Specifying Security Requirements and Assurances of Software Components

Khaled Md. Khan[1] Jun Han[2] Yuliang Zheng[2]

[1]School of Computing and Information Technology
University of Western Sydney
PO Box 10 Kingswood, NSW 2747
Australia
e-mail: k.khan@uws.edu.au

[2]School of Network Computing
Monash University
McMahons road
Frankston, Vic 3199 Australia
e-mail: {jhan,yuliang}@pscit.monash.edu.au

Abstract

In a distributed component based system, it is important to model and specify the QoS (quality of services) requirements of software components. In this paper we propose a security specification structure that can be used to model and specify required and ensured security properties of software components. The specification of security requirements can be an aid to software engineers on two main fronts: (i) to specify security requirements during the development of software components, (ii) with support for introspection, a candidate component can be queried about its required and ensured security properties by other software components or human composers in distributed environments. Our proposed specification structure is intended to capture and model pre and post execution security behaviour of software components. The paper provides a preliminary modelling of security function primitives of software components.

Keywords.

Software component, software compositional relationship, ensured security properties, required security properties.

1. Introduction

Technologies such as CORBA, HTML provide connection or composable glue code for constructing application software from independent components such as Beans from Sun's Enterprise JavaBeans, objects from OMG CORBA, Microsoft's DCOM and ActivX [4]. When software components developed by third parties are composed and distributed over open public communication channels, there are greater opportunity for security threats to the entire enclosed application. Security is one of the vital non-functional quality factors expected to be provided by software components, and lack of this cannot be a favourable contributing factor to the success of a viable component market. Although non-functional quality factors are considered important, main focus in component technology has been placed on achieving component functionality. The requirements of QoS are not often well specified during the development of components. The code and algorithms implementing software quality properties are usually strongly interwoven with the codes that implement component functionality and business logic [5].
In a component-based system, each component has its own security properties, and these need to be adequately specified in such a way that other components as well as human users should be able to 'read' those features, and reason their impact on the combined system. The ability of a component to reason the security impact of other components is far more needed and important. The fundamental question is how a component decides if a client component is secure enough to execute. Similarly, how a client component knows what would be the ultimate security impact on an application system if it assembles with a 'foreign' component. A component must be very clear about what the security contract across the connection with other component is, what security provisions each component requires and ensures, and what would be the ultimate security of the composed system [7]. The need for a security specification structure along with the interface signature is acute as expressed concerns in [9], [10], [11], [12].
In this paper we have proposed a simple security specification structure to model the security function of individual software components. The structure can specify the nature of security considerations of a component. It is based on four fundamental compositional elements: identities of contracting components, operations to be performed in a compositional relationship, security attributes required and ensured by a component, and the resources to be used by other components. Our structure is partially based on the functional requirements defined in the Common Criteria (CC) for Information Technology Security Evaluation, version 2.0 [1]. Common Criteria for Information Technology Security Evaluation, version 2.0 (CC) provide a schema for evaluating IT system, and enumerate the specific security requirements for such systems. The functional requirements in CC describe the security behaviour or functions expected of an IT system to counter threats. These requirements consist of eleven classes. In this paper we use one particular class called user data protection.
The rest of the paper is structured as follows. Section 2 presents four major elements required for compositional relationships (CR). In section 3, a structure for specifying security requirements is proposed. The security requirements specifications of individual component along with examples are proposed in section 4. We outline our further work and concluding remarks in section 5.

2. Properties of compositional relationships

A compositional contract between any two given components is governed by one or more *compositional relationship* [3]. A CR can be determined by the intentions and purposes of the contracting components. To make an intended CR, a client component makes a request to a server component specifying its intended operations such as execute message, receive message, send message, create object, and destroy object. The server may grant the intended operations depending on the nature of requested relationships and the security attributes of the client. The establishment of a compositional relationship is based on the following four fundamental elements:

i. Identities
ii. Operations
iii. Security attributes, and
iv. Resources (data, information).

2.1 Identities

In any security specification, the identities of the participating components in a CR need to be established. In general, two types of entities are involved in a composition: *server* components, and *client* components. However, a server component may also make a request, in such case the component plays the role of a client. A *client* is an entity which causes an *operation* to be performed on *resources* that belong to a *server* component. The terms 'client' and 'server' are used to model the role that a component plays in a particular use scenario.

2.2 Operations

A compositional relationship is based on the type and nature of operations performed by the interacting components. An *operation* may be performed on a *server* component by *a client* component [2]. It may include reading data from a file owned by the server, writing data on a file or entities that belong to the server, sending messages, receiving messages, connecting request, creating an object, destroying an object, and so on. The server component may check its security policy (SFP) to ensure that an operation in a CR is permitted to the specified client given that the *required security attributes* are provided.

2.3 Security attributes

Security attributes are used to authenticate components, authorise operations, protect data, and so on. Examples of such *security attributes* can be passwords, private keys, secret keys, public keys, shared keys, digital signatures among others. These can be classified into two different types: *ensured security attributes* and *required security attributes.* A *server* component may have some *ensured* security attributes as well as *required* security attributes for a compositional contract [12]. A *required* security attribute is an invariant in a sense that it is the required property of a component that other interested parties must satisfy during the composition according to the contract [8]. A *required* security attribute governs the relationship between two contracting components. It is a precondition the component must ensure that the proper security attribute is provided, and its validity is ensured. Similarly, an *ensured* security attribute is a postcondition in a sense that it is the responsibility of the component to maintain the committed security *assurances* during the composition of the contract. The relationship between a client and a server that incorporates quality of service (QoS) properties is known as *QoS contract* [6].

2.4 Resources

A security QoS contract depends on the type of resources (data, information) to be affected by the requested *operations*. A server component may specify different access constraints for different client components for the same operations on its resources. In a runtime composition, the server component may enforce specific controls over access to its various objects and methods.

3. Structure for security requirements specification

Based on these four compositional elements we can formulate a structure that can capture the security requirements and assurances of software components. The security properties of individual component can be specified with a predicate-like structure as

security_function({entities},{operations}, {security_attributes}, {resources})

where, *entities* can be a set of active entities such as client or server components, or both. *Operations* are a set of actions the *entities* may perform such as read, write, create, destroy, print, so on. S*ecurity* *attributes* can be passwords, private keys, public keys, secret keys, and so on. Each of the arguments contained in the structure can be further decomposed as

entities=(entity_ID, entity_URL, entity_developer's_ID)
operations=(operation_name, operation_duration, operation_frequency)
security_attributes =(key_standard, key_length, algorithm_used)
resources=(resource_value, resource_type, resource_range).

To model simple security functions we may not need the detail decomposition of the structure. Figure 1 shows a decomposed hierarchical structure of the security specification model. Figure 1 shows two levels of hierarchy of the structure. Each argument in level 1 has a corresponding low level structure.

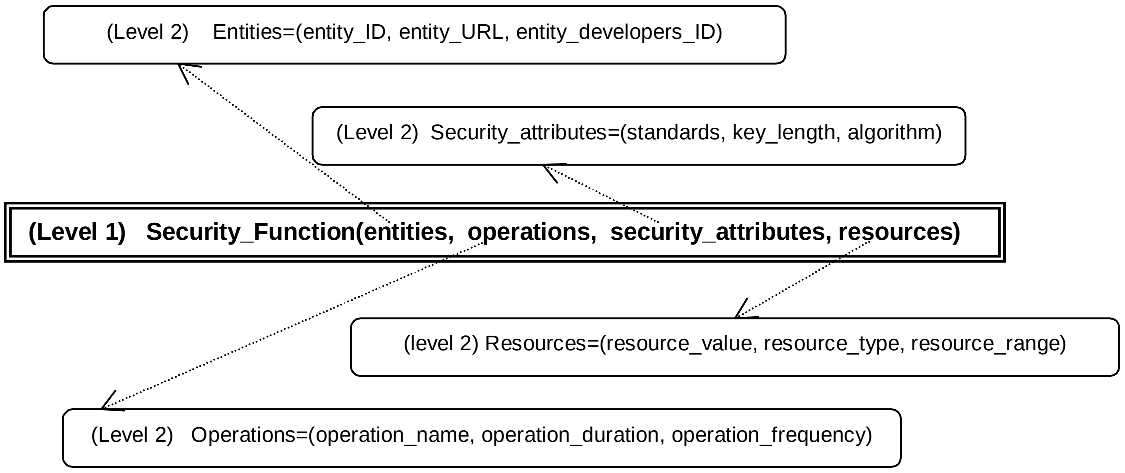

(Level 2) Entities=(entity_ID, entity_URL, entity_developers_ID)
(Level 2) Security_attributes=(standards, key_length, algorithm)
(Level 1) Security_Function(entities, operations, security_attributes, resources)
(level 2) Resources=(resource_value, resource_type, resource_range)
(Level 2) Operations=(operation_name, operation_duration, operation_frequency)

Figure 1. Hierarchical decomposition of the security specification structure

4. Security specification of components

In this section, we describe a simple distributed computation system for banking transactions, and use this as an example in our specification structure. Individual business application system of account holders can dynamically assemble with a banking system component over the open communication network. The identity of the banking component is "S", a server component. It receives requests for account balance reports from its users' systems. The component finds the account balance for the valid request, and transmits it to the requestor. Most of the users' application systems are located in various remote machines.

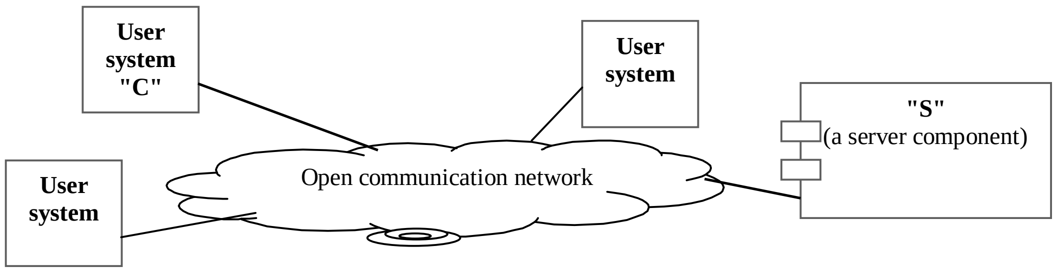


Figure 2. A scenario of distributed banking specification

When a user generates a connection request for a compositional relationship to the component from their application system (the client system), the component "S" then identifies itself with its interface signatures. If both parties agree to establish a compositional relationship, the user system (the client) transmits its public key, if there is any, to receive secure data from "S". Component "S" also broadcasts its public key to the user (the client) for secure transmission of data to "S" provided that

the user's identity is authenticated. To make the data transmission confidential, all messages are encrypted with the public key of the recipient system, and messages can be decrypted with the private key of the recipient system. The user system must fulfil the *required security* property of the component to get the *ensured security* of the messages from the component "S". We assume the identity of a user system is "C". This scenario is illustrated in Figure 2.
The structure proposed in the previous section provides a simple approach for specifying the security requirements of individual components on which compositional relationship among components can be established. The security specification structure will be applied to the example just described to model and illustrate some security functions defined in CC [1]. It is important to note that a security specification is defined to withstand one or more specific threats. In the reminder of this section we will apply our structure to model various required and ensured security properties provided by the user system and the banking component as outlined in the system description. Each specification structure modelling a security function (SF) is presented along with the corresponding security function policy (SFP) of the SF and an instance of the structure.

Spec 1: Access request
We can specify a security interface requirement of a component as
    CR_request(entities, operations, security_attributes, resources)
Before any security QoS contract is made between an identified client component and a server component, the authorisation must be granted by the server component. The prior authorisation request for such a CR requirement can be modelled in a server component as
SF: **CR_request_from(client_ID, requested_operations, security_attribute, resources)**
Spec 1.(a)
SFP: A server component receives a connection request from a client. This can be considered as a required property of the server component.
An instance of this specification can be,
    CR_request_from("C", "get_account_balance", "NULL", "NULL")
Note that, in this example no security attributes or resources have been specified as those were not required by the server component at this stage.
On the other hand, the corresponding request generated by a client component can be specified as
SF: **CR_request_to(server_ID, requested_operations, security_atributes, resources)**
Spec 1 (b)
SFP: A client component makes a request for an operation to be performed on the server component. This is an ensured property of the component.
An instance of this specification can be,
    CR_request_to("S", "get_account_balance", "NULL", "NULL")
In this example it shows that the client does not provide any security attributes or resource names because those properties are not required by the server at this stage. Figure 3 shows both specifications defined in Spec 1.

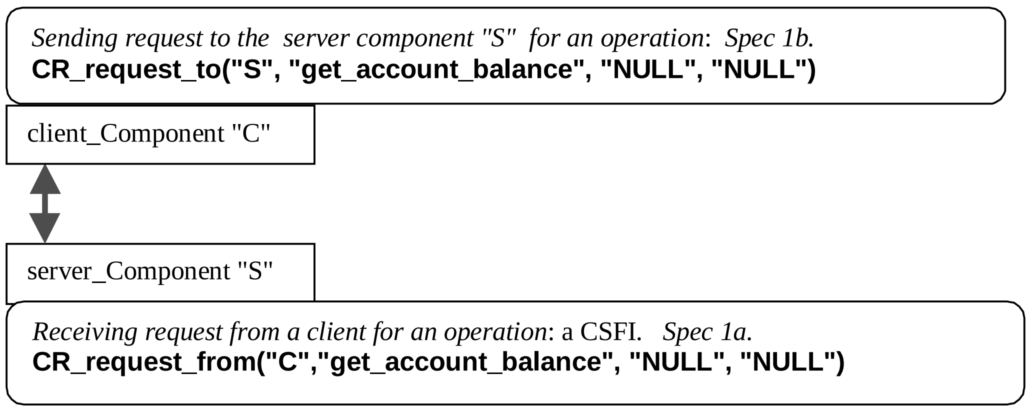

Sending request to the server component "S" for an operation: *Spec 1b.*
CR_request_to("S", "get_account_balance", "NULL", "NULL")
client_Component "C"
server_Component "S"
Receiving request from a client for an operation: a CSFI. *Spec 1a.*
CR_request_from("C","get_account_balance", "NULL", "NULL")

Figure 3. Instances of Spec 1(a) and Spec 1(b)

Spec 2: Access authorisation

In this specification, the security property *required* by a server component can be modelled as

SF: **authorise_access(client_Id, requested_operations, security_attribute, resources)**

Spec 2(a)

SFP: The predicate Spec 2(a) authorise_access specifies a client which makes a request to perform an operation on the server. requested_operations can be one or a set of operations as specified to be performed on the server component. security_attribute can be an authentication data of the client such as a password or a key.

An instance of Spec_2(a) specification can be

authorise_access("C", "get_account_balance", $(\text{password})_C$, "NULL")

The corresponding security attribute *ensured* by a client component can be characterised as

SF: **access_request(server_ID, requested_operations, security_attribute, resources)**

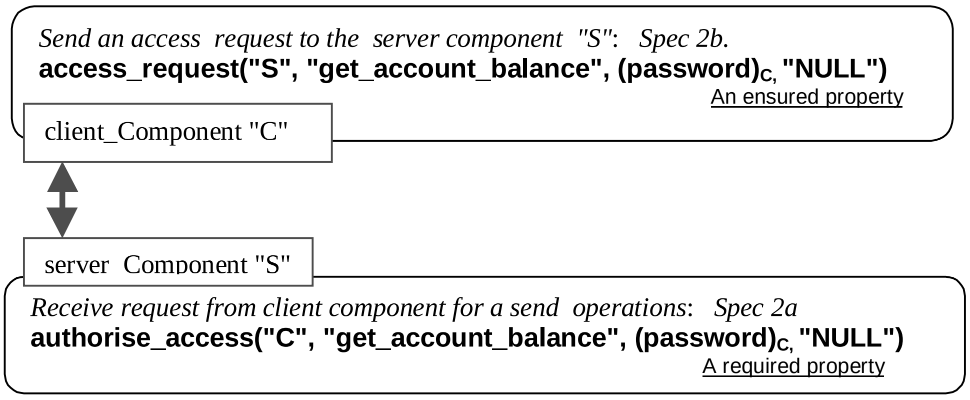
Spec 2 (b)

SFP: Predicate 2(b) specifies that a client intends to do one or more operations on a server component using the specified security attribute.

An instance of the Spec_2(b) can be,

access_request("S", "get_account_balance", $(\text{password})_C$, "NULL")

A client component send a message to the server, and *ensures* the security attribute key or password as *required* by the server component. The instances of Spec_2 (a) and Spec_2(b) are shown in Figure 4.



Send an access request to the server component "S": Spec 2b.
access_request("S", "get_account_balance", (password)C, "NULL")
An ensured property
client_Component "C"
server Component "S"
Receive request from client component for a send operations: Spec 2a
authorise_access("C", "get_account_balance", (password)C, "NULL")
A required property

Figure 4. Instances of Spec_2 (a) and Spec_2 (b)

Spec 3.: Data authentication

A server or a client component may have a security function to check the integrity of an input data, and this can be specified as

SF: **validate_in_data(sender_ID, operations, security_attribute, in_data)** Spec_3

SFP: validate_in_data predicate specifies an evidence or guarantee of authenticity of the data contents in_data in terms of associated security_attribute. security_attribute can be a key used to decrypt the received incoming data. Examples of such attributes are private keys, secret keys, shared keys, and so on.

An instance of this property could be

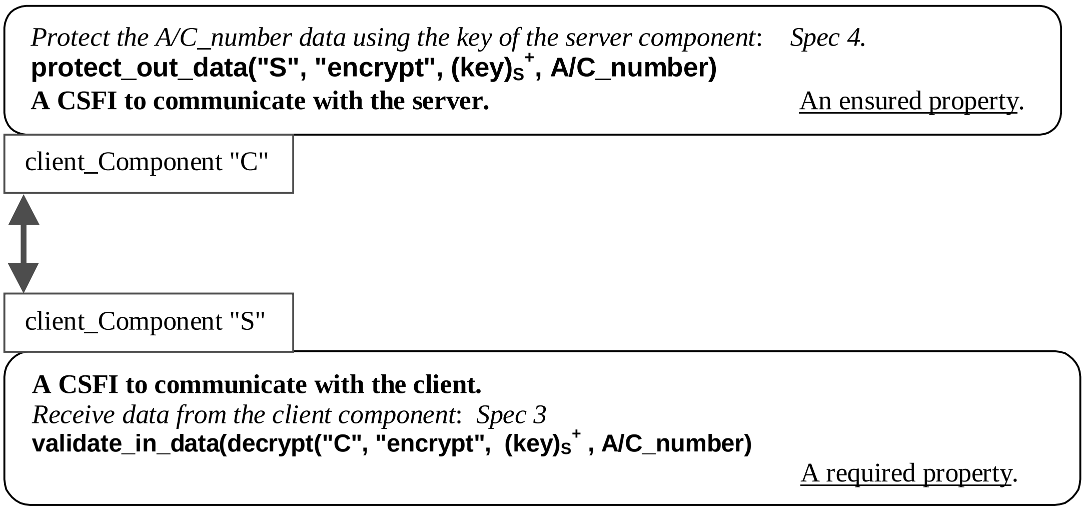
validate_in_data("C", "encrypt", $(\text{key})_S^+$, A/C_number)

In this example, the $(\text{key})_S^+$) is the public key of the server component that receives data from "C" as data is encrypted with its ("S") public key. It implies that only the recipient component "S" can decrypt the message A/C_number using its private key to get the actual account number of the client.

Spec 4.: Export of data to outside control of components
A server or client component may have a security function to encrypt an outgoing data with proper security attribute, and that can be specified as
SF: **protect_out_data(receiver_ID, operations, security_attribute, out_data)** Spec_4
SFP: The protect_out_data predicate specifies the type of security_attribute used to encrypt the outgoing data produced by the sender component. Example of security_attribute can be public key, shared key and so on.
An instance of this can be,

protect_out_data("S", "encrypt", (key)$_S$+, A/C_number)

In this example, the (key)$_S$+) is the public key of the component "S" to whom this data is to be sent. It shows that the "A/C_number" sent by the client "C" is encrypted with the public key of the recipient component "S".
However, the out_data may not be always encrypted, in such case, the values of the operation and security_attribute remain NULL.



Protect the A/C_number data using the key of the server component: Spec 4.
protect_out_data("S", "encrypt", (key)$_S^+$, A/C_number)
A CSFI to communicate with the server.
An ensured property.
client_Component "C"
client_Component "S"
A CSFI to communicate with the client.
Receive data from the client component: Spec 3
validate_in_data(decrypt("C", "encrypt", (key)$_S^+$, A/C_number)
A required property.

Figure 5. Instances of Spec_3 and Spec_4

Figure 5 shows instances of Spec_3 and Spec_4. According to this specification, client component can send a bank account number, A/C_number, protected by the public key of the server component. And the server component receives input data A/C_number, and can extract the actual data by using its own private key from the client component.

Spec 5: Information flow control

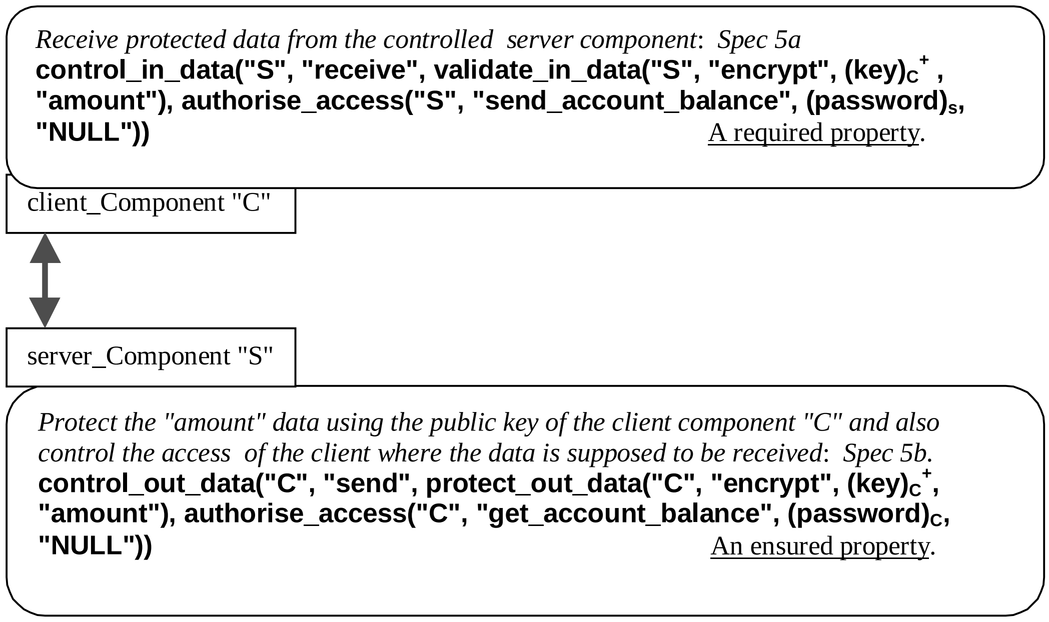
SF: **control_in_data(entity_ID, operations, validate_in_data(sender_ID, operations, security_attribute, in_data), authorise_access(client_ID, operations, security_attribute, resources)**

Spec_5 (a)

SFP: Spec_5a specifies the controlled or uncontrolled in_data flow from anywhere to the controlled component. The controlled component can be a server or a client component where this specification is defined. The nested predicate Spec_3 is used in this specification. It describes that before a component receives a data from another component, it verifies whether the sender component has the prior authorisation to establish a compositional relationship with it or not.

SF: **control_out_data(entity_ID, operations, protect_out_data(receiver_ID, operations, security_attribute, out_data), authorise_access(client_ID, operations, security_attribute, resources))** Spec_5 (b)

SFP: Spec_5b specifies the controlled or uncontrolled out_data flow from a controlled server to a controlled or uncontrolled client. In two nested predicates in this specification, the first one specifies the nature of the security protection of the out_data as defined in Spec_4. The second nested predicate authorise_access specifies whether the receiving client is controlled or uncontrolled according to the Spec_1. It describes that before a component sends a data to another component, it verifies whether it has the prior authorisation to establish a compositional relationship with that component or not.
These express how a controlled server component puts limitations on importation of data, controls on the sharing and interportation of the security attribute between a client and a server, e.g., identity information, security data associated with data and so on. Figure 6 shows instances of these two specifications.


Receive protected data from the controlled server component: *Spec 5a*
control_in_data("S", "receive", validate_in_data("S", "encrypt", $(key)_C^+$, "amount"), authorise_access("S", "send_account_balance", $(password)_S$, "NULL")) A required property.

client_Component "C"

server_Component "S"

Protect the "amount" data using the public key of the client component "C" and also control the access of the client where the data is supposed to be received: *Spec 5b.*
control_out_data("C", "send", protect_out_data("C", "encrypt", $(key)_C^+$, "amount"), authorise_access("C", "get_account_balance", $(password)_C$, "NULL")) An ensured property.

Figure 6. Instances of Spec_5 (a) and Spec_5 (b)

All specifications presented here demonstrate that each component may have distinct security properties, and each may be administered and maintained independently. In this work we have not included other security aspects of compositions such as non-deducibility, non-leakage of information, and non-interference. For simplicity, we have only considered the access control and validity of data in our work in this paper.

5. Further work and conclusion

We have proposed a specification structure to capture and specify security properties of independent software components. The structure is based on four fundamental compositional elements. Each of the elements can be further decomposed to specify more security related information of a component. This model can expose the nature of security considerations of a component to other components. We have shown with a simple example how a component can have a specified ensured security properties corresponding to required security properties of another components.

Our security specification primitives can be the part of component introspection in such a way that a component should know its required and ensured security properties, and can communicate this knowledge to other interested components. With support for introspection, a component can be queried about its security properties (ensured, and required). The availability of such meta-information along with the component interface signature may facilitate the prior understanding of the security nature of components and their possible compositional effect in dynamic run-time discovery and execution. The proposed structure can also be applied to non-distributed software system to model the security requirements.
The major limitation of the work presented in this paper is that we have not defined how a compositional security specification would be built based on these security specifications. We are currently working in that direction to formalise some binary compositional rules. Our current work is intended to combine the security specification of one component with that of another component in order to model a compositional security specification. The compositional specification should define rules about the impact of combining two primitive security specifications in a given time. Based on a binary relationship between two specifications we may be able to formulate a more complex multiple relationships among a group of components. We are currently working to formalise our approach for a complete model of security requirements specification. The formal specification will be tested in a real component based system.

Acknowledgments

Authors would like to thank the anonymous referees for their useful comments and suggestions.

References

[1] ISO/IEC-15408. (1999). Common Criteria project. Common Criteria for Information Technology Security Evaluation, Version 2.0.NIST, USA. http://csrc.nist.gov/cc/, June 1999

[2] Khan, K., Han, J., Zheng, Y.,"Characterising User Data Protection of Software Components", Proceedings ASWEC'2000, IEEE Computer Society press, Canberra, April 28-29, 2000, pp. 3-12.

[3] Khan, K., Han, J., Zheng, Y., "Security Characterisation of Software Components and Their Composition", IEEE Proceedings 36th Conference on Technologies on Object-Oriented Languages and Systems (Tools Asia 2000), Xian, China, October 29-November 4 2000, pp 240-250,

[4] Maurer, P., "Components: What If They Gave a Revolution and Nobody Came?", IEEE Computer, 33-6. June 2000, pp. 28-34.

[5] OMG-DARPA-MCC Workshop on Compositional Software Architecture, January 6-8 1998, Monterey, California

[6] Selic, B.,"A Generic Framework for Modelling Resources with UML", IEEE Computer, June 2000, pp. 64-69.

[7] D'Souza, D., Wills, A. :Objects, Components, and Frameworks with UML - The Catalysis Approach, Addison-Wesley, 1998.

[8] Meyer, B., :Applying "Design by Contract", IEEE Computer, vol. 25, no. 1992, pp. 40-51

[9] Meyer, B., Mingins, C., "Providing Trusted Components to the Industry", IEEE Computer, June 1998, pp. 104-105.

[10]Thomson, C.: Workshop Reports. 1998 Workshop on Compositional Software Architectures, Monterey, California. http://www.objs.com/workshops/ws9801/report.html.

[11] Lindquist, U., Jonsson, E., "A Map of Security Risks Associated with using COTS", IEEE Computer, June 1998, pp. 60-66.

[12] Beugnard, A., et al., "Making Components Contract Aware", IEEE Computer, July 1999, pp. 38-46.