned to each first-level property in the `<properties>` tag. Those priorities levels may be assigned using the scale 0 (very low) to 9 (very high), which is the scale commonly used in most decision- making processes nowadays.

```
<properties notation="W3C">
 <property name="security" composition="OR" priority=7>
    <property name="user-authorize"><type>xsd:string</type></property>
    <property name="manager-
      authorize"><type>xsd:string</type></property>
 </property>
 <property name="isRunningNow"
  priority=4><type>xsd:boolean</type></property>
</properties>
```

Figure 8. Priority of the property elements

5. Related work

In this section we discuss some of the works proposed in the literature which can be related to our proposal in two main areas, namely: (a) documentation of components and (b) non-functional requirements (both areas focused on COTS components trading).

In the first place, there are a number of proposals related to enhancing the documentation of components. Dong et al. [8] show a proposal on component specification template, which includes functional information (i.e., structural and behavioural aspects of the services), non-functional properties, and some extra information (i.e., applicability, standards, related components, and sample uses). IBM[1] is working on a proposal for documenting their large grained components, and there are several interesting proposals from SEI[2] claiming better component documentation. On the other hand, Jun Han [12] has also defined some component specification templates in a joint project with Fujitsu Australia, which provide semantic information for proper usage and selection of components on top of their standard signature description. Finally, Cho [4], Bastide et al. [1], and Canal et al. [3] propose IDL extensions for dealing with protocol information, using state transitions, Petri nets, and pi-calculus, respectively. Our previous work [13] even shows how most of these proposals can also be smoothly integrated into XML templates that could be used for effective trading of COTS components. However, most of these approaches deal with the components' non-functional requirements in a very simplistic way. What we have showed here is that these functional descriptions can be easily integrated with a consolidated proposal for dealing with non-functional requirements, such as NFR, hence allowing to connect both worlds.

In the second place, there are the works proposed in the literature for dealing with non-functional requirements. Franch uses a notation, called NoFun, for describing non-functional requirements [10]. This approach is product-oriented, i.e., NoFun is a notation for describing non-functional information of software systems at the product level. This notation was defined in a component-programming framework. In a previous work [7], we proposed an approach, named the CRE (COTS-based Requirements Engineering) Method, which was developed to facilitate a systematic, repeatable and requirements-driven COTS software selection process. A key issue supported by this method is the definition and analysis of non-functional requirements during the phases of COTS evaluation and selection. One of the advantages of our proposal with those is the integration with a well-known RE framework, which allows to incorporate COTS components into the traditional RE methods, enabling requirements decomposition and traceability, as discussed above.

6. Conclusions

Component-Based Software Development (CBSD) aims at building software systems by searching, selecting and integrating COTS components. In a previous work [13], we analyzed the required features that COTS components traders should have, and presented COTStrader, an Internet-based trader for COTS components that handled the heterogeneity, scalability and evolution of COTS markets.

However, the treatment of non-functional requirements was too simplistic, as in most current proposals. Non-functional requirements cannot be ignored during the assessment of commercial components in order to obtain an effective and realistic trading.

[1] http://www.ibm.com/software/components

[2] http://www.sei.cmu.edu

In this context, our proposal introduces a way to describing COTS components' non-functional information, integrating it with functional information using COTS-XMLSchema templates [13]. Non-functional requirements are described using the NFR approach, which has been successfully used in many situations. The XML templates introduced here have been primarily defined for documenting and looking for COTS components, hence allowing more effective trading processes in open systems, particularly for the Internet. This work have combined the functional and non-functional aspects of COTS components into a single description, and integrated into a trading process. As future work, we plan to concentrate on the system's non-functional requirements, and how they can be mapped into the system's software architecture requirements, and then into the individual components' requirements. Basically, we are concerned with the traceability of such requirements along the software life-cycle, which is a critical issue in CSBD.

References

[1] Bachman, F., Bass, L., Buhman, C., Comella-Dorda, S., Long, F., Robert, J., Seacord, R., and Wallnau, K, "Technical Concepts of Component-Based Software Engineering" (Vol 2). Technical Report CMU/SEI-2000-TR-008. SEI, 2000.

[2] Bastide, R., Sy, O., and Palanque, P., "Formal Specification and Prototyping of CORBA Systems." *In Proceedings of ECOOP'99*, number 1628 in LNCS, Springer-Verlag, 1999, pp. 474-494.

[3] Canal, C., Fuentes, L., Troya, J. M., and Vallecillo, A., "Extending CORBA Interfaces with π-calculus for Protocol Compatibility." *In Proceedings of TOOLS Europe 2000*, France, June 2000. IEEE Press, pp. 208-225.

[4] Cho, I., McGregor, J., and Krause, L., "A Protocol-Based Approach to Specifying Interoperability Between Objects." *In Proceedings of TOOLS'26*, IEEE Press, 1998, pp. 84-96.

[5] Chung, L., Nixon, B., Yu, E., and Mylopoulos, J., *Non-Functional Requirements in Software Engineering.* Kluwer Academic Publisher, 2000.

[6] Dean, J., and Vigser, M. R., "System Implementation Using Off-the-shelf Software." *In Proceedings of the 9th Annual Software Technology Conference*, April 1997.

[7] Alves, C., Castro, J., and Alencar, F., "Requirements Engineering for COTS Selection." *In Third Workshop on Requirements Engineering,* Rio de Janeiro, Brazil, 2000.

[8] Dong, J., Alencar, P. S. C., and Cowan, D. D., "A Component Specification Template for COTS-based Software Development." *First Workshop on Ensuring Successful COTS Development.* May 1999.

[9] Dukic, L., "Non-Functional Requirements for COTS Software Components." *Workshop Ensuring Successful COTS Development.* May 2000.

[10] Franch, X., and Burgues, X. A., "Language for Stating Component Quality." *XIV Simpósio Brasileiro de Engenharia de Software.* October 2001.

[11] Garlan, D., Allen, R., and Ockerbloom, J., "Architectural Mismatch (Why it is hard to build systems out of existing parts)." *Proceedings of the 17th International Conference on Software Engineering,* April 1995.

[12] Han, J., "Temporal Logic Based Specifications of Component Interaction Protocols." In J. Hernández, A. Vallecillo, and J. M. Troya (eds.). *Proceedings of the ECOOP'2000 Workshop on Object Interoperability (WOI'00),* June 2000, pp. 43-52.

[13] Iribarne, L., Troya, J. M., and Vallecillo, A., "Trading for COTS Components in Open Environments." *To appear in Proceedings of 27th Euromicro. Workshop on Component Based Software Engineering*, Warsaw, Poland. September 2001. IEEE Software.

[14] ISO/ITU-T. Information Technology - Open Distributed Processing - ODP trading function. Rec. ISO/IEC DIS 13235, ITU-T X.9tr. Feb, 1996.

[15] Kotonya, G., and Sommerville, I., "Requirements Engineering with Viewpoints." *Software Engineering,* 1(11) 1996, pp. 5-18.

[16] Ncube, C., and Maiden, N., "PORE: Procurement-Oriented Requirements Engineering Method for the Component-Based Systems Engineering Development Paradigm." *International Workshop on Component-Based Software Engineering,* May 1999.

[17] OMG. "The CORBA Component Model (CCM)." *Object Management Group*. June, 1999. http://www.omg.org.

[18] Rosa, N., Alves, C., Cunha, P., Castro, J., and Justo, G., "Using Non-Functional Requirements to Select Components: A Formal Approach." *In Proceedings of IDEAS'01,* San José, Costa Rica. April 2001.

[19] Vallecillo, A., Hernández, J., and Troya, J. M., "Object Interoperability." *In ECOOP'99 Workshop Reader,* number 1743 in LNCS, pages 1-21. Springer-Verlag, 1999.

[20] W3C. XML Schema Language. XML Schema subgroup. World Wide Web Consortium.Technical Report, June 2001. http://www.w3.org/XML/Schema.

[21] W3C. XQuery 1.0: A Query Language for XML. XML Query subgroup. World Wide Web Consortium. Technical Report, June 2001. http://www.w3.org/TR/query-algebra/

Appendix A. The NF-Schema in COTS-XMLSchema

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!-- This is a part of COTS-XMLSchema -->
<!-- Non Functional description -->
 <xsd:element name="properties">
  <!-- Properties tag has a notation attribute, -->
  <xsd:complexType>
   <xsd:attribute name="notation" type="xsd:string"/>
   <!-- then properties may choice between a href attribute or
    one or more property tags -->
   <xsd:choice>
    <xsd:attribute name="href" type="xsd:uriReference"/>
    <xsd:element name="property" maxOccurs="unbounded">
     <!-- Property has 3 attributes: name, composition and priority
      -->
     <xsd:complexType>
      <xsd:attribute name="name" type="xsd:string"/>
      <xsd:attribute name="composition" minOccurs="0">
       <xsd:simpleType>
        <xsd:restriction base="xsd:string">
         <xsd:enumeration value="OR"/>
         <xsd:enumeration value="AND"/>
        </xsd:restriction>
       </xsd:simpleType>
      </xsd:attribute> <!-- end composition attribute -->
      <xsd:attribute name="priority" minOccurs="0">
       <xsd:simpleType>
        <xsd:restriction base="xsd:positiveInterger">
         <xsd:minInclusive value="0"/>
         <xsd:maxInclusive value="9"/>
        </xsd:restriction>
       </xsd:simpleType>
      </xsd:attribute> <!-- end priority attribute -->
      <!-- Elements of a property: type, value, implementedBy,
           implementedIn, and one or more property tags (appointing
to property element just defined above) -->
      <xsd:sequence>
       <xsd:element name="type" type="xsd:datatype"/>
       <xsd:element name="value" minOccurs="0"/>
        <xsd:complexType>
         <xsd:choice>
          <xsd:attribute name="href" type="xsd:uriReference"/>
          <xsd:simpleContent>
           <xsd:extension base="xsd:string"/>
          </xsd:simpleContent>
         </xsd:choice>
        </xsd:complexType> <!-- end value element -->
       <xsd:element name="implementedBy" type="xsd:string"
                    minOccurs="0" maxOccurs="unbounded"/>
       <xsd:element name="implementedIn" type="xsd:string"
                    minOccurs="0" maxOccurs="unbounded"/>
       <xsd:element href="property"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element> <!-- end property element -->
   </xsd:choice>
  </xsd:complexType>
 </xsd:element> <!-- end properties element -->
</xsd:schema>
```

Appendix B. The query NF-Shema in COTS-XMLSchema

```
<xsd:schema xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>

<!-- Queries looking for NF -->

 <xsd:element name="propertyMatching">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="constraints" type="locationType"/>
    <xsd:element name="preferences" type="locationType"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>

<!-- A complex type is declared both in constraints and preferences -->

 <xsd:complexType name="locationType">
  <xsd:attribute name="notation" type="xsd:string" minOccurs="0"/>
  <xsd:choice>
   <xsd:attribute name="href" type="xsd:uriReference"/>
   <xsd:simpleContent>
     <xsd:extension base="xsd:string"/>
   </xsd:simpleContent>
  </xsd:choice>
 </xsd:complexType>

</xsd:schema>
```