Building and Using Quality Models for Complex Software Domains[¥]

Juan P. Carvallo[‡], Xavier Franch, Carme Quer
Universitat Politècnica de Catalunya (UPC)
c/ Jordi Girona 1-3 (Campus Nord, C6) E-08034 Barcelona (Catalunya, Spain)
{carvallo, franch, cquer}@lsi.upc.es

Abstract

The use of quality models in software package procurement provides a framework for the description of the domain which the package belongs to. Package descriptions and user quality requirements may be translated into the quality concepts defined in the model making package procurement more efficient and reliable. In this paper we address the construction of quality models for complex software domains, defined as domains that imply a mixture of functionalities. Procurement processes taking place in complex domains require not a single package to be selected but a set of them. As a consequence, instead of a standard, single quality model, we need a more elaborated quality model for driving the simultaneous procurement of multiple software packages. We describe the parts that compose these kind of models, the methodology for building them and their usage in software procurement. We apply the approach to the complex domain of mail server systems.

1. Introduction

The impact of quality requirements during *software package procurement* [1] has been recognized as crucial by the methodologies and processes proposed so far for driving this activity [2, 3]. Therefore, there is a need for obtaining, in an efficient way, reliable and comprehensive descriptions of software package quality.

Quality models [4, 5, 6] are a specially appealing way of structuring these descriptions. A *quality model* for a given *software package domain* (i.e., a domain for which software packages may be bought, downloaded or obtained by whatever means) provides a taxonomy of software *quality features* and *metrics* for computing their value during package procurement. Once a quality model is available, package descriptions and quality requirements may be translated into the quality concepts defined therein, making package procurement more efficient and increasing the confidence in its result (see fig. 1).

In [7, 8] we presented a methodology aimed at building ISO/IEC quality models for software package domains, and we explored its usage in the context of package procurement. This methodology proved to be quite useful, but we discovered some drawbacks that hamper its applicability. These drawbacks stem from two different sources. On the one hand, usual quality models such as the ISO/IEC-based do not include explicitly any description of neither the environment where packages are going to operate nor the underlying hardware platform. On the other hand, packages may be required to offer some functionalities that they do not provide directly.

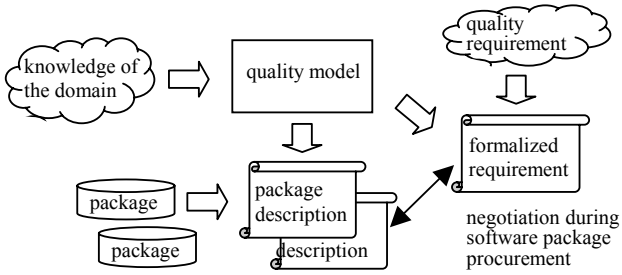

Figure 1 Using a quality model in software procurement.

These issues are especially relevant when considering *complex software domains*, i.e. domains that offer a mixture of functionalities. Complex domains fall into categories such as groupware, communication (mailing, videoconferences, etc.), document management, and others that currently play an important role in the daily functioning of medium- and large-size companies. In addition to the domain of interest (the *main domain*), complex domains are composed of some *secondary domains*, which are on account of two different reasons:

- Facilities such as anti-virus and compression tools, directory services, tracking tools, etc., have become widely established and their presence is assumed in most complex domains.
- Successful packages tend to bind applications that were not originally related to them. A usual reason for this is that product suppliers often include features to differentiate their products from their competitors'. After some months or years, these added functionalities may become standard in the domain.

Due to this diversity, procurement processes taking place in complex domains require not a single package to be selected but a set of them, which must be integrated into a software system (sometimes known as *Commercial-Off-The-Shelf-based system*, or *COTS system* [9]). As a

¥ This work is partially supported by the Spanish research programme CICYT under contract TIC2001-2165.
‡ Juan P. Carvallo's work has been supported by an AECI grant.

consequence, instead of a standard, single quality model for a kind of software package, we need a more elaborated quality model for driving the simultaneous procurement of multiple software packages.

The objective of this paper is to identify and define the parts that compose quality models suitable for complex software domains; to propose a methodology for building these quality models; and to illustrate their usage in the context of multiple package procurement. Our discussion is conducted by a particular example, the construction of a quality model for the domain of mail server systems. It is worth to remark that this example was also chosen in [7]; this fact allows easier comparison of the current proposal with our previous work.

The rest of the paper is organized as follows. Section 2 introduces the overall structure of quality models for complex domains and presents an overview of the proposed methodology. Section 3 presents the mail server case study. Sections 4 to 6 apply the methodology to this case study. Section 7 illustrates the usage of this kind of quality models in software procurement. Finally, section 8 gives the conclusions and the current work.

2. COSTUME: A Methodology for Building Quality Models for Complex Domains

We describe in this section the COSTUME methodology, aimed at defining quality models for complex software domains (abridged, *complex quality models*). COSTUME (*CO*mplex *S*of*T*ware domain q*U*ality *M*odel d*E*velopment) consists of four activities that are presented below, which may be intertwined or iterated as required. The result of COSTUME is a complex quality model structured in three main components: description of the environment of the complex domain; individual ISO/IEC 9126-1-based quality models for the domains which the complex domain is divided into; and a high-level description of the hardware platform where systems from the domain are to be installed.

Furthermore, it should be remarked that complex quality models are parameterised, being the parameters collected from the complex quality model components enumerated above. *Quality parameters* are defined as those features whose values affect somehow the behaviour of one or more quality attributes. They may be classified as *environmental parameters* and *platform parameters*, depending on the part of the model they arise from. Environmental parameters use to be fixed and with little margin of negotiation due to organizational constraints, whilst the nature of platform parameters is more diverse.

Activity 1. Describing the environment of the domain

Systems do not operate in isolation; they communicate with other systems and with people, which act together as their environment. From the requirements engineering point of view, it is utterly important to make explicit this environment to elicit and possibly negotiate requirements on the system during procurement. Therefore, the first component of a complex quality model is the *environment model*, which describes the *actors* in the environment and the most important dependencies that can be found among them and the system.

We distinguish four different types of actors in the environment of the system: human, representing different types of users; organizations, providing information or logical resources to the system; hardware resources, as mechanical devices governed by the system or providing data to the system; and other software systems, which provide data to, or collect data from, the system.

Although this first model may seem simple to build, it is not always the case. On the one hand, some of the actors may not be evident. In the case of people, the concept of user may be confused with the wider one of stakeholder; we remark that just those actors participating in the system once in operation (such as end-users and the system administrator) should appear in the environment model (excluding stakeholders such as the sales manager). Concerning organizations and software systems, some actors may remain hidden until the complex domain is examined in more depth; for instance, in the document management domain, the need for identifying an actor for performing document imaging (i.e., the conversion of paper documents into electronic images) may not appear until some document management products have been examined. Last, hardware actors may be confused with platform components, which must appear in another part of the model. Hardware actors are clearly out of the boundaries of the system and they are usually related to some of the functionalities of the domain; an example of hardware actor is a scanner for the domain of document imaging. On the contrary, a cluster of servers must be considered as a platform component whose existence remains hidden to system users. The iterative nature of COSTUME may help to discover hidden actors and avoid these confusions.

One of the subactivities involved in activity 1 is the determination of the environmental parameters. They often refer to the number of instances of a particular actor in the system. A typical case is 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 subcharacteristic. Sometimes parameters also refer to particular features of actors, such as the type of workstation (dump, PC, mobile devices, etc.) used to visualise documents stored in a document management system; in this case, the involved ISO/IEC subcharacteristics are attractiveness and suitability.

Activity 2. Building a quality model for the main domain
We have already mentioned that complex domains are composed of a main domain and some secondary domains. Main domains enclose those functionalities that are the main target of the complex domain, e.g.: providing document management and tracking in document management; providing communication infrastructure in mail servers; automation of business processes in workflow technologies; etc. Activity 2 focus on the construction of a quality model for this main domain.

In [7, 8] we have proposed a methodology for building a ISO/IEC-based quality model for a software domain considered in isolation. This methodology follows six steps for tailoring a departing quality model proposed as part of the ISO/IEC 9126-1 standard. To be more precise, the standard determines six quality characteristics and their decomposition into subcharacteristics. From this starting point, our proposal:

- Adds new subcharacteristics specific to the domain, refines the definition of some existing ones, or even eliminates some.
- Creates a hierarchy of quality subcharacteristics.
- Decomposes subcharacteristics into quality attributes, which keep track of particular observable features of packages in the domain.
- Decomposes complex attributes into simpler ones, which are directly measurable.
- State relationships among quality entities, which allows more accurate analysis of requirements.
- Determines metrics for the measurable attributes.

Quality model construction is endangered by many risk factors. We propose several tips to try to overcome these usual difficulties; see [7, 8] for further details.

Activity 3. Identifying and building quality models for the secondary domains
During the construction of the quality model above, functionalities other than the main ones will be discovered and reported. Some of them will be bound to a particular complex domain or a category of them, such as the one of routing tools (bound to communication support domains). Others will be more general, such as the domain of anti-virus tools. In any case, in order to get a complete quality model for the complex software domain, we require individual models for these secondary domains to be built too.

At a first glance, this activity may seem cumbersome but in fact, we argue that it pays off. On the one hand, it supports return of investment through model reuse; e.g., once a quality model for anti-virus tools has been built, it may be reused in a great deal of domains. On the other hand, quality models for secondary domains may be left incomplete. This second guideline aligns with the observation that requirements on secondary domains are usually not as detailed as the ones on the main domain. For instance, when referring to anti-virus tools in the context of e-learning, requirements are often something as vague as "The documents managed by the system shall be not infected by virus" without specifying e.g. which actions are required once the virus is detected. Therefore, refinement of secondary domains may be delayed until a particular procurement process requires more detail.

As mentioned earlier, package procurement in complex software domains results in a set of packages to be selected. Secondary domains provide a guide for identifying which categories of packages should be analysed. However, it is worth to remark that the mapping from secondary domains to software packages is not one-to-one but many-to-one or even many-to-many. This is due to the fact that successful packages in a domain tend to incorporate features and functionalities which they were formerly not intended for. Examples in the ERP systems domain are the inclusion of graphical-statistical and reporting facilities, and of workflow and project management engines. For this reason, it becomes utterly important to decouple the decomposition of the domain into actors from the decomposition of the system into packages. Assignment of packages to system actors become then a point of study that we have already addressed in other work [10] and that will be mentioned here in section 7.

Activity 4. Characterising the underlying platform
When considering complex software domains, some quality subcharacteristics 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 quality model proposals do not pay attention to this platform and this fact hampers the use of quality models in real procurement experiences.

For this reason, we have decided to incorporate also a platform model into the quality model. A *platform model* includes the definition of those hardware architectural patterns and elements that are relevant in the domain. Of course, we are interested only in high-level architectural patterns and elements, discarding low-level equipment such as dispatchers and hubs.

Platform model of the complex domain is built from the platform models of its component domains. Therefore, there is a tight relationship between this activity and the previous one.

As in the case of the environmental model, platform parameters must be identified. They are usually more complex to deal with than environmental ones. 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). We list a couple of examples below, pointing out which ISO/IEC subcharacteristics are they referring to:

- In a great deal of complex domains, time efficiency is greatly influenced by the number of servers in the system and their characteristics (CPU, …).
- Fault tolerance is greatly influenced by the layout of servers in the architecture. Servers may be organized into clusters, and cluster may be managed following diverse strategies (active-active, active-passive, etc.) that affect this subcharacteristic. Also, organization of discs concerning RAID level has a great impact on fault tolerance.

When defining quality attribute metrics, these platform characteristics may become as important as the quality model features themselves. For instance, regarding security issues, influence of secure protocols is comparable to software security capabilities such as authentication of users or virus detection.

3. A Complex Software Domain: Mail Servers

Mail servers (also known as *message servers*) are the core of the communication and coordination infrastructure of companies of any type. The mail servers domain is a good case study for many reasons; to name some: its structural properties (wide range of functionalities, strong influence of platform, etc.); its intensive use worldwide; and the existence of an overwhelming number of mail-related products. A successful mail service deployment depends on its correct selection and a quality model may be used as the cornerstone for this process.

In [11] we built a quality model for the mail servers domain following the methodology given in [7, 8]. As mentioned before, we discovered some flaws in this model, namely:

- The quality model considered the mail server domain as isolated. We realized that, due to the complexity of the domain, it is utterly important to identify the boundaries of the system more accurately, determining which people, organizations, hardware resources and other software systems interact with it.
- In addition to their main functionality, i.e. supporting mailing facilities, mail servers are supposed nowadays to offer other additional functions, such as videoconference infrastructure, virus detection, backup and recovery facilities, etc. We dealt with these features directly in the mail server quality model itself, using the ISO/IEC suitability and interoperability subcharacteristics, increasing substantially the size of the model and damaging understandability and reuse of the result.
- When defining metrics for the attributes to carry out compliance checking, we discovered that some of them depend on both environmental and platform properties, such as number of existing users and the type of server clustering. Without this information, the metrics become incomplete and useless.

As shown in section 7, the inclusion of these elements in the quality model can be a great help for requirements engineering during package procurement.

In the next sections we are going to illustrate the COSTUME methodology for building complex quality models using the domain of mail servers as case study, aiming at solving the problems mentioned above.

4. Building the Environmental Model

We arrange the construction of the environmental model into the following steps.

Step 1. Identification of the environmental actors

Table 1 presents a first proposal of environmental actors for the mail servers domain, including the mail server system itself. We identify the actor's type and also give a short description of the actor's goal. Two important points are worth to remark:

- Complex domains such as the mail servers one will always require one or more administrator actors to make the system able to run.
- The mail user and mail client actors must be clearly distinguished. The mail client is just a means that the mail user employs to achieve its goals. This situation is found again in other domains (e.g., videoconference) in which the client-server architectural model is used.

Actor	Abb.	Type	Goal
Mail Server System	MSS	Software	Provide communication infrastructure
Mail Client System	MCS	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	Fwll	Hardware	Filter incoming requests

Table 1 Environmental actors of the mail server.

Step 2. Statement of dependencies among actors

To represent the most important dependencies among actors we build an *i** SD model [12]. We focus on dependencies involving the mail server system (thus, dependencies among the user and the mail client would appear in the mail client environmental model itself).

Fig. 2 shows the *i** SD model for the mail server case. Goals and resources have to be with the main functionalities that mail servers offer, namely mailing, cooperation facilities, address lists management and administrative duties. Soft goals are identified following the ISO/IEC subcategories; for instance, we find soft goals concerning security, efficiency and usability. We

remark that also the mail server behaviour may depend on environmental actors, as reflected by the two dependencies stemming from MSS to MSA (e.g., mail server performance depends on the administrator ability to perform an appropriate tuning of the system).

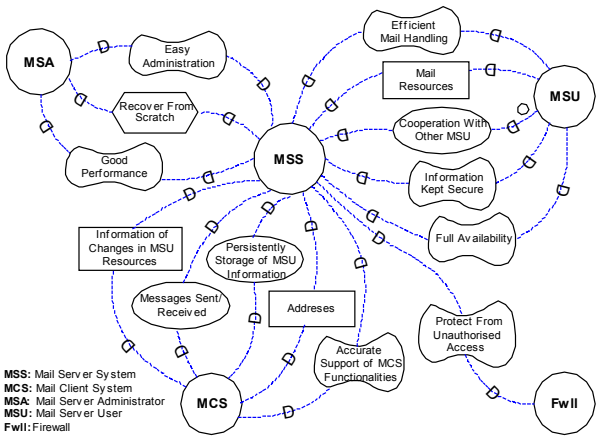

Figure 2 SD diagram for the environmental model.

Step 3. Determining environmental parameters

The mail servers domain provide a few but quite representative environmental parameters, some of which are summarized in table 2; we include the ISO/IEC subcharacteristics affected by each of them. Some are evident enough as to be identified during a first iteration of COSTUME, as it happened with the number of registered users; others appeared later, usually when defining metrics for attributes, such as the one for the number of simultaneous connections allowed, in activities 2 or 3.

Parameter	ISO/IEC Subcharacteristics
Number of registered users	time efficiency, resource utilization
Number of simultaneous connections allowed	time efficiency, fault tolerance
Percentage of connections by mail client type (program, webmail, …)	time efficiency, fault tolerance, security, interoperability
Average amount of information managed	time efficiency, resource utilization

Table 2 Environmental parameters of the mail server.

5. Building and Composing the Individual Quality Models

Activity 2 as presented in section 2 is devoted to the construction of a quality model for the main domain. In the case of mail servers, we have developed this model in [11]. For this reason, we focus on activity 3, which is again defined as a sequence of steps.

Step 1. Identifying secondary domains

The dependencies in the above environmental model reflect the existence of a set of secondary domains in the complex domain of mail servers. For instance, the goal dependency *Cooperation With Other MSU* is related with a set of groupware domains, such as those for meeting scheduling and voice and video-conference. In table 3 we show a list with the most relevant secondary domains identified, grouped by category. The third column states the relationship of the domains with the dependencies in the environmental model.

Category	Secondary Domains	Dependencies
Groupware Support	Meeting Scheduler Tools	• Co-operation With Other MSU
	Voice and Video-Conference Tools	
	Document Management and Workflow 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	• Information Kept Secure
	Anti-virus Tools	
Administrative Support	Backup and Recovery Tools	• Full Availability
	Message Tracking Tools	• Efficient Mail Handling • Good Performance • Easy administration
	Configuration and Administration Tools	
Communication Support	Routing Tools	• Efficient Mail Handling

Table 3 Secondary domains for the mail servers case.

Step 2. Obtaining quality models for the secondary domains

We have already mentioned in section 2 that the situation with respect to the construction of quality models for secondary domains is diverse. On the one hand, for those secondary models whose quality models are already built from past experiences, we may just reuse them. The rest of the models have to be constructed and may be left incomplete and not refined until a particular procurement process requires more detail. In this case study, we have decided to construct just the environmental model of the secondary domains.

For the sake of brevity, we present here the quality models for only three of the secondary domains identified: the *Anti-virus Tool* (AVT), the Directory *Server* (DS) and the *Routing Tool* (RT) domains. The environmental models of these domains are presented in figure 3. As it can be observed, in each of these domains we have identified an administrator actor (acronyms that end with an 'A') and some user actors (acronyms that end with a 'U'). Some comments on the figure:

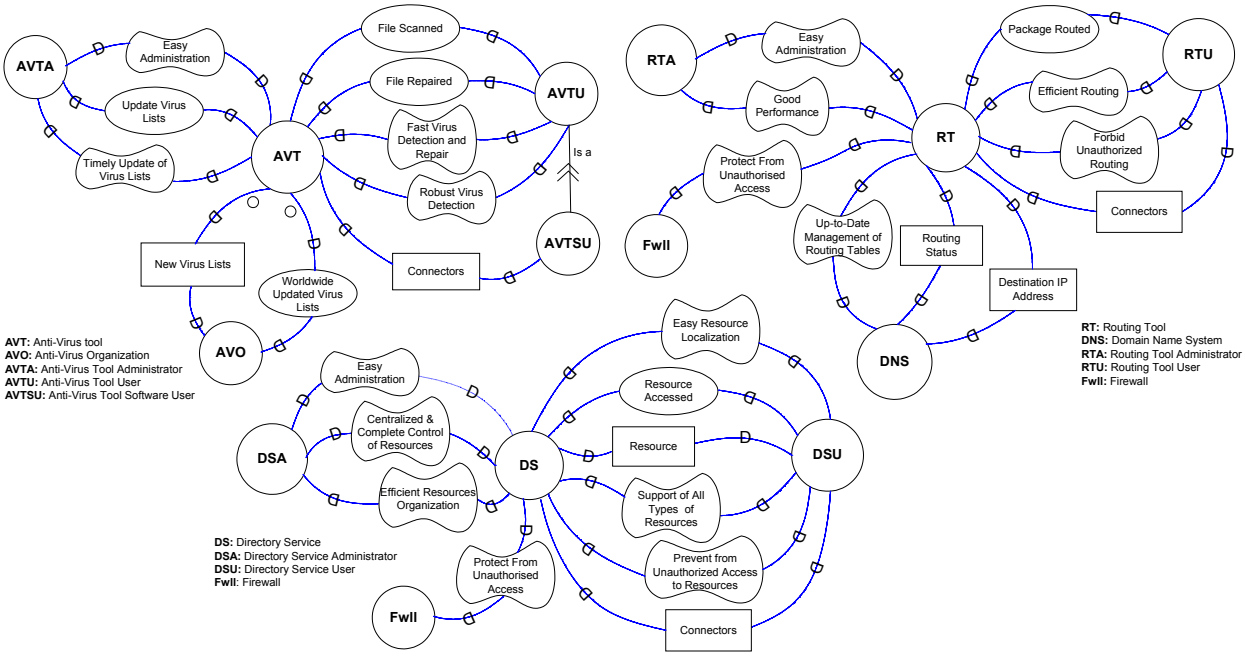

Figure 3 SD diagrams for three secondary domains of the mail server complex domain.

For the sake of brevity, we present here the quality models for only three of the secondary domains identified: the *Anti-virus Tool* (AVT), the *Directory Server* (DS) and the *Routing Tool* (RT) domains. The environmental models of these domains are presented in figure 3. As it can be observed, in each of these domains we have identified an administrator actor (acronyms that end with an 'A') and some user actors (acronyms that end with a 'U'). Some comments on the figure:

- The AVT corresponds to a package needed in a MSS to protect users from mails containing infected files. The AVO (*Anti-Virus Organization*) provides updated virus lists. ATVU stands for any type of anti-virus user, and AVTSU covers the specific case of a software tool using the anti-virus.
- The DS corresponds to a package that is needed in a MSS to maintain resources of MS users. Depending on the type of directory, other information apart from e-mail-related may be stored.
- The RT corresponds to a package needed in a MSS to give communication support. The DNS (*Directory Network Service*) is the actor that manages routing tables from where the RT obtains IP addresses.

Step 3. Combining the quality models to form a compositional system model

Individual quality models, both for the main and the secondary domains, give an exhaustive but individual, isolated view of parts of the complex domain. The next step consists in combining these quality models in order to deduce which dependencies and quality entities of the main domain's quality model depend on quality entities of the secondary domains' quality models.

The composition of the SD diagrams corresponding to the environmental models[2] relies mainly in the concept of *actor combination*. This combination is applied when two actors represent the same in the context of the quality model being constructed. Dependencies of the new actor are defined as the union of the dependencies of the combined actors. In our case, identification comes from two sources:

- The mail server actor corresponds to the user actor in most of the secondary domains. Concerning the three domains we are considering, figure 3 shows that the actors to be combined in the new model with the actor of the mail server system (*MSS*) are: the anti-virus tool software user (*AVTSU*), the directory server user (*DSU*) and the routing tool user (*RTU*).
- The firewall actors (*Fwll*) that appear in the domains of the mail server, the directory service and the routing tool, must be combined.

Once the actors have been combined and the dependencies arranged with respect to this combination,

[2] Since in this example we have restricted the secondary quality models to environmental models, we are not addressing the composition of individual quality entities that would appear in the ISO/IEC-based quality model.

the next action is to redraw the *i** SD environmental diagram of the *MSS* (see figure 2) to convey all this information. Figure 4 is a schematic view of the compositional system model (that can be found complete in the appendix just for referee purposes). The boundary of the system takes combination into account.

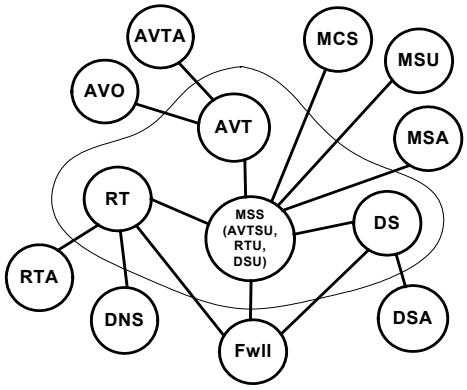

Figure 4 Schematic view of the compositional system model.

Step 4. Deducing dependencies among models

Once the internal structure of the system into actors is known, responsibilities must be assigned to them. In other words, we assign every dependency appearing in the environmental model of the mail server domain into the corresponding actor of the compositional system model.

This final step of the activity is utterly important because an erroneous procurement of a package bound to a secondary domain can provoke that the MSS will not be able to successfully deliver the goals of the actors in the MSS environment.

This step can be supported by the use of a *i** SR diagram for the *MSS* actor (see figure 5). In this diagram we can see that, for example, the MSS will not be able to provide *Efficient Mail Handling* to its users (MSU) if its RT is not able to provide to the MSS *Efficient Routing*.

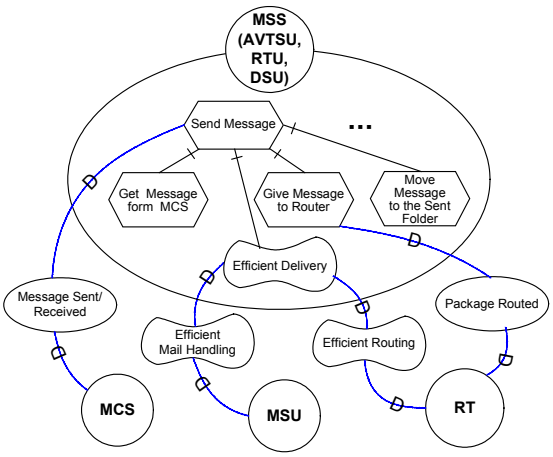

Figure 5 Part of the SR diagram for the MSS actor.

6. Building the Platform Model

Activity 4 of section 2 mentions that some quality subcharacteristics of complex software domains are greatly influenced by the underlying platform. Therefore, in order to provide a fully useful quality model, the platform has to be considered and modelled. Platform models are used to detect possible dependencies between software actors and platform patterns as well as to detect related parameters to be included in the model. This activity is defined by the following steps.

Step 1. Identifying platform patterns and the actors which they may be applied to

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. It is important to keep in mind that the objective is to identify only those high-level patterns and hardware components which directly affect the domain.

Table 4, column 1, presents some platform patterns which influence the actors in the mail servers domain, as shown in column 2. A brief presentation follows [13]:

- *Cluster.* A group of local servers[3], working together in a synchronized manner and sharing their resources. This is the usual way of managing the mail resources such as mailboxes under the control of the MSS actor. On the contrary, DS and RT are not usually arranged in clusters due to their cost and operatibility complexity.
- *RAID.* Acronym that stands for Redundant Array of Independent Disks. The array combines several hard disk drives to improve fault tolerance and performance. MSS, RT and DS use this pattern intensively.
- *Load-balancing*. To distribute processing and communications activity equally between devices of a computer network in such way that no one is overloaded. RT and DS use load-balancing as well as replication (see below) as a kind of substitutive for clusters.
- *Replication*. The process of keeping duplicate copies (replicas) of a database. Replication also synchronizes databases replicas in such way that changes made to one replica are reflected in all the others.
- *Single server.* A dedicated computer on a network that manages network resources, such as files or data bases. This is the kind of hardware element required for the AVT actor.

[3] Do not confuse the "server" hardware component with the mail "server".

Step 2. Analyzing the influence of platform patterns over domain subcharacteristics
Once the platform has been modeled, the influence of its components on the main and secondary domains' subcharacteristics has to be analyzed.

Table 4 shows some results of this analysis in the third column. We state if the influence is positive (+) or negative (-). As an example, the use of clusters allow 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 supports fault tolerance. Also clusters can be configured to provide load balancing improving response time. This is an example of a platform pattern including functionalities from other simpler pattern. The price to pay in this scheme are the administrative duties for maintaining properly the clusters.

Step 3. Identifying platform attributes
Each platform pattern includes particular attributes which tailor their influence on the system. Those attributes are treated as platform parameters in the complex domain quality model. The fourth column of Table 4 presents some of them. As in the case of environmental parameters, platform parameters will be analyzed in each particular procurement process

Platform	Actor	Subchars.	Parameters
Cluster	MSS	Fault Tolerance (+) Time Efficiency (+) Operability (-)	• Type (Active-Active, Active-Passive, etc.) • Number of servers in the cluster
RAID	MSS DS RT	Fault Tolerance (+) Time Efficiency (+) Operability (-)	• Level 1 (1 though 5) • Level 2 (combinations of level 1)
Load Balancing	DS RT	Time Efficiency (+)	
Replication	DS RT	Fault Tolerance (+) Recoverability (+) Time Efficiency (-)	• Type (one way, bi-directional, etc.) • Scope (full, selective) • Location (local, remote or both) • Number of replicas
Single Server	AV	Maturity (+) Operability (-)	• Location (centralised or remote) • Operating system • Hardware resources (processor, RAM, HD, etc.)

Table 4 Platform components and their influence on actors' subcharacteristics.

7. Using the Complex Quality Model during Package Procurement

Once the quality model for a complex software domain is built, it becomes possible to describe packages in this domain and to express quality requirements:

- Quality models provide a general framework to get uniform descriptions of candidate packages, improving the reliability and efficiency of the procurement process.
- Quality requirements can be formulated in a structured manner in terms of the quality concepts appearing in the model. This process may help to discover some ambiguities and incompleteness and, once solved, the resulting requirements can be more easily compared to the package descriptions.

In this section, we explore in more depth how quality models may help requirements elicitation. We propose some procurement activities centred on quality models that could be eventually integrated into widespread methodologies and processes such as OTSO [2] or PORE [3] (we do not address others such as requirements prioritisation, initial candidate screening or final decision). As usual, these activities will take place intertwined and iterated throughout the procurement process.

7.1 Building candidate architectures

Since complex quality models result in multiple packages to be selected, we need to determine which are the candidate architectures to be evaluated. There are some issues to be considered:

- As we mentioned in section 2, system actors do not correspond necessarily to individual software packages. Furthermore, two packages of the same domain may cover different actors, e.g. some mail servers may offer backup facilities whilst others not. This makes arrangement of packages into the architecture more difficult to analyse.
- It may be the case that an actor may be covered by more than one package. Sometimes these repeated coverage is decided during the procurement process, when it becomes evident that no single package satisfy the requirements stated on one of the domains.
- Some of the actors may be covered by packages which already exist in the organization. A typical case is the directory service, which nowadays tend to be a centralised resource, shared by many systems in the organisation. Also it may be probably the case of the anti-virus tool, although negotiation is more likely than in the former case, particularly because most anti-virus vendors provide differentiated versions of their products for their use in mailing products.
- Some packages satisfy the requirements on their corresponding actors, but may be not compatible. Interoperability is a ISO/IEC subcharacteristic which is crucial during this activity.

Taking all these issues into consideration, we may configure the candidate architectures to be evaluated during procurement [10].

7.2 Analyse quality model parameters

The role of quality parameters in software package procurement has been highlighted in previous section. Therefore, attention must be devoted to their nature and the values they take.

Quality parameters may be categorized by different criteria and may impact in different ways on requirement elicitation:

- They may be given or open. Most environmental parameters are given, such as the number of registered users and the level of concurrent access to be supported. A few, such as a scanner device in the document imaging domain, may not exist. Platform parameters are more diverse. Values for open parameters must be determined during the procurement process.
- They may be evident or not. The number of registered users is also an example of evident parameter. But this is not the case of concurrent access level, since this information requires to predict the future use of the mail system.
- Given parameters may be negotiable or not. Besides given, most environmental parameters are not negotiable, except for hardware actors which, as well as most platform parameters, tend to be negotiable, although in fact each particular case may be different. For instance, negotiation may be rejected in the case of recent equipment investment or in small-size companies.

7.3 Analyse quality requirements

Individual quality requirements should be analysed with respect to the complex quality model 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.

Some requirements may be easy to analyse, such as "The system shall provide Spanish interface" or "The system shall support the most common communication protocols". Others are very vague, such as "The system shall detect infected files", requiring further interaction with the stakeholders to determine more precisely what kind of virus management should be carried out. Finally, some requirements are definitively difficult to analyse. We present below one of such requirements.

Let's consider a typical requirement on mail servers such as "Message transmission time shall take less than 1 minute". Processing this requirement demands the following steps to be taken:

Step 1. *Determine which are the involved compositional system actors of the quality model*. In our case, the mail server package, the directory service and the routing tool (see fig. 4).

Step 2. *Determine which are the involved attributes of the quality models for these actors' domains*. Although there are also others, we consider here the most influential one, namely *Message Throughput* in the mail server domain.

Step 3. *Complete the definition of these attributes, if needed. Message Throughput* needs not to be further decomposed, but let's assume that the metrics is still not defined. In this case, a careful investigation is required. In fact, during the construction of the mail server quality model, this was one of the most difficult attributes to analyse due to its strong dependence on environmental and platform parameters. We analysed the information coming from a lot of (human and written) sources, including widespread benchmarks such as the Microsoft MMB2 [14]. In addition to some platform parameters we have mentioned in section 6, we discovered some environmental parameters that influence the attribute such as the expected number of concurrent accesses. Besides quality parameters, message size was also identified as relevant for message throughput.

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. Do not take yet quality parameters into account.* In our example, the previous step has pointed out the importance of message size. All the benchmarks that we looked up provide different tables for different messages sizes (among other information). For this reason, we reformulated the requirement to take this factor into account as: "Transmission time shall take less than 1 minute for messages without attachments, and less than 5 minutes per Mbyte for those with attachments". When analysing this new requirement, other part of the quality model deserved our attention, namely the existence and performance of data compression facilities for compressing attachments. We were 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 the message throughput example, this step means to test if the available benchmarks show that the requirement can be satisfied, taking into account the values of the given and non-negotiable parameters such as the number of users.

Step 6. *Analyse implications of the requirement exploring the compositional system model and the system SR diagram*. In our case the requirement has to be with

the achievement of the soft goal *Efficient Mail Handling*. From the SR diagram of the MSS, we observe that the values of the RT quality attributes that model its ability of providing an efficient routing are relevant to check the soft goal compliance.

Another point worth to remark is the application of the quality model to elucidate responsibilities of attainment. On the one hand, the environmental dependencies have been assigned to individual actors in the composite system model. On the other hand, also platform architecture has been explicitly recognised as playing an important role. As a consequence, a single requirement such as "The system shall not lose messages" would be distributed into requirements on the mail server package itself (e.g., providing appropriate queue resources), on the administrative actors (e.g., tracking and recovery facilities) and the architectural platform (e.g., mirroring).

8. Conclusions

In this paper we have proposed the use of quality models for driving software procurement in the context of complex software domains, i.e., those domains offering a mixture of functionalities. We have argued that standard quality models may fail in dealing with complex domains appropriately. We have proposed some extensions of these traditional quality models incorporating information about the environment and the platform of the system and also providing some guidelines to put together the individual domains that compose the complex one.

We have also presented a methodology called COSTUME to drive the construction of such type of quality models and we have illustrated this methodology in the complex domain of mail servers. The methodology arises from our experience in procurements we have participated in complex domains as ERP systems [14], document management, e-learning, groupware and in other domains as component libraries [15].

The construction of quality models for its use in package procurement usually pays off. Nevertheless, having into account that this process may require some substantial effort under certain circumstances, we would like to enumerate other contexts that may benefit of the existence of quality models: *system development* where quality attributes may be used to guide system development and quality assessment procedures [17]; *product quality assessment and certification*; *market exploration*, where quality attributes can help providers to know which properties would be more interesting for buyers of their new product versions; and *reference model construction,* for those organisations who base their revenues in selling product reports and white papers.

Currently we are focussing on the study of reuse of quality models. Reuse of quality models can be utterly important to facilitate the return of the investment done in their construction. One possible way is by building a taxonomy of COTS categories and domains. Category quality models could be reused in the construction of quality models of any domain belonging to the category.

Two other lines of current research are: how to apply compliance tests, that is, how would we check that a certain component meets one or more user requirements; defining of a process model for COTS procurement putting together the activities seen in section 7 with the usual ones [2, 3].

References

[1] A. Finkelstein, G. Spanoudakis, M. Ryan. "Software Package Requirements and Procurement". In *Proceedings 8th IEEE IWSSD*, 1996.

[2] J. Kontyo. "A Case Study in Applying a Systematic Method for COTS Selection". In *Proceedings 18th IEEE ICSE*, 1996.

[3] N. Maiden, C. Ncube. "Acquiring Requirements for COTS Selection". *IEEE Software* 15(2), 1998.

[4] *ISO/IEC Standard 9126-1 Software Engineering – Product Quality – Part 1: Quality Model*, 2001.

[5] R.G. Dromey. "Cornering the Chimera". *IEEE Software*, 13(1), 1996.

[6] B. Kitchenham, S.L. Pfleeger. "Software Quality: the Elusive Target". *IEEE Software*, 13(1), 1996.

[7] X. Franch, J.P. Carvallo. "A Quality-Model-Based Approach for Describing and Evaluating Software Packages". In *Proceedings 10th Joint ICRE*, 2002.

[8] X. Franch, J.P. Carvallo. "Using Quality Models in Software Package Selection". *IEEE Software*, 20(1), 2003. This is a revised version of [7].

[9] B.C. Meyers, P. Oberndorf. *Managing Software Acquisition*, Addison-Wesley, 2001.

[10] X. Franch, N. Maiden. "Modeling Component Dependencies to Inform their Selection". In *Proceedings 2nd ICCBSS*, 2003.

[11] J.P. Carvallo, X. Franch, C. Quer. "Defining a Quality Model for Mail Servers". In *Proceedings 2nd ICCBSS*, 2003.

[12] E. Yu. "Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering". In *Proceedings 3rd IEEE ISRE*, 1997.

[13] R. Buyya. *High Performance Cluster Computing*. Prentice-Hall, 1999.

[14]http://www.microsoft.com/exchange/techinfo/planning/2000/mmb2desc.asp

[15] P. Botella, X. Burgués, J.P. Carvallo, X. Franch, J.A. Pastor, C. Quer. "Towards a Quality Model for ERP System Selection". To appear as a chapter of the book *Component-Based Software Quality: Methods and Techniques*, LNCS, 2003.

[16] X. Franch, J. Marco. "A Quality Model for the Ada Standard Container Library". To appear in the proceedings of the 8th ICRST. Ada Europe'03. Toulouse. June, 2003

[17] J. Bøegh, S. Depanfilis, B. Kitchenham, A. Pasquini. "A Method for Software Quality Planning, Control, and Evaluation". *IEEE Software*, 23(2), 1999.

Appendix. Compositional System Model. This figure is not part of the paper. We include it for referee purposes.