Towards Unified Treatment of Security and Other Non-Functional Properties[*]

Elke Franz Christoph Pohl
Institute for System Architecture
TU Dresden, Germany
{ef1|cp6}@inf.tu-dresden.de

ABSTRACT

With the increasing popularity of Component-Based Software Engineering, considering non-functional properties like QoS and security becomes an ever-important challenge. Both issues are crosscutting concerns that benefit from a separated consideration as aspects. The interference of both is a non-trivial problem that has to be dealt with at various levels from specification at design time to runtime support. This paper tackles these problems by examining existing approaches for describing QoS and security properties, followed by an attempt to unify concepts from both worlds.

1. INTRODUCTION

Component-based software engineering has attracted increasing attention in the last years. Despite the fact that it is not trivial to compose working applications from different, independently developed software components, the fulfillment of non-functional requirements is of great interest for a variety of applications. Considering and enforcing non-functional properties in component-based applications is a current research field, which is closely related to Aspect-Oriented Software Development: Aspect-oriented approaches bear the potential to capture arbitrary non-functional requirements and properties as single aspects, thus allowing for better separation of concerns.

The COMQUAD project focuses on these topics. Non-functional properties include both quality of service (QoS) properties like response time or throughput, as well as security requirements, e.g., confidentiality, integrity, or availability. These properties are crosscutting concerns indeed, as they affect a wide range of system functions. Consequentially, we handle them as aspects. The specification and runtime support for these aspects is part of our component model [4].

[*]This work is part of the DFG-funded research group 428 "COMQUAD – **Com**ponents with **Qu**antitative properties and **Ad**aptivity" at Dresden University of Technology. See `http://www.comquad.org/`.

Dealing with non-functional properties requires the specification of these properties as well as their enforcement at runtime. Aagedal introduced the Component Quality Modeling Language (CQML) [1] as a component-based specification language for non-functional properties of service offers and their corresponding requirements. He states in his thesis that security properties could also be described by CQML but he concentrates on quantitative properties only. Within the COMQUAD project, this language has been extended to CQML$^+$ [8] by adding a resource clause to capture the requirements for sufficient computational resources to fulfill quantitative demands.

The description of security properties is subject of an article by Khan and Han [5]. They introduce a framework for the specification of security properties for components, considering the protection of input and output data as an example. Components are provided with a description of required and ensured security properties, which are derived directly from special security functions. Different security requirements like confidentiality and accountability may be covered by one description. An active interface compares security properties of components before they are connected. Mismatches between security descriptions are detected but can be ignored.

Multilateral security aims at enabling all parties to express and enforce their protection goals, while only minimal trust in other parties is required. Wolf and Pfitzmann [10] propose possibilities for realizing these objectives. Their description of security requirements concentrates on the negotiation between participants. Users only specify their personal weighting of protection goals, which are treated separately. Enforcement of protection goals by cryptographic functions is hard coded within the system, while users can specify their preference of concrete mechanisms by themselves. We have decided to use this approach within our architecture, because it takes care of negotiation, which is necessary to deal with potentially conflicting requirements. Security conflicts are detected, leading to the negotiation of compromises. If negotiation fails, a connection between service provider and user cannot be established. Furthermore, the approach is flexible w.r.t. specification, i.e., the means of enforcement can be changed without the necessity to alter the description of security requirements.

The approaches cited above do not consider the joint treatment of security and quantitative properties. There is not only a need for unified treatment, implying a unified specification, but also the necessity to capture the *interference* of these aspects, which is commonly referred to

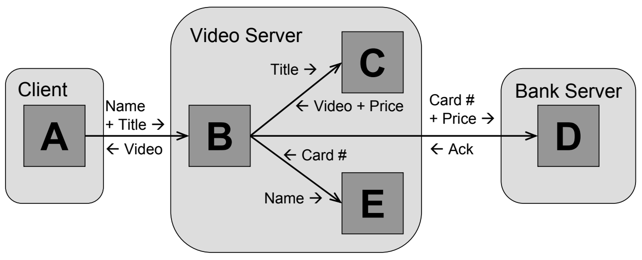

Figure 1: Distributed video on demand service scenario

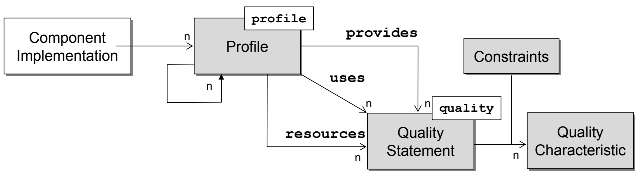

Figure 2: Simplified model of CQML$^+$

as *feature interaction* [6]. Protection goals usually imply the employment of security mechanisms, which increases resource requirements. This in turn reduces the amount of resources available for other computational tasks, thus influencing other non-functional properties like the achievable QoS.

We handle security properties as aspects, which are modelled by special system components for various security mechanisms. These components are dynamically inserted by our runtime environment into component networks for servicing client requests during contract negotiation phase. Users may weigh their preferences for certain mechanisms and security goals, thus controlling contract negotiation actively. The details of contract negotiation are subject of another publication [3].

In this paper, our first goal is to investigate possibilities for a unified specification of quantitative and security requirements. As we will show, CQML$^+$ is not expressive enough for the specification of security requirements as proposed in [10]. Furthermore, trade-offs between conflicts of both quantitative and security demands have to be considered.

The remainder of this article is organized as follows: Section 2 introduces suitable current approaches for describing QoS and security properties using a simple example. The semantic for negotiating protection goals is then extended to deal with potential conflicts with quantitative requirements in Sect. 3, followed by a discussion in Sect. 4 why security requirements still cannot simply be described by the same means as quantitative requirements. Starting points for possible solutions are presented in Sect. 5 before we finally conclude with a short outlook.

2. EXISTING APPROACHES FOR DESCRIBING NON-FUNCTIONAL PROPERTIES

Before we go deeper into the discussion of existing approaches to the issue of declarative description of non-functional properties, let us take a closer look at a small example application, which we will use to explain our concepts. Fig. 1 shows a simple video on demand (VoD) service with three involved parties: a client node, a video server hosting the actual service, and a bank server acting as a payment provider.

Five software components are acting together in this scenario: The client application (A) transmits a customer *name* and movie *title* to the booking interface (B) to receive a *video* stream. (B) requests the *title* from movie database (C) and gets a *video* stream for client (A) and the corresponding *price*. (B) also queries the customer database (E) for *name*'s credit *card♯*, which can be used to settle debts with payment provider (D).

2.1 Quality of Service and Quantitative Properties

The use of CQML$^+$ [8] for describing quantitative non-functional properties of component implementations has been anticipated introductorily. We will give a short overview of its underlying concepts and explain how it can be used to model typical QoS-related properties, such as timeliness or accuracy.

Fig. 2 depicts the basic elements of the CQML$^+$ language: *Component implementations* feature one ore more QoS `profiles`, i.e., operating modes consisting of a number of `quality` statements, which describe what qualities the implementation `provides` for its exported interfaces and what it `uses` from imported interfaces of its collaborating components. Furthermore, `resources` clauses can be used to declare the amount of needed resources. Each of these `quality` statements constrains a certain `quality_characteristic` in its value range. Assuming for the example from Fig. 1 that the `quality` statements `low_delay`, `fast_response`, and `fast_network` have already been defined before for a set corresponding set of `quality_characteristics`, we could write for instance:

```
profile videoBinding for B {
  provides low_delay (B.booking.getVideo);
  uses fast_response (C.outvideo.getVideo);
  resources fast_network (network);
}
```

This simply means a `videoBinding` can be provided for component B, which provides a low delay for the `getVideo` operation of the booking interface port `booking` if the runtime environment guarantees fast response times for component C's `getVideo` operation at port `outvideo` and if a fast `network` resource is available. Of course, the example is oversimplified and incomplete for the sake of clarity.

Quality statements and characteristics are usually packaged in CQML$^+$ libraries categorized in domains for which the corresponding semantic is well agreed upon. An in-depth discussion of all concepts and features of CQML$^+$ [8] would go beyond the scope of this paper. However, our short example should have been detailed enough to demonstrate the "look and feel" of describing non-functional properties with this kind modeling languages.

2.2 Security Requirements

Security requirements of users are usually described by protection goals. There is a variety of possible protection goals which can be classified into confidentiality, integrity, and availability goals. Depending on the object of protection, we can describe more specific goals like anonymity as confidentiality of communication circumstances.

An important difference to quantitative properties is the fact that there may be conflicting security interests, especially in case of protection goals with individual perspectives of involved stakeholders [10]. An example is the accountability of messages: One of the communication partners agrees to be accountable. Depending on the scenario, one of the participants gets an advantage over the other one. In contrast, fulfilling quantitative requirements mainly depends on the availability of computational resources of the executing machine; involved parties usually take no critical advantages over others. Because we do not only consider protection goals but also quantitative demands, security requirements of a user may be additionally influenced by conditions of his system, e.g., available CPU time, memory etc. A system providing multilateral security has to consider conflicting interests. Because the approach in [10] explicitly aims at providing multilateral security, we investigate possibilities to apply it in our scenario.

As an example, let us discuss confidentiality of communication in the video server scenario (Fig. 1) for the use case "get video stream". Actually, this is a protection goal about which the participants share a common perspective: They want to protect the content of their communication against possible eavesdroppers in the unsecure communication network. The participants may have different weightings for this protection goal. The server is strongly interested in protecting the content of the video stream, because he wants to prevent attackers to get the video data without payment. The user is less interested in encrypting the video stream, moreover, required computing costs may be undesirable to him. But he would agree to use encryption if the video server insists.

To reduce potential conflicts during negotiation, [10] introduces a five level gradation (`unconditional`, `if_possible`, `dont_care`, `if_necessary`, `on_no_condition`) to express protection demands. The negotiation results in a "yes" or "no" decision whether a protection goal is to be enforced. Only the combination of the weightings `unconditional` and `on_no_condition` will produce a conflict that requires interaction. If both participants use `dont_care`, the system can automatically decide to enforce the protection goal by default.

In the example described above, the user weighs the protection goal as follows:

```
communication_confidentiality = if_necessary
```

The provider of the video server, however, uses the weighting

```
communication_confidentiality = unconditional
```

Negotiation between user and video server results in the decision "yes". The use of a concrete encryption mechanism is determined by a comparison of the preference lists of user and video server. Mechanisms are not directly specified along with protection goals in order to increase flexibility of choice.

3. ADJUSTING SECURITY NEGOTIATION FOR COMPONENTS

The description of security properties introduced above considers only use cases of single applications. In component-oriented architecture however, there are no monolithic applications. Different components are involved in processing use cases. It would be desirable to associate security requirements of use cases to the components they are assigned to, because this would allow the migration of components between hosts without the necessity to change security descriptions.

Because it may become impossible to fulfill both quantitative and security requirements, we need a strategy in case of conflicts. In case of insufficient resources, either quantitative or security demands have to be reduced. A first step in this direction is to extend the semantics of the weighting described above. The extension is based on the assumption that participants willing to reduce their security requirements in case of conflicts with requirements of others would also be willing to reduce them in case of insufficient computational power. The extended semantic, according to [7], is as follows:

`unconditional`: The protection goal has to be enforced.

`if_possible`: The protection goal is enforced unless the communication partner wants to prevent this by all means (`on_no_condition`) or in case of insufficient resources even after reducing quantitative requirements.

`dont_care`: The protection goal is enforced according to the communication partner's preferences. If resources are insufficient, security demands are reduced first.

`if_necessary`: The protection goal is enforced only if the communication partner requires it *and* if there is enough computational power.

`on_no_condition`: The protection goal must not to be enforced.

Due to the fact that further negotiations may become necessary if components migrate or have to use services of other components, the result of negotiation should reflect the component's original weighting. Table 1 describes the extended negotiation matrix, which allows the use of negotiation results for further negotiation:

Table 1: Extended Negotiation Matrix

requirements	on_no_condition	if_necessary	dont_care	if_possible	unconditional
on_no_condition	on_no_condition	on_no_condition	on_no_condition	on_no_condition	*conflict*
if_necessary	on_no_condition	if_necessary	if_necessary	if_possible	unconditional
dont_care	on_no_condition	if_necessary	dont_care	if_possible	unconditional
if_possible	on_no_condition	if_possible	if_possible	if_possible	unconditional
unconditional	*conflict*	unconditional	unconditional	unconditional	unconditional

Despite these adjustments, describing security requirements in CQML$^+$ raises some new problems, which are pointed out in the next section.

4. PROBLEMS OF EXISTING APPROACHES

It would be desirable to use existing techniques for quantitative properties to describe security properties in order to simplify unified treatment.

However, there are some problems if we try to describe security requirements with CQML$^+$. A closer look at descriptions of QoS-related and security properties reveals some important differences. First of all, the approaches for quantitative properties inherently describe the semantic of these properties. If the user demands `low_delay` for the video stream, this can be expressed directly by response times on a lower abstraction level. The requirements on other components (`fast_response`) as well as the requirements on the resources (`fast_network`) can be directly derived. In contrast, descriptions of security requirements considered here capture the weighting of users but not the semantic of security properties themselves.

Furthermore, QoS-related properties are in principal handled as supply and demand. The necessity for negotiation is obvious for security properties but also necessary for QoS-related properties. Especially in case of conflicting requirements it is necessary to define the preference of requirements in order to make a decision.

We have to decide at which level security requirements shall be described. There are different meta-levels of components [2]: specification, implementation, installation, and instantiation. As described in Sect. 2.1, QoS-related properties are usually assigned to implementations. Profiles describe different operating modes that can be chosen depending on demands. However, security requirements should not explicitly depend on the operating mode of a component. Therefore, we suggest to assign security requirements to the specification of a component.

Further security demands of users or others, e.g., the server providers themselves, may result in requirements on the container itself or on all components equally [9]. So it is likely that a system provider wants to execute only "original" components, which are not modified by others than the component providers. Components have to be signed by their providers to allow to test their integrity and accountability upon initialization. CQML$^+$ does not directly consider such global requirements. Therefore, they need to be forwarded by `uses/provides` clauses. But these are in fact rather inappropriate for this purpose. It would be more desirable to let the container make global demands.

If we try to use CQML$^+$ to describe security requirements, we also have to solve the question of which clauses to use for expressing requirements. We could either use `uses/provides` or `resources` clauses. Specifying security as `resources` is generally unsuitable. A resource is demanded for and can be provided by the system or not. It would be semantically curious to specify that a resource must not be used.

Therefore, we had a look at `uses/provides` clauses. However, these are dedicated for specifying supply and demand. To enable negotiation, we would have to introduce cyclic specifications. For an interface, each component has to specify its demands, but it also has to receive demands of other components it wants to use.

Resulting problems are pointed out in [7] and shall be summarized in the following. Let us assume that the component `B` of the example in section 2 provides function `getVideo` as part of `BookingInterface` exposed as port `booking`:

```
interface BookingInterface {
  VideoStream getVideo(String title, String name);
}

component A{
  uses BookingInterface myPort;
}

component B{
  provides BookingInterface booking;
}
```

As stated in section 2.2, the user (component A) wants confidentiality only if necessary, while the video server (component B) unconditionally demands confidentiality. To enable a successful negotiation, the interval of the `provides` clause has to be part of the `uses` clause's interval of acceptable values [1].Therefore, [7] introduces a number of `quality` statements to express proper `provides` clauses for possible security requirements. For our example, we need a suitable `provides` description for `if_necessary` and `unconditional`:

```
// for if_necessary:
quality confidentiality_all (op:operation)
{
  confidentiality(op) = on_no_condition or
  confidentiality(op) = if_necessary or
  confidentiality(op) = dont_care or
  confidentiality(op) = if_possible or
  confidentiality(op) = unconditional;
}

// for unconditional:
quality confidentiality_all_but_on_no_condition
(op:operation)
{
  confidentiality(op) = if_necessary or
  confidentiality(op) = dont_care or
  confidentiality(op) = if_possible or
  confidentiality(op) = unconditional;
}
```

Using these specifications, we can express components' requirements as follows:

```
profile confidentiality_if_necessary for A {
  uses confidentiality_if_necessary
    (myPort.getVideo);
  provides confidentiality_all
    (myPort.getVideo);
}

profile confidentiality_unconditional for B {
  uses confidentiality_unconditional
    (booking.getVideo);
  provides confidentiality_all_but_on_no_condition
    (booking.getVideo);
}
```

However, the semantic of `uses/provides` in CQML$^+$ cannot be met. Actually, for QoS-related properties the "negotiation" may restrict the `uses` interval of acceptable values.

That is the reason why the `provides` interval must be part of the `uses` interval as cited above, e.g., [10..12] would be a permissible restriction of [10..15] but [9..11] would not. A successful negotiation w.r.t. QoS-related properties would yield exactly the value specified in the `uses` clause regardless of any restrictions, e.g., simply the value 10 out of the interval [10..12] above. This is not problematic at all because the `uses` clause just means that the using component *is able to process all possible input data of the specified domain.* The fact that the used component does not accept all possible values of this interval has no effect on processing.

In case of negotiation, however, each weighting specifies a certain negotiation strategy. Therefore, valid results of a negotiation have to be a acceptable weightings for both components—as one can conclude, it has to be the stronger weighting. The negotiation specified in the `uses` clause is only the result of negotiation when two components with identical demands are connected. So negotiation in the example above results in `unconditional`. Even if both `if_necessary` and `unconditional` would lead to enforcing the protection goal, we have to prevent the stronger weighting for further negotiations. Therefore, instead of `if_necessary` as specified in the `uses` clause, negotiation has to yield `unconditional`.

5. PROPOSED SOLUTIONS

First of all, we need two stages for unified treatment of security and other non-functional properties: (*i*) negotiation of concurring requirements and (*ii*) enforcement of requirements.

At the first stage, there has to be a negotiation between different requirements. We have to specify strategies for conflicting situations as already sketched for the extended semantic of security requirements. However, we will need more comprehensive language features if we want to describe further preferences. To give an example: Users also want to specify which of the security requirements may be reduced first, e.g., whether confidentiality is more important for them than accountability. We have not yet considered trade-offs between conflicting quantitative demands like performance and quality of the data. Until now, we have not yet been able to implement a convenient way to negotiate quantitative properties between different nodes as well.

For a real negotiation we need a new semantic. As we have seen in the example above, it is inappropriate to just allow reducing the `provides` domain for a real negotiation. This is only sufficient for supply and demand. Instead, we need a language feature that allows results of negotiations to differ from values specified by `uses` clauses. The resulting value must not conflict with requirements of participants. The existing `uses/provides` clauses are inappropriate for this purpose. At the moment, we use an XML-based description format to specify the weighting for negotiation. The following code snippet shows the server provider's security demands for the booking component `B`'s `booking` port as an example:

```
<negotiation_settings>
  <preference>
    confidentiality, integrity,
    accountability, quality
  </preference>
  <demand
    component_spec="B"
    interface="booking"
    confidentiality="unconditional"
    integrity="if_necessary"
    accountability="if_possible"
    preference="confidentiality,
      accountability, delay"/>
</negotiation_settings>
```

Potentially conflicting requirements are listed as single items within the `demand` element, together with their weighting. An explicit `preference` list can describe the ordering in which single requirements may be reduced if they are not weighted as unconditional. These preferences can be declared globally for all demands or overridden per demand. We currently align the XML-binding of CQML$^+$ for better interoperability with the negotation settings format.

The components' runtime environment has to provide a set of rules to support comprehensive negotiation. Careful study of interferences between single properties is necessary for this. First of all, a plausibility test for requirements of single components is necessary. Second, the runtime environment has to negotiate the resulting requirements on the application. Participants have to be informed in case of conflicts.

At the second stage, the runtime environment tries to enforce the requirements. Now the system tries to connect components by processing CQML$^+$ statements. The strategy of restricting requirements is predetermined by the result of the first stage. The result of the second stage is sent to the other participants. Because they have negotiated a valid strategy for reducing, each result of the second stage is acceptable for them. Therefore, both use the result with less demands for computational power, i.e., the configuration both can fulfill. The current prototype of our model [4, 3] performs negotiation by iteratively aligning conflicting demands, but we will also try to find an analytical solution to this optimization problem.

To provide multilateral security, we have to enable the participants to set their requirements. That means that the descriptor which defines the negotiation settings is not written only by the component provider. The server provider as well has to be enabled to configure his preferences and, if necessary, to overwrite requirements suggested by the component provider.

6. SUMMARY AND OUTLOOK

In this paper, we have discussed possible steps towards a unified treatment of security and other non-functional properties. We have mainly considered the issues of negotiating between potentially conflicting requirements and how to raise the semantic of existing approaches to the level of components. Our current solution uses different languages to describe negotiation preferences and quantitative properties. Even if these different languages are used at different levels, we need clear interfaces between these levels. Therefore, a unified specification language is desirable. There are two possible ways: A new specification language covering both negotiation and description of quantitative properties on one hand. On the other hand, the existing CQML$^+$ can be extended to allow to control the negotiation as sketched in Sect. 5. To reuse existing solutions as far as possible, we are currently investigating the latter alternative. A unification of concepts and descriptions would however provide a cleaner solution.

Existing CQML$^+$ requires consideration of further demands, e.g., a suitable description for the semantic and the enforcement of security properties would be needed. Additional means for describing global (security) requirements would also become necessary in contrast to the current practice of attaching all requirements to components.

Summarizing the problems pointed out with existing languages, the model has to be extended to support all component meta-levels in a unified manner and global requirements on the runtime environment.

7. REFERENCES

[1] J. Ø. Aagedal. *Quality of Service Support in Development of Distributed Systems*. PhD thesis, University of Oslo, 9 Mar. 2001.

[2] J. Cheesman and J. Daniels. *UML Components: A Simple Process for Specifying Component-Based Software*. Addison Wesley Longman, Inc., 2001.

[3] S. Göbel, C. Pohl, R. Aigner, M. Pohlack, S. Röttger, and S. Zschaler. The COMQUAD container architecture. Submitted for publication., Feb. 2004.

[4] S. Göbel, C. Pohl, S. Röttger, and S. Zschaler. The COMQUAD component model – enabling dynamic selection of implementations by weaving non-functional aspects. In *International Conference on Aspect-Oriented Software Development (AOSD'04)*, Lancaster, UK, 22–26 Mar. 2004. ACM. To appear.

[5] K. M. Khan and J. Han. Composing security-aware software. *IEEE Software*, 19(1):34–41, 2002.

[6] E. Pulvermüller, A. Speck, M. D'Hondt, W. DeMeuter, and J. O. Coplien. Feature interaction in composed systems, ECOOP 2001 Workshop Proceedings. Technical Report 2001-14, Universität Karlsruhe, Sep 2001.

[7] M. Reiche. Unterstützung von mehrseitiger Sicherheit und Dienstgüte in komponentenbasierten Systemen. Diplomarbeit, Dresden University of Technology, Sept. 2003. In German. English title: Support for multilateral security and QoS in component-based systems.

[8] S. Röttger and S. Zschaler. CQML$^+$: Enhancements to CQML. In J.-M. Bruel, editor, *1st Intl. Workshop on Quality of Service in Component-Based Software Engineering*, pages 43–56, Toulouse, France, June 2003. Cépaduès-Éditions.

[9] U. Wappler. Sicherung der Integrität von Softwarekomponenten. Großer Beleg (term paper), Dresden University of Technology, 15 Oct. 2003. In German. English title: Securing integrity of software components.

[10] G. Wolf and A. Pfitzmann. Empowering users to set their protection goals. In K. Rannenberg and G. Müller, editors, *Multilateral Security in Communications – Technology, Infrastructure, Economy*, volume 3 of *Informationssicherheit*, pages 113–135. Addison-Wesley, München, July 1999.