

Requirements Engineering for COTS-based Software Systems

Juan P. Carvallo

ETAPATELECOM

Calle Larga y Ave. Huayna Capac,
edif. Banco Central Ecuador, Cuenca
+593 7282 8228

jpcarvallo@etapatelecom.net

Xavier Franch

Universitat Politècnica de Catalunya

UPC – Campus Nord, Omega 122
Barcelona, Spain
+34 93 41 37881

franch@lsi.upc.edu

Carme Quer

Universitat Politècnica de Catalunya

UPC – Campus Nord, Omega 119
Barcelona, Spain
+34 93 41 37888

cquer@lsi.upc.edu

ABSTRACT

Large software systems are often deployed putting together many Commercial-Off-The-Shelf software components (COTS). The selection of the COTS to be integrated is driven by the software system requirements. In this paper, we propose the RECSS method aimed at supporting requirements elicitation and analysis in the context of COTS-based software systems. RECSS builds a goal model of the system environment which identifies the external elements that interact with it. Next, the system is decomposed into actors for which ISO/IEC-based quality models are built. As part of the process, environmental and platform characteristics that influence the behaviour of the system are identified. Last, the resulting artefacts are used to analyse and refine the system requirements.

Categories and Subject Descriptors

D.2.1 [Requirements Engineering]: Methodologies

General Terms

Measurement, Documentation, Design.

Keywords

COTS, Requirements Engineering, Quality Models.

1. INTRODUCTION

The role that software requirements play during the selection of *Commercial-Off-The-Shelf software components* (COTS for short) [10, 16] has been explored by the methods and processes proposed so far for driving this activity [17]. These methods and processes show the need for obtaining, in an efficient way, reliable and comprehensive descriptions of COTS quality supporting the formulation and analysis of system requirements.

Quality models [12] are an especially appealing way of structuring the information about software characteristics and

quality. They provide a hierarchy of software *quality features* (e.g., efficiency, availability) and *metrics* for computing their value. In the COTS context, a quality model is bound to a *COTS domain*, defined as a software domain for which COTS may be bought, downloaded or obtained by whatever means, such that they are atomic, i.e., there are not COTS in the market covering just part of the functionality considered for a software domain.

In the requirements engineering discipline, quality models have been proposed for a long time as a vehicle to formulate and analyze system requirements [7, 15]: once a quality model is available, system requirements may be expressed using the quality concepts defined therein. In the COTS context, some recent work proposes to describe the quality of the COTS available in the market in terms of these features [1, 4, 18]. Then, COTS selection can be conceptually seen as a matching problem in which the COTS and the requirements are expressed using the same ontology, as proposed in the seminal paper [10].

In the last years, we have carried out several industrial experiences applying such methods (see [4] for a summary). We experienced the benefits of working with quality models but also discovered some limitations in the current processes. These limitations stemmed from different sources:

- The gap among the target organization business processes and the quality model technical features was too big. The rationale behind the identification of features was not clear.
- The methods treated COTS components as isolated. However, they need to communicate with others, either to enable their installation or to extend or enhance their functionality. We dealt with these extensions directly in the quality model itself, using features such as *Suitability* and *Interoperability*, increasing the size of the model and damaging its understandability, evolution and reuse.
- The definition of some metrics depended on some aspects both from the target organization and the underlying platform, such as the number of registered users and the number of servers in a cluster. These features remained hidden.
- The methods focused on quality model construction only. The impact of quality models in requirements engineering was not studied in a systematic way.

As a result, we propose a method named RECSS (Requirements Engineering for COTS-based Software Systems). The method is outlined in section 2 and then studied in sections 3 to 6. Sections 7 and 8 provide some related work and the conclusions.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC '08, March 16-20, 2008, Fortaleza, Ceará, Brazil.

Copyright 2008 ACM 978-1-59593-753-7/08/0003...\$5.00.

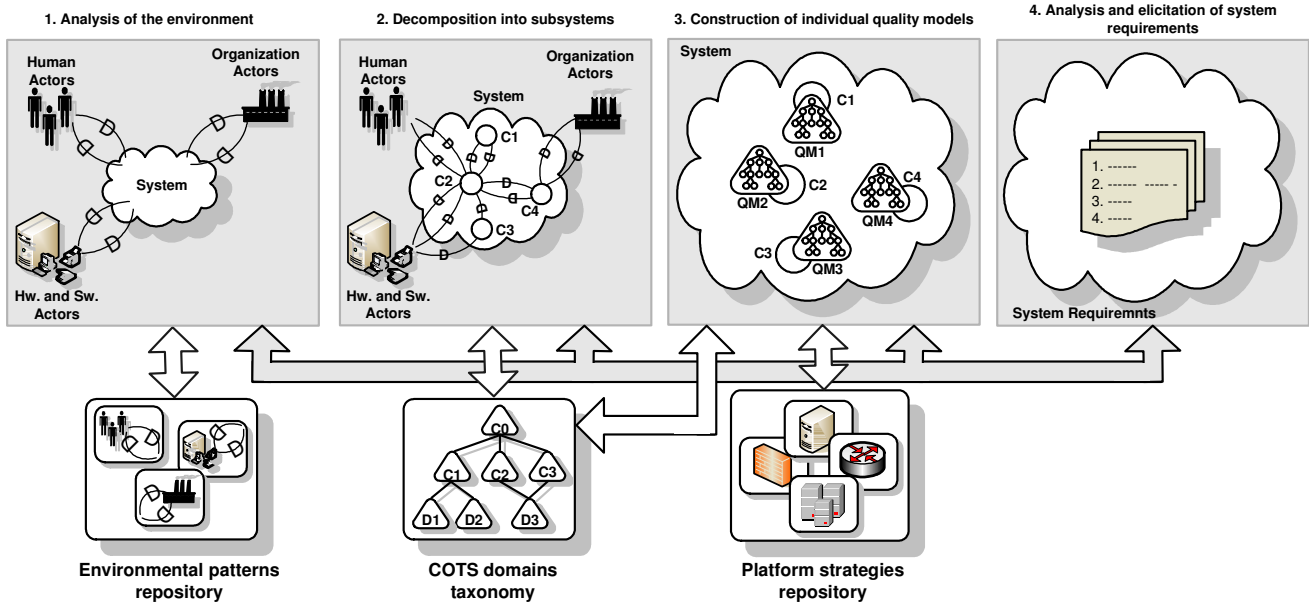


Figure 1. Overview of RECSS.

2. GENERAL OVERVIEW OF RECSS

RECSS consists of 4 activities that can be iterated and intertwined at any required extent (see Fig. 1):

- *Modelling the environment of the system.* Starting from the initial, normally incomplete, vague and even contradictory set of requirements, the organizational elements that surround the system are identified, as well as other external software systems which it interacts with. Dependencies among the system and the environment are established.
- *Decomposing the system into actors.* The system is decomposed into individual actors, each offering well-defined services. Services are identified starting from the model obtained in the previous activity. Dependencies among the new actors are established, and those identified in the previous activity are assigned also to these actors.
- *Building individual quality models for the actors.* ISO/IEC 9126-1-based quality models [13] are developed for each actor identified in the system. Environmental and platform features that influence on the metrics are identified, collected and presented as *system quality parameters*, because their variation could eventually require changes in the architecture of the system.
- *Analysing and eliciting system requirements.* The obtained models may be used to analyse the initial requirements, making them operational and detecting possible conflicts. Also, new requirements can be proposed from the information managed in RECSS.

The result of RECSS is twofold. On the one hand, the artefacts created, which can be used both for describing the software components candidate to be incorporated in the system and for supporting system maintenance (e.g., evolution of the organization or existence of new kinds of products offering new functionalities). On the other hand, the system requirements themselves, that can be used during procurement. The artifacts

in the two first activities are mainly represented by means of goal models, more specifically Yu's *i** Strategic Dependency (SD) models [20]. Also Strategic Rational (SR) models are used for decomposing the system actor. Knowledge of *i** is required to fully understand the first two activities of RECSS.

As shown in Fig.1, the RECSS method promotes return on investment by organizing the knowledge available from previous experiences. We use three kinds of repositories:

- *Environmental patterns repository.* Those situations that appear over and over in different system environments are identified, refactored and defined as patterns. For each pattern, we state which requirements lead to it and which ISO/IEC 9126-1 quality factors are addressed. Environmental patterns are introduced in section 3.
- *COTS domain taxonomy.* We organize the domains of the components available in the market as a taxonomy with the aim of facilitating the identification of the kind of components required in the system. The taxonomy includes quality models bound to its nodes' domains as well as relationships among them. The quality models include not only the usual quality features and their metrics, but also parameterized requirements that can be used during requirements analysis and elicitation. We do not introduce taxonomies in the paper, we refer to [5] for more details.
- *Platform strategies repository.* We maintain a catalogue of the most usual platform strategies used when deploying a system (such as clustering, load balancing, and so on). For each of them, the ISO/IEC subcharacteristics affected by their use are identified, as well as the platform parameters likely to appear. Platform strategies are presented in section 5.2.

3. SYSTEM ENVIRONMENT MODELING

The first activity of RECSS builds the *environmental model* that describes the *actors* in the environment of the system and the most important *dependencies* among them and the system.

The identification of the environmental actors and their dependencies is mainly based in the use of an environmental patterns repository. Patterns are described using the style of [11]:

- The *problem* solved by a pattern is expressed as a set of high-level requirements, which will be goals for the system if the pattern is considered.
- The *context* is the same for every pattern in the catalogue, since all of them are patterns to apply in the analysis of the environment of a system.
- The *solution* will be described as an *i** SD model. The pattern provides a scheme of a general solution that must be specialized for a concrete application of the pattern in a system.
- *Consequences* are non-functional quality entities that the use of the pattern affects in a positive or negative way. We use ISO/IEC 9126-1 subcharacteristics as a way to state these consequences.

Table 1 shows some examples. It does not include all the information for the sake of brevity, e.g. the pattern solutions include the possible types of every environmental actor (human, software, etc.) and a description of their actors' role. Let's consider the *Full Availability* pattern. We would select this pattern from the repository in case that the requirement "The system must offer full availability" applies to the system-to-be. The pattern identifies two environmental actors needed in this case, namely a system administrator and a system user, and the dependencies among these actors and the system. On the one hand, the *System User* depends on the *System* to obtain the *Full Availability* softgoal. On the other hand, the *System* depends on the *System Administrator* for executing the *Recovery From the Scratch* when the system fails. The consequences of the pattern inform us of the advantages or drawbacks to apply this pattern in our system. Taking this pattern would improve the fault tolerance and the recoverability of the system-to-be ('+' sign).

In the rest of the paper we will use a *Mail Server System (MSS)* as case study to support the description of RECSS. The actors in the environment of a MSS identified are enumerated in Table 2. For each actor we include the type of the actor and a short description of its main goal. The environmental model (see Fig. 2), represented by means of a SD diagram, allows the representation of the dependencies among the environmental actors and the system. In actor identification we use the patterns of Table 1, which were selected from the full repository taking into account the initial requirements of the system and the problem stated in the pattern. One of the patterns we apply for the *Mail Server System* is the *Full Availability* pattern, since one of our initial requirements is that "The mail server system shall work 24 hours 7 days a week". Taking the actors in the pattern solution and applying them, we identify two environmental actors, which are the *Mail Server User* and the *Mail Server Administrator*. In this case the pattern states that the *System User* and the *System Administrator* are human actors.

Table 1. Catalogue of environmental patterns (excerpt).

Name	Definition
Full Availability	Problem: The system shall offer full availability Consequences: + fault tolerance; + recoverability
Easy Administration	Problem: The system shall be easy to administrate Consequences: + operability
Efficient Operation	Problem: The system shall provide efficiency in some of its functionalities Consequences: +time behaviour; + resource utilization
Client-Server	Problem: 1) The server shall provide the functionalities offered to the users of the client. 2) The client must give data to the system to perform its functionalities Consequences: + suitability
Firewall	Problem: The server shall be protected from unauthorized accesses. Consequences: + security; - time behaviour

Needless to say, environmental actors may sometimes not be identified with the existing patterns. It becomes necessary to add them by hand and later decide if they are part of a new pattern to be added to the repository.

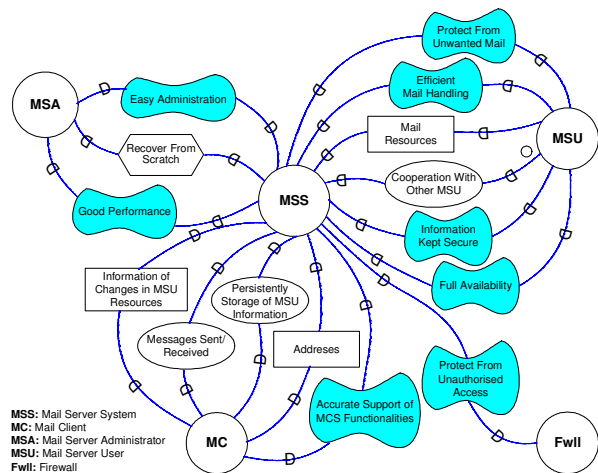


Figure 2. SD diagram for the environmental model.

Table 2. Environmental actors for a MSS.

Actor	Abb.	Type	Goal
Mail Server System	MSS	Software	Provide communication infrastructure
Mail Client	MC	Software	Provide access to messages
Mail Server User	MSU	Human	Send and get messages
Mail Server Administrator	MSA	Human	Put mail server to work accurately and efficiently
Firewall	Fwl	Hardware	Filter incoming requests

4. DECOMPOSING THE SYSTEM

During this activity the system in the environmental model is decomposed according to its functionality, to facilitate its individual analysis. We do not take a component-oriented view but an actor-based one: we identify the actors that play a role in the system and the relationships among them. Actors represent atomic COTS domains that may be deduced from the marketplace and maintained in a taxonomy [5]. More precisely, the procedure followed in this activity is: 1) create a SR diagram for the system actor with its goal as root; 2) decompose recursively this goal into other intentional elements until no more detail is needed; 3) reassign the environmental dependencies identified in the SD model to the obtained intentional elements; 4) create subactors of the system corresponding to software domains that fulfill the goals implied by the intentional element; 5) allocate the intentional elements into these subactors; 6) identify new dependencies among intentional elements of the subactors. Steps 2 to 6 take place iteratively and they intertwine as required.

Table 3 abstracts the most important information of the SR diagram corresponding to the MSS actor. Each actor corresponds to a software domain. We group them into categories. We skip actors' goals for brevity. The third column shows the environmental model dependencies that are allocated to each actor. These dependencies are used to guide the identification of intentional elements while refining the SR model: a dependency usually induces an intentional element to take care of its satisfaction.

5. SYSTEM ACTORS QUALITY MODELS

Once the actors that play a role in the system have been identified, RECSS prescribes the statement of their quality by means of ISO/IEC-9126-1-based quality models. Several methods exist for building models using this standard (e.g. [1, 8]). We focus on a fundamental issue not targeted in these proposals, namely the definition of system quality parameters that influence the value of the metrics. We distinguish environmental and platform parameters.

5.1 Environmental parameters

Environmental parameters reflect the state of the environment where the system operates. They often refer to the number of instances of a particular actor in the system. Examples are the *number of registered users in e-learning platforms*, or the *number of simultaneous connections supported by videoconference systems*; both of them impact on the ISO/IEC Time Efficiency quality feature. Sometimes parameters also refer to particular features of actors, such as the *type of workstation* (dump, PC, mobile device, etc.) *used to visualize*

Table 3. Environmental actors for a MSS.

Category	Actors	Rationale
Communication Support	Mail Servers	• Messages Sent/Received
	Routing Tools	• Efficient Mail Handling
Groupware Support	Meeting Scheduler Tools	• Co-operation With Other MSU
	Voice and Video-Conference Tools	
	Chatting Tools	
	Instant Messaging Tools	
	News Servers	
	Lists Servers	
Resources	Directory Services	• Mail Resources • Addresses • Persistent Storage of MSU Information
	Compression Tools	• Mail Resources
Security Support	Anti-Spam Filter Managers	• Protect from Unwanted Mail
	Anti-virus Tools	• Information Kept Secure
Administrative Support	Backup and Recovery Tools	• Full Availability
	Message Tracking Tools	• Efficient Mail Handling • Good Performance
	Configuration and Administration Tools	• Easy Administration • Recover from Scratch

documents stored in a document management system; in this case, the involved ISO/IEC quality features are Attractiveness and Suitability. Often, environmental parameters have little margin of negotiation due to organizational constraints.

The MSS case provides a few but quite representative environmental parameters, some of which are summarized in Table 4; we include the ISO/IEC quality features affected by each of them. Some are evident enough as to be identified even before defining the metrics, more precisely when building the environmental model (e.g., number of registered users).

Table 4. Environmental parameters of the mail server

Parameter	ISO/IEC Quality Features
Number of registered users	Time Efficiency, Resource Utilization
Number of simultaneous connections allowed	Time Efficiency, Fault Tolerance
Percentage of connections by mail client type	Time Efficiency, Fault Tolerance, Security, Interoperability
Average amount of information managed	Time Efficiency, Resource Utilization

5.2 Platform parameters

When considering software systems behavior, some quality features such as Time Efficiency, Security and Fault Tolerance are greatly influenced by the underlying platform where the system is going to operate. As far as we know, current COTS selection methods and techniques proposed so far do not deal explicitly with this issue; this characteristic makes an important difference from the current practices in the industry. Platforms can be configured in several ways by defining some politics for managing their hardware components. Because of their influence on the quality of the overall system, different combinations have to be evaluated and their benefit and drawbacks must be carefully analyzed.

Table 5 presents some platform strategies which influence the actors in the mail servers system. The presentation of their precise meaning exceeds the scope of the paper, we refer to [2] as a good introductory book. We show in the second column which ISO/IEC 9126-1 quality features are influenced by each strategy, either positively (+) or negatively (-). As an example, the use of clusters allows different recovery mechanisms to be implemented. For instance, clusters can be configured in a way such that their servers perform the same activities and are usually replicated so they can backup each other in case of failure. This policy supports Fault Tolerance. Also clusters can be configured to provide load balancing improving Response Time. The price to pay in this scheme is the administrative duties for maintaining properly the clusters.

Each platform strategy is based upon particular attributes which determine their influence on the system. These attributes are treated as platform parameters in RECSS. They are usually more complex to deal with than environmental ones. The third column of Table 5 presents some of them. They refer to the diversity of equipment, the amount and power of equipment units of a particular kind and the hardware architecture (including software governing it). Just to mention one example: Fault Tolerance is greatly influenced by the layout of servers in the architecture. Servers may be organized into clusters, and clusters may be managed following diverse strategies (see Table 5, row 1) that affect this quality feature. Also, organization of discs concerning RAID level has a great impact on Fault Tolerance.

Table 5. Platform parameters and quality features influenced

Platform	Quality Features	Parameters
Cluster	Fault Tolerance (+) Time Efficiency (+) Operability (-)	<ul style="list-style-type: none"> Type(Active-Active, Active-Passive, ...) Nb. of servers in the cluster
RAID	Fault Tolerance (+) Time Efficiency (+) Operability (-)	<ul style="list-style-type: none"> Level 1 (1 though 5) Level 2 (level 1 combinations)
Load Balancing	Time Efficiency (+)	
Replication	Fault Tolerance (+) Recoverability (+) Time Efficiency (-)	<ul style="list-style-type: none"> Type (one way, two ways, ...) Scope (full, selective) Location (local, remote, both) Number of replicas
Single Server	Maturity (+) Operability (-)	<ul style="list-style-type: none"> Location (centralized, remote) Operating system Hw. resources (processor, ...)

5.3 Putting requirements into quality models

In some sense, RECSS can be viewed as a method for relating system requirements to quality features of the components to be procured. For this reason, we aim at bridging the gap between quality features and requirements. One way to do so is including the latter in the quality model that contains the former. Then, when starting a requirements elicitation and analysis process involving a quality model, the requirements therein form a baseline upon which the process can run.

Once requirements are translated in terms of the quality features, both the informal statement and the translation can be stored together with the involved features to be reused in future projects. Of course, just those requirements that are not too system-specific appear in the quality model. Usually, they must be abstracted from the current system to become reusable.

Some of the system requirements bound to the model are generic, e.g. *The system shall allow [Y] simultaneous connections* where *Y* stands for the environmental parameter defined in Table 4, row 2. Other generic cases appear when considering the domain of the attribute as a parameter, e.g. *The system shall provide an interface written in the following languages: [P(Z)]*, being *Z* the domain of all the possible values of the worldwide languages and *P(Z)* the set of elements from *Z*.

Quality features can have more than one system requirement bound. There are four main reasons behind:

- Requirements are identical but presented differently. For instance, *Message throughput shall be [X] Mb per second* and *Message throughput shall not exceed [X] seconds for messages with an average size of [Y] Mb*.
- Requirements are compatible but at a different level of abstraction. E.g., the feature *Encryption Algorithm* may have bound the requirements *The system shall encrypt messages before sending them* and *The system shall apply an [X] bits encryption algorithm to messages before sending them*.
- Requirements state different things, not necessarily exclusive. For instance, the requirements *Messages shall include a To, a CC, an a BCC field* and *The To field shall allow for multiple destination addresses*.
- Requirements state mutually exclusive conditions. For instance, the Installability-related requirements *The system shall be deployed using open source support* and *The system shall be deployed using licensed tools*. Of course these two requirements cannot be selected together.

6. ANALYSIS AND ELICITATION OF REQUIREMENTS

Once all actors' quality models have been built, we can use them as an aid for analysing and refining existing requirements, and also for suggesting new ones. As a result, we usually discover some ambiguities and incompleteness that are necessary to solve before proceeding further in the procurement process.

6.1 Analysing system requirements

System requirements should be analysed before performing any kind of compliance test. Doing so, we may discover ambiguities and other similar problems, and we may obtain a more structured expression of requirements, before investing time in their evaluation. The different artefacts proposed in RECSS can be used with this aim. The usual situations found are: (1) refinement: high-level requirements can be made operational using the quality features that are in the individual quality models; (2) decomposition: a requirement that involves many actors may be decomposed into others that apply to these actors; (3) reformulation: the adoption of environmental patterns and/or platform strategies that relate to an ISO/IEC quality feature with different tendencies (i.e., supporting vs. conflicting with) indicates conflicting requirements which at least should be identified and possibly negotiated.

As an example, let's consider a typical requirement on mail servers such as *Message transmission time shall take less than 1 minute*. We process the requirement as follows:

Step 1. Determine which are the involved system actors of the quality model. The goals of system actors are compared with the requirement. In our case (see Table 2), we obtain as initial set of actors' domains the *Mail Servers* (to send the message), the *Directory Services* (to obtain the addressee) and the *Routing Tools* (to compute the route).

Step 2. Determine which are the involved features of the quality models for these actors' domains. Although there are others (of course at least one feature for domain), we consider here the most influential one, namely *Message Throughput* in the mail server domain.

Step 3. Complete the definition of these features, if needed. *Message Throughput* needs not to be further decomposed, but let's assume that its metrics is still not defined. In this case, a careful investigation is required. We analysed the information coming from a lot of sources, including widespread benchmarks such as the Microsoft MMB2. In addition to some platform parameters enumerated in Table 5, we discovered some environmental ones that influence the feature such as *Expected Average Message Size*.

Step 4. Analyse if the requirement is sound and complete with respect to the quality model. If not, reformulate it and carry out a regression analysis. All the benchmarks that we looked up provide different tables for different messages sizes. For this reason, we conducted a negotiation with stakeholders to reformulate the requirement to take this factor into account as: *Transmission time shall take less than 30 seconds for messages without attachments, and less than 2 minutes per Mbyte for those with attachments*. Therefore we go back to the previous activity and since space reduction becomes important, we introduced a *Compression Tools* subactor in the SR diagram in order to provide efficient data compression facilities for attachments (i.e., the *Compression Tools* actor, linked with the mail server with the *Compress Message* soft goal dependency, which refers to the *Resource Utilization* quality feature). We are compelled therefore to refine also this part of the model.

Step 5. Make a first and fast compliance test to decide whether the requirement is feasible or not. If it isn't, reformulate it. Concerning our example, this step means to test if the available benchmarks show that the requirement can be satisfied, taking into account those parameters whose value is known and non-negotiable, such as the number of users.

6.2 Generating system requirements

An interesting characteristic of our approach is that we may use RECSS' artefacts to help in the elicitation process, suggesting requirements that are left. More precisely, we find the cases:

- Parameter elicitation. In order to compute the metrics of the quality models, it is necessary to know the value of all the system parameters. Then, missing requirements providing these values (e.g., *The expected average message size for messages with attachments is 100 Kbytes*) must be elicited.
- Pattern-related requirements elicitation. Environmental patterns are explicitly related to ISO/IEC quality features such as Time Efficiency, Accuracy, etc. If there exists some goal in the system concerning those features (e.g., *The system shall be accurate*) we can investigate which patterns

support it and analyse its viability presenting the associated requirements to the requirements engineer. On the other hand, when a pattern P has been selected, there are two consequences: first, all the requirements of the pattern are considered as system requirements, even if not initially stated; second, we search for other patterns that support or conflict with the same quality features as P and we present the requirements to the engineer, who may decide to apply one of these patterns.

- Quality requirements elicitation. The requirements engineer may analyse the requirements bound to quality features that are related with those ones that correspond to existing requirements. For instance, the requirement presented in 6.1 was operationalized in terms of the quality attribute *Message Throughput* that belongs to the *Time Efficiency* quality feature. A sibling of this attribute is *Server Response Time* that has some parameterised requirements bound, for instance *The server must respond to at least the [X]% of connection requests in less than [Y] milliseconds*. So, this is a good candidate of requirement to be included.

7. PREVIOUS AND RELATED WORK

This work is based upon previous results of our group. A general introduction to our experiences in COTS selection appears in [4]. In [3] we used a similar framework to solve the problem of constructing quality models for composite systems. We also defined four activities and the first three were similar to the ones here, but with significant differences: we didn't use goal models at all; we didn't use patterns for modelling the system environment; we didn't formulate a precise procedure for decomposing the system; we didn't formulate the concept of system parameter (environmental or platform). All of these issues have been fundamental in this paper.

Other approaches exist that use quality models for requirements engineering in the COTS context [1, 8, 9, 18], although the use of goal models together is not as usual [6]. Differences with these approaches are:

- We provide more precise guidelines than these approaches.
- From the point of view of quality, they consider component-based systems as single monolithic units.
- They are oriented to the construction of costume-made software. In our approach we consider models to evaluate COTS components.
- They do not consider any means of reusing the knowledge gained in the experience (e.g., the requirements themselves), whilst RECSS may use taxonomies, patterns and catalogues.
- They do not consider the platform strategies required to deploy the systems nor the system parameters required by the metrics of the identified quality attributes.

With regard to the use of *i** as a pattern specification language, some approaches are also found. [14] proposes the use of organization patterns, their objective being also to discover requirements, but starting from a different model, independent of the software system to be developed. Organization patterns as presented in this work are more coarse-grained than our environmental patterns; in fact, both approaches seem to be complementary because we could think on defining organization patterns as a combination of environmental patterns. We also mention [19] as an approach that uses patterns to build agent

systems that follows a set of concrete non-functional requirements. Although the principle behind the proposal is close to ours, it differs in the target of the method, which seems to be a role diagram with no dependencies.

8. CONCLUSIONS

In this paper we have proposed a method called RECSS to drive the requirements analysis and elicitation activities in the context of COTS-based software systems using quality and goal models. The combination of these two types of models allows dealing with two different axis of quality requirements in COTS-based systems: compositionality and intentionality are covered by goal models, whilst operationalization by quality models. The method may promote return on investment by organizing the knowledge available from previous experiences using some repositories of the presented artifacts (e.g., taxonomy of domains, catalogue of patterns and system parameters, etc.). The method has arisen from our experience in procurements processes we have participated in more than a dozen of software domains. Dealing with real industrial experiences has been crucial to identify some points that we have incorporated in our proposal, such as the consideration of the platform strategies and their consequences, and the adoption of requirements-driven politics to manage the repositories and models of RECSS.

9. ACKNOWLEDGMENTS

This work has been partially supported by the CICYT programme project TIN2004-07461-C02-01.

10. REFERENCES

- [1] L. Beus-Dukic, J. Bøegh. "COTS Software Quality Evaluation". In *Proceedings 2nd ICCBSS*, 2003.
- [2] R. Buyya. *High Performance Cluster Computing*. Prentice-Hall, 1999.
- [3] J.P. Carvallo, X. Franch, G. Grau, C. Quer. "COSTUME: A Method for Building Quality Models for Composite COTS-Based Software Systems". In *Proceedings 4th QISIC*, 2004.
- [4] J.P. Carvallo, X. Franch, C. Quer. "Determining Criteria for Selecting Software Components". *IEEE Software*, 24(3), 2007.
- [5] J.P. Carvallo, X. Franch, C. Quer, M. Torchiano. "Characterization of a Taxonomy for Business Applications and the Relationships Among Them". In *Proceedings 3rd ICCBSS*, 2004.
- [6] P. Donzelli, P. Bresciani. "Goal oriented requirement engineering: a case study in e-government". In *Procs. 15th CAiSE*, 2003.
- [7] R. Dromey. "Cornering the Chimera". *IEEE Software*, 13(1), 1996.
- [8] X. Franch, J.P. Carvallo. "Using Quality Models in Software Component Selection". *IEEE Software*, 20(1), 2003.
- [9] D. Firesmith, "Using Quality Models to Engineer Quality Requirements". *Journal of Object Technology*, 2(5), 2003.
- [10] A. Finkelstein, G. Spanoudakis, M. Ryan. "Software Component Requirements and Procurement". In *Proceedings 8th IEEE IWSSD*, 1996.
- [11] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, 1995.
- [12] IEEE Std 1061-1998. *IEEE Standard for a Software Quality Metrics Methodology*, 1998.
- [13] *ISO/IEC Standard 9126-1 Software Engineering – Product Quality – Part 1: Quality Model*, 2001.
- [14] M. Kolp, P. Giorgini, J. Mylopoulos. "Organizational Patterns for Early Requirements Analysis". *Proceedings 15th CAiSE*, 2003.
- [15] B. Kitchenham, S.L. Pfleeger. "Software Quality: the Elusive Target". *IEEE Software*, 13(1), 1996.
- [16] B.C. Meyers, P. Oberndorf. *Managing Software Acquisition*. Addison-Wesley, 2001.
- [17] A Mohamed, G Ruhe, A Eberlein. "COTS Selection: Past, Present, and Future". In *Proceedings 14th IEEE CBSE*, 2007.
- [18] A. Rawashdeh, B. Matalkah. "A New Software Quality Model for Evaluating COTS". *Journal of Computer Science*, 2(4), 2006.
- [19] M. Weiss. "Pattern-Driven Design of Agent Systems: Approach and Case Study". *Proceedings 15th CAiSE*, 2003.
- [20] E. Yu. "Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering". In *Proceedings 3rd ISRE*, 1997.

11. BIOGRAPHY OF AUTHORS

Juan Pablo Carvallo is the manager of the Informatics department of Etapatelecom, a telecommunications company based in Cuenca, Ecuador, and a part-time teacher at Azuay University's Systems Engineering Department. His research interests include requirements engineering, COTS-based system development, and COTS selection, evaluation, and certification. He received his PhD in informatics from the Universitat Politècnica de Catalunya.

Xavier Franch is an associate professor at the Universitat Politècnica de Catalunya's Department of Software. His main interests include requirements engineering, COTS-based development, and software quality. He's a member of the IEEE. He received his PhD in informatics from the Universitat Politècnica de Catalunya.

Carme Quer is an associate professor at the Universitat Politècnica Department of Software. Her research interests are selection of COTS components, and component-based software development. She received her PhD in informatics Universitat Politècnica de Catalunya.