A Scenario Based Security Characterisation of Software Components

Khaled Md. Khan*, Jun Han, Yuliang Zheng
School of Network Computing
Monash University
McMahons Road, Frankston, Vic 3199,
Australia
e-mails: *k.khan@uws.edu.au, {jhan, yzheng}@monash.edu.au*

Abstract

In this paper, we propose a simple security characterisation model for software components. The proposed structure makes an attempt to model the security properties of interacting components based on 'scenarios'. The approach is adapted from the notion of 'scenarios' used in analysing qualities at the software architectural level. We argue that security of a software component does not exist in isolation but rather have strong binding with a use context such as whether a system is secure or not with respect to a specific threat. The notion of scenarios has led us to adopt a context-based security characterisation model of software components. We use message communication protocols and architectural description of components' functionality as valuable tools to describe a particular scenario of a composed system. This approach provides us with the means to identify the *required security properties* as well as *ensured security properties* of participating components in a compositional contract. The characterisation model is based on a simple structure that is used to capture the security properties of interacting components in a particular 'scenario'. The proposed structure consists of a predicate and four distinctive elements. The elements comprise *identities* of contracting components, *operations* to be performed in a compositional contract, *required* and *ensured security attributes*, and the *data* to be used in the particular compositional contract.

1 Introduction

The traditional approaches of developing software from scratch are being gradually taken over by the emergence of component-based software development. Application systems are increasingly assembled from pre-existing independent software components that are developed and available form third parties. OMG's CORBA, Sun's Enterprise JavaBeans and Microsoft's DCOM/ActiveX are the major players in such component-based technologies [1]. However, as software components are acquired from various sources and used in far greater scale, the ultimate security of the composed application system might be in question as the security properties of the 'foreign' components may remain unknown before a composition takes place. When software components developed and distributed by third parties are assembled with the application system, there are greater opportunities for security threats to the entire composed system. Security properties are considered non-functional quality-of-services of software components. Although the quality-of-services are considered important, main research focus and thrusts have been placed on achieving functionality [2]. In most cases, the security requirements of software component are not often well specified during the design of the components. The code and algorithms implementing the non-functional quality attributes are usually strongly interwoven with the codes that implement component functionality and business rules and logic [3]. As such, the specifications of the non-functional quality properties are lost, and remain unknown to the user before those are tested in real life context. Consequently, the much-sought trust cannot be established on the software components as anticipated [1]. This is particularly alarming for security intensive information systems such as e-commerce and health information systems. We believe that software

* The author is currently serving School of Computing and IT, University of Western Sydney

components have their own security properties, and those properties need to be adequately characterised in such a way that other components as well as human user could 'read' those features, and reason about their impact on the enclosing application system. The ability of a software component to reason about the security impact of other components is very important and desired as reported in [8], [9], [10], [11]. The very fundamental question is how a client software component or application system decides if a server candidate component is secure enough to be assembled with. In this paper, we propose a simple security characterisation model to address this issue.

Our proposed characterisation structure makes an attempt to model the security properties of interacting components based on 'scenarios'. The approach is adapted from the notion of 'scenarios' used in analysing qualities at the software architectural level [4]. A scenario can be better understood by means of message communication protocols and well defined architectural descriptions. We argue that security of a software component does not exist in isolation but rather have strong binding with a use context, e.g., whether a system is secure or not with respect to a specific threat [4]. The notion of scenarios has led us to define a context-based security characterisation model. A context-based characterisation approach provides us with the means to identify the *required security properties* as well as *ensured security properties* of interacting components in a compositional contract. Our proposed characterisation model captures the security properties of software components interacting in a particular 'scenario'. The proposed characterisation model is based on a structure that consists of a predicate and four distinctive elements. The elements comprise *identities* of contracting components, *actions* to be performed in a compositional contract, *required* and *ensured security attributes*, and the *data* to be used in the particular compositional contract.

In the next section, we give a brief overview of a scenario along with its associated protocols and architectural description which are used as a case study throughout the paper. In section 3, the related threats to the security of the composed system, and the security measures employed by the interacting components to withstand the threats are described. A structure to model the security characterisation of atomic components is presented in section 4. We conclude in section 5 with a brief note on our future work on the topic.

2. A system 'scenario'

In this section, we describe a distributed computation system for business tax calculations as an example that is used as a case study in the subsequent sections. Individual business application systems used by the business community can be dynamically assembled with a tax calculation software component over the open communication network. The name of the component is *TaxCalc*, a server component. The functionality of the *TaxCalc* is *calculate_tax_return*. The component calculates *tax* from a sale, and returns the result to the

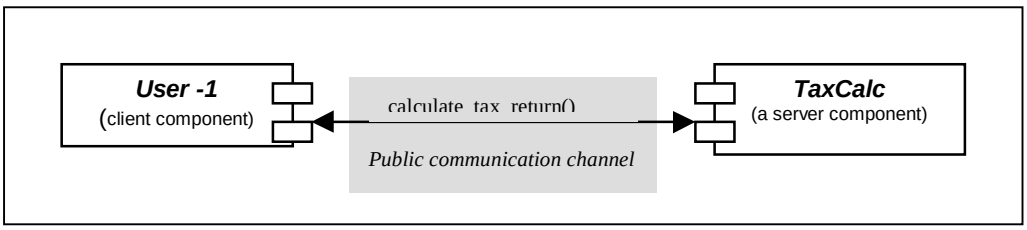


Figure 1. A scenario of a composed system

legitimate users. For the sake of simplicity, we assume there is only one legitimate client of the *TaxCalc* called *User-1* in a particular use scenario. When *User-1* sends a connection request for a compositional contract to the component TaxCalc (the server component), *TaxCalc* then identifies itself with its interface signatures. *User-1* also transmits its system identity in terms of its URL address. Whenever both components agree to establish a

compositional contract, the entire system is said to be composed. Figure 1 shows a composed system of *TaxCalc* and *User-1*. The shaded rectangle depicts the supporting infrastructure consisting of public communication channels, the connectors/middleware, communication protocols, and so on.

2.1 . Architectural description

In this compositional contract the main functionality is *calculate_tax_return().* To achieve this functionality, each participating component has *required* and *ensured* properties. These are outlined as follows.

TaxCalc	User-1
required:	*provided:*
tax_return_form	*tax_return_form*
provided:	*required:*
tax	*NULL*

The architectural description of this scenario is also partially illustrated in Figure 2 with the notation presented in [5].

```
Composed system Tax Return
Component TaxCalc
        Port provide = tax
        Port require = tax_return_form
        Spec TaxCalc specification…….
 Component User-1
        Port provide = tax_return_form
        Port require = NULL
        Spec User-1 specification……
Composite TaxCalc_User-1
        Role User-1 [User-1 protocol]       //possible behaviour of User-1
        Role TaxCalc [TaxCalc protocol]  //possible behaviour of TaxCalc
        Glue [glue protocol]   //connecting protocols are provided by the infrastructure
             //to combine the behaviours of TaxCalc and User-1  to form a composed system
```

Figure 2. Architectural description of the scenario

The bold texts denote the reserved words, a double black slash means starting of a comment line. The fundamental elements in the architectural description are briefly explained as follows.

- A **port** defines a logical point of interaction between the components and its environment such as provide-port (out data), require-port (in data)
- The **role** describes the expected behaviour of each component in a compositional contract such as
  - The **role** of User-1 describes its behaviour as a sequence of service requests (e.g., transaction_request()), and receiving of the results (e.g., tax_results())
  - The **role** of TaxCalc describes its behaviours as a sequence of handling of requests (e.g., transaction_accept_reject()), and return of results (e.g., tax_results())
- The **glue** specification describes how the activities of the TaxCalc and User-1 roles are coordinated, i.e., the interaction protocol.
  - The activities are sequenced in a time-order:
    - User-1 requests service
    - TaxCalc handles request

- User-1 provides tax_return_form
- TaxCalc calculates tax and returns the result
- User-1 receives the calculated tax .

2.2 Message communication protocols

We can use the message communication protocols of this functionality to describe interactions between components of the composed system. Allen and Garlan have successfully applied protocols to description of software interactions and connectors [5]. We, however, use the protocols a bit further; to define and analyse the security properties involved in functionality in a particular scenario. The message communication protocols of the scenario are presented in Figure 3. The detail of the protocols is not described here as the topic is beyond the scope of the paper. The shaded area shown in Figure 3 is the connector of the composed system. The protocol includes a transaction identifier within each data message that ties the message to the transaction. The data formats and contents are not defined or constrained by the protocols. A *transaction_request()* message is used by *User-1* to initiate a new transaction containing the transaction identifier and identity of the participating components.

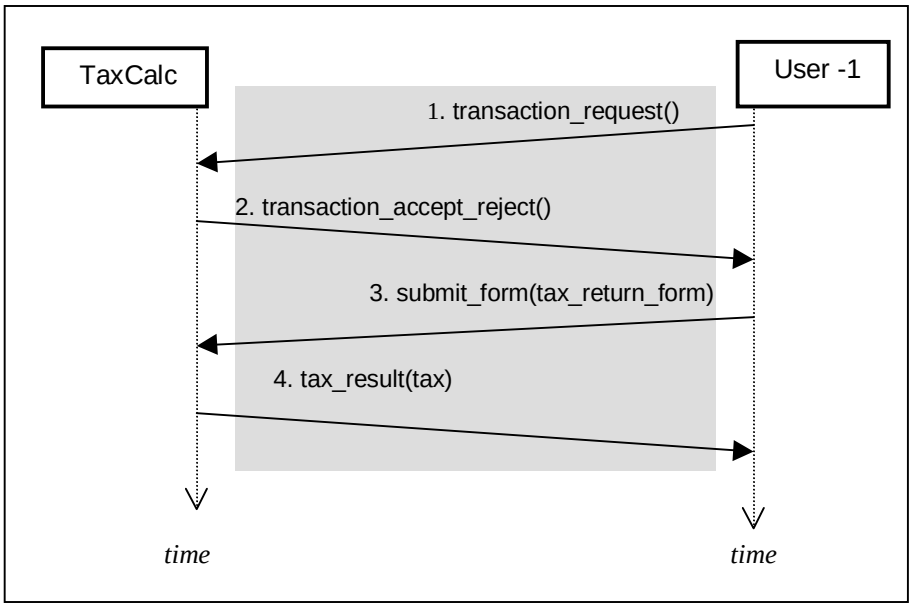


Figure 3.: Message communication protocols of the functionality "calculate_tax_return"

A *transaction_accept_reject()* message transmits either an accept or a reject message to *User-1*. Once the transaction is established, *User-1* sends the *tax_return_form* to *TaxCalc*, and *TaxCalc* transmits the *tax* value to *User-1* according to the protocol. These messages contain the transaction identity, the data, the message signature, identity of the recipient component and so on. The number assigned to each message depicts the order of the message transactions governed by the protocols.

3. Threats and security properties

We have described how functionality can be achieved in terms of coordinated interactions between two software components. The interactions have been illustrated with an architectural description and message communication protocols in the previous section. In this section, we describe how the issue of security is related to the functionality in a particular scenario.

Protecting data from unauthorised disclosure is known as confidentiality of data. It means that only the intended recipient of a message can make sense of it. If the message or data falls into wrong hands, the message becomes effectively useless. Confidentiality can be accomplished by some form of cryptographic technique. In our scenario, the messages passed between *TaxCalc* and *User-1* must remain confidential. In this section we describe the potential threats to our composed system, and the possible security measures employed by the participating components to withstand the threats.

3.1 Threats to the composed system

In our system scenario, let assume a hostile component identified as *Hostile-1* makes constant attempt to intercept the messages, and makes sense out of the messages exchanged between *TaxCalc* and *User-1. Hostile-1* is particularly interested in data such as *tax_return_form* and *tax*. It tries to intercept the *tax_return_form* and *tax* data from the transmission and extract the actual information out of these data. The attack capabilities of *Hostile-1* are rather limited such that it cannot modify the intercepted data, and is also unable to retransmit the modified data to the original recipient. The component can only tries to retrieve the data exchanged between two components. Figure 4 illustrates this scenario.

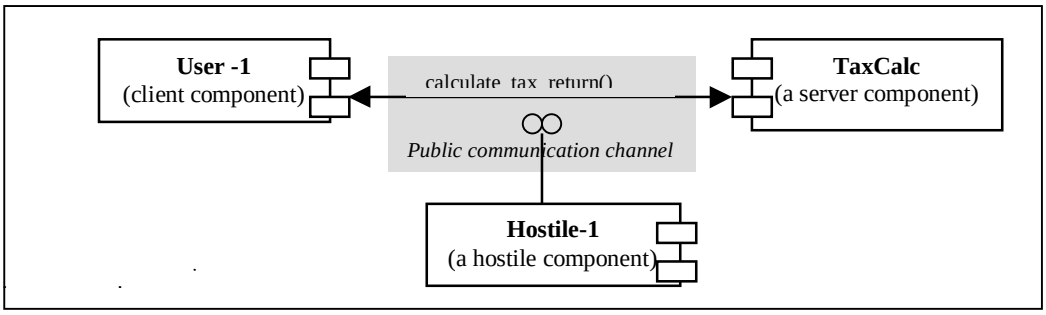


Figure 4. Threats to the compositional contract

3.2 Security properties

To withstand the threat pertained in this scenario, we can find two types of security mechanisms: one is employed by the underlying infrastructure such as protocols, connectors, and possibly much low level system properties like operating systems and hardware devices. The other type of properties is employed by the interacting components. From a compositional point of view, it is generally assumed that certain low level security properties are already in place by the supporting infrastructure such as protocols, middleware and operating systems. We assume the following security mechanisms are provided by the underlying infrastructure:

- All messages in the transaction use a session identifier and a session key such as those used in Secure Sockets Layer (SSL) and Secure Electronic Transmissions (SET) protocols [6], [7],
- Message ordering can also be verified by including a signed sequence number or virtual time stamp field in each message,
- Data files owned by the components are protected by the operating systems, and so on.

Detail discussion on these types of security is beyond the scope of this paper. Our main focus is to identify what *required* and *ensured* security properties are supported by the interacting components in a particular scenario to achieve a functionality. Whenever a compositional contract is made between *TaxCalc* and *User-1*, *TaxCalc* transmits its public key to *User-1,* and *User-1* sends its public key to *TaxCalc* for secure transmission of data. It

is assumed that the data exchanged between the *TaxCalc* and the *User-1* is considered confidential. To make the exchanged messages confidential, all messages sent to the *TaxCalc* must be encrypted with the public key of the *TaxCalc*.

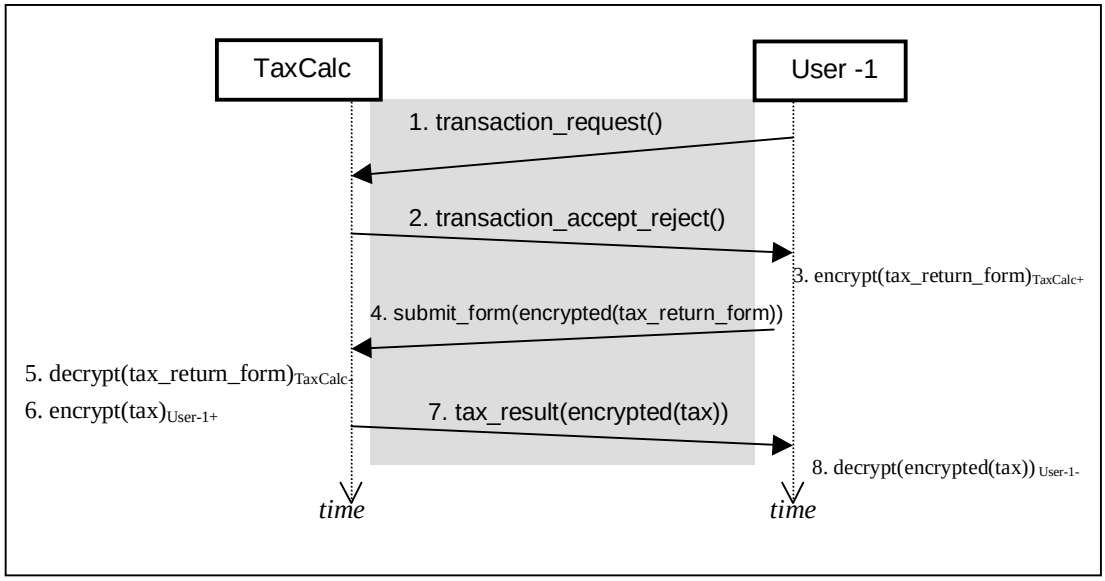


Figure 5. An extended communication protocol with security functions

User-1 transmits its *tax_return_form* which is confidential, and will remain confidential between *TaxCalc* and *User-1* only. Upon receiving of the *tax_return_form*, *TaxCalc* calculates the total *tax*, and transmits it to *User-1*. The *tax* data is also confidential. The following security policies and the supporting security functions are provided by the two interacting components:

- Security policy:
  Data such as tax_return_form, and tax contained in the message can only be observed by TaxCalc and User-1 (confidentiality)
- Security functions:
  - *tax_return_form* data is encrypted with the public key of TaxCalc
  - TaxCalc decrypts the encrypted *tax_return_form* with its corresponding private key
  - TaxCalc encrypts the *tax* data using the public key of User-1
  - User-1 decrypts the encrypted *tax* data using its corresponding private key.

These security functions are well illustrated in an extended message communication protocol as shown in Figure 5. TaxCalc- and TaxCalc+ used in Figure 5 denote the private key and the public key of TaxCalc respectively. Similarly User-1- and User-1+ show the private and public keys of User-1 respectively. The operations cited outside the shaded rectangle illustrate the internal operations of the corresponding components.

We apply these extended protocols to description of software interactions between two components. The architectural description of the security properties supported by *TaxCalc* and *User-1* is presented in Figure 6. Figure 6, in fact, describes a security contract between components *TaxCalc* and *User-1*. The security properties illustrated in Figure 6 can be translated into various security policies as shown in Figure 7.

4. A characterisation model

If we analyse the security policies of Figure 7, we can identify a common format that can be used to capture the entire security properties. Consider the following required security policy defined in the component *TaxCalc*.

A required security property of TaxCalc*:*

tax_return_form is to be encrypted by User-1 with the TaxCalc's public key

In the security policy above we can find four types of information as circled: *identity* (*User-1*), an *operation* (*encrypt*), a *security attribute* (public key of TaxCalc), and a *data* (tax_return_form).

Composed system Tax Return
Component TaxCalc
ensure = $encrypt(tax)_{User\text{-}1+}$ //encrypt with the public key of User-1
require = $encrypted(tax\_return\_form)_{TaxCalc+}$ //encrypted with the public key of TaxCalc
Spec TaxCalc specification…….
Component User-1
ensure = $encrypt(tax\_return\_form)_{TaxCalc+}$ //encrypt with the public key of TaxCalc
require = $encrypted(tax)_{User\text{-}1+}$ //encrypted with the public key of User-1
Spec User-1 specification…….
Composite TaxCalc_User-1
Role user-1 [User-1 protocol] // possible security behaviour of User-1
Role TaxCalc [TaxCalc protocol] //possible security behaviour of TaxCalc
Glue [glue protocol] //combining the security behaviours of two roles to form a composed system

Figure 6. Architectural description of the security properties of the composed system

With these four elements we can formulate a characterisation structure as

a_security_function(S, O, K, D)

where S is the identity of a component in a particular scenario, O is an arbitrary set of operations performed by the component S, K is a set of security attributes used by the member of S to do the operation O. D is an arbitrary set of data or information that belongs to a component. More on this can be found in [12]. We can now apply this structure to the security policies shown in Figure 7 to characterise the security properties. The resulting characterisation instances for the particular scenario are shown in Figure 8.

Security policies of TaxCalc :
required security property:
tax_return_form is to be encrypted by *User-1* with *TaxCalc's* public key
ensured security property:
tax is to be encrypted by TaxCalc with the public key of *User-1*

Security policies of User-1 :
ensured security property:
tax_return_form is to be encrypted by User-1 with the public key of *TaxCalc*
required security property:
tax is to be encrypted by *TaxCalc* with *User-1's* public key

Figure 7. Security policies of the two interacting components

Component: TaxCalc

required security property:

protect_in_data("User-1", encrypt, $key_{TaxCalc+}$, "tax_return_form")

ensured security property:

protect_out_data("TaxCalc", encrypt, $key_{User-1+}$, "tax")

Component: User-1

ensured security property:

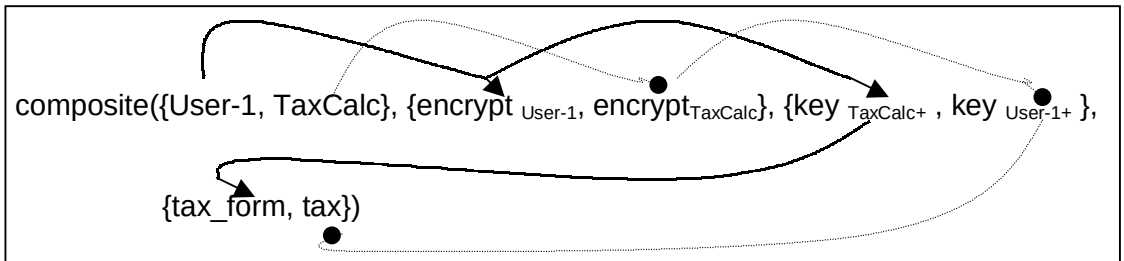
protect_out_data("User-1", encrypt, $key_{TaxCalc+}$, "tax_return_form")

required security property:

protect_in_data("TaxCalc", encrypt, $key_{User-1+}$, "tax")

Figure 8. Security characterisation instances of components *TaxCalc* and *User-1*

From these instances we can further characterise a single compositional primitive of the entire composed system as depicted in Figure 9 using the same characterisation structure used for atomic component. In Figure 9, the thin dotted lines with round filled head show that operation encrypt is performed by *TaxCalc* on the data *tax* with the corresponding security attributes key $_{User-1+}$. Whereas, the solid thick lines with filled arrow denote the operation, the corresponding security attributes and the associated data related to the component *User-1*.
The first argument {User-1, TaxCalc} specifies the identities of all participating components in a particular scenario. The second argument {$encrypt_{Users-1}$, $encrypt_{TaxCalc}$} shows different ordered operations performed by the participating components. The third argument {$key_{TaxCalc+}$, key $_{User-1+}$ } specifies the corresponding ordered security attributes of the operations used by the components. The fourth argument {tax_form, tax} shows data used in the compositional contract.



Interpretations of the arguments:-

- *identities* of all participating components
  e.g., {User-1, TaxCalc},

- ordered *operations* performed by the participating components respectively
  e.g., {encrypt $_{Users-1}$, $encrypt_{TaxCalc}$},

- corresponding ordered *security attributes* of the operations respectively
  e.g., {key $_{TaxCalc+}$, key $_{User-1+}$ }

- *data* used in the compositional contract respectively
  e.g., {tax_form, tax})

Figure 9. Characterisation of compositional primitive

The operation with the component identity as its subscript shows that the operation is performed by that particular component. The number of operation must have the same number of corresponding security attributes as shown with {key $_{TaxCalc+}$, key$_{User-1+}$ }. The chronological order of the security attributes must be consistent with the order of the operations. This composition primitive is a production of two binary compositional relationships.

5. Conclusion and further work

In this paper we have proposed a component security characterisation model based on a specific scenario. We have demonstrated with a simple example that a specific scenario can be well presented in terms of message communication protocols and architectural descriptions. These tools are used to identify the individual operations, their *required* and *ensured* security properties, specific threats, the associated security policies and functionality for a specific scenario of the composed system. The characterisation model is based on a predicate-like structure consisting four arguments. The structure has been used to model the security characteristics of atomic components for a specific use context. We have also demonstrated with a simple instance that the same structure can also be used to model the compositional primitive of two interacting components. However, the example used in the paper is a simple composition with simple threats. More complex threats and complicated security properties are currently being actively investigated.

We are currently working to refine and formalise the binary compositional rules for security characterisation. Our work is intended to combine the security specification of an atomic component with that of another component in order to model a compositional security characterisation. The compositional characterisation is intended to define rules about the impact of combining two or more primitive security characterisations in a given time.

References

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

[2] Han, J., Zheng, Y., "Security Characterisation and Integrity Assurance for Software Components and Component-Based Systems", 1998 Australasian Workshop on Software Architectures, Monash University, Nov. 24 1998, pp. 83-89

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

[4] Bass, L, Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley, 1998.

[5] Allen, R., Garlan, D.,"Formalising Architectural Connection", Proc. 16th Int'l Conf. on SW Engineering, IEEE Computer Society, 1994, pp. 71-80.

[6] Steves, D., Yurkanan, C., Mohamed, G., "Properties of Secure Transaction Protocols", Proceedings JENC8, 1998, pp 713-1:10.

[7] VISA, MasterCard, et al. "Secure Electronic Transaction Specification", Books 1-3, WEB document, June 1996.

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

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

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

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

[12] Khan, K., Han. J., Zheng, Y., "Security Characterisation of Software Components and Their Composition", IEEE Proceedings 36th Int' l Conference on Technology of Object-Oriented Languages and Systems (TOOLS-Asia 2000), Oct. 30- Nov. 4 2000, Xi' an, China, pp. 240-249.