J. Lopez, J. J. Ortega, J. M. Troya, and J. L. Vivas, "How to Specify Security Services: A Practical Approach", *7th IFIP Conference on Multimedia and Communications Security (CMS03)*, LNCS vol. 2828, pp. 158-171, 2003.
NICS Lab. Publications: https://www.nics.uma.es/publications

# How to specify security services: a practical approach

Javier Lopez[1], Juan J. Ortega[1], Jose Vivas[2], Jose M. Troya[1]

[1]Computer Science Department, E.T.S. Ingeniería Informática
University of Malaga, SPAIN
{jlm, juanjose, troya}@lcc.uma.es

[2]Hewlett-Packard Labs.
Bristol, UK
jsv@hplb.hpl.hp.com

**Abstract.** Security services are essential for ensuring secure communications. Typically no consideration is given to security requirements during the initial stages of system development. Security is only added latter as an afterthought in function of other factors such as the environment into which the system is to be inserted, legal requirements, and other kinds of constraints. In this work we introduce a methodology for the specification of security requirements intended to assist developers in the design, analysis, and implementation phases of protocol development. The methodology consists of an extension of the ITU-T standard requirements language MSC and HMSC, called SRSL, defined as a high level language for the specification of security protocols. In order to illustrate it and evaluate its power, we apply the new methodology to a real world example, the integration of an electronic notary system into a web-based multi-users service platform.

## 1 Introduction

Many problems with security critical systems arise from the fact that developers seldom have a strong background in computer security. However, nowadays it is widely accepted that an adequate specification of a system is required in order to obtain a robust implementation. There is currently an increased need to consider security aspects at the early stages of system development. This need is not always met by adequate knowledge on the part of the developer. This is problematic since security is most often compromised not by breaking the dedicated mechanisms, but by exploiting weaknesses in the way those mechanisms are used. Therefore security mechanisms cannot simply be inserted into the system as an afterthought. In consequence, security aspects should be considered already at an early stage of the software development life cycle.

Results obtained using formal specification techniques are not readily applicable in the context of a real world development environment. First of all there is a requirements engineering problem: how to capture the intended security requirements. Then we have an implementation problem. Thus, it is not obvious how to reconcile the mathematical notion of a perfect public key with the fact of a stored file representing

a couple of numbers n and e encoded according to the *Basic Encoded Rules* (BER). Also, although we often talk about secure channels, in reality what we have are things such as https connections.

Security requirements are commonly expressed as system constrains, but often they are in fact a kind of service that must be provided by a variety of mechanisms. In this sense, security requirements are not different from e.g. real-time requirements, and should be treated in an analogous way. Accordingly, we refer to these services as security services.

The rest of this paper is organized as follows. In Sect. 2 we present some common security concepts. In Sect. 3 we give an overview of a couple of representative specification languages. Sect. 4 is dedicated to a brief introduction of the communication requirements language *Message Sequence Charts* (MSC), and the *High-Level MSC* (HMSC). In Sect. 5 we describe a new specification language, SRSL, which is an extension of MSC and HMSC. Sect. 6 is dedicated to the description of an application of SRSL to a real world example, and in Sect. 7 we present some conclusions.

## 2 Specification of security properties paradigm

A security protocol [7] is a general template describing a sequence of communications and making use of cryptographic techniques to meet one or more particular security related goals. The basic security services [11] provided by security mechanisms (cryptographic algorithms and secure protocols) are authentication, access control, data confidentiality, data integrity, and non-repudiation.

The notion of authentication includes authentication of origin and entity authentication. Authentication of origin can be defined as the certainty that a message that is claimed to proceed from a certain party actually originated from it. As an illustration, if during the execution of a protocol Bob receives a message, supposed to come from Anne, then the protocol is said to guarantee authentication of origin for Bob if it is always the case that, if Bob's node accepts the message as being from Anne, then it must indeed be the case that Anne sent exactly this message earlier. Thus, authentication of origin must be established for the whole message. Moreover, it is often the case that certain time constraints concerning the freshness of the message received must also be met. Entity authentication protocols, by its turn, guarantees that the claimed identity of an agent participating in the protocol is identical to the real one.

Access control service ensures that only authorized principals can gain access to protected resources. Usually the identity of the principal must be established; hence entity authentication of origin is also required here.

Confidentiality may be defined as prevention of unauthorized disclosure of information.

Data integrity means that data cannot be corrupted, or at least that corruption will not remain undetected. If it were possible for a corrupted message to be accepted, then this would show up as a violation of integrity and the protocol must be regarded as flawed.

Non-repudiation provides evidence to the parties involved in a communication that certain determined steps of the protocol have occurred. This property appears to be

very similar to authentication, but here the participants are given capabilities to fake messages up to the usual cryptographic constrains. It uses signature mechanisms and a trusted notary.

These services are enforced using cryptographic protocols or similar mechanisms, and it is essential to determine which ones are needed. In order to specify a security system it is not necessary to know how the analysis of the system will be carried; however, it is absolutely indispensable to identify the security services required.

## 3 Overview of security specification languages

We focus here on two kinds of specification languages applied to security systems: languages for software engineering and languages for the design and analysis of cryptographic protocols [6].

A representative example of the first kind of language is UML - *Unified Modeling Language* [10]. UML is a language for the specification, visualisation, development, and maintenance of software systems. The notation consists basically of graphical symbols, including a set of diagrams giving different views of a system.

J. Jurjens [3] has defined an extension of UML, UMLsec, to specify standard security requirements on security-critical systems. The aim of this work is to use UML to encapsulate knowledge on prudent security engineering and to make it available to developers not specialised in security. It is based on the most important kinds of diagrams for describing object-oriented software, class diagrams, state-chart diagrams, and interaction diagrams, and uses the basic elements offered by UML to extend the language, i.e. stereotypes, tagged values, and constraints.

An example of the second kind of language is CAPSL [8]. This is a high-level formal language intended to support security analysis of cryptographic authentication and key distribution protocols. A protocol specification in CAPSL can be translated into a *multiset rewriting* (MSR) rule intermediate language like CIL, and the result can be used as input to different security analysis tools.

A CAPSL specification has tree sections: protocol specification, type specification, and environment specification. The type and environment specifications are optional. A protocol specification is a description of behaviour and consists of three parts: declarations, messages, and goals. A type specification defines cryptographic operators, whereas an environment specification provides scenarios used by model-checking tools to verify the protocol. A CAPSL extension called MuCAPSL is also under development [9], intended to support the specification of protocols for secure multicast.

We believe it is important to develop a specification language integrating both kinds of languages. As a first approach to achieve this aim we propose here a new language, the *Security Requirements Specification Language* (SRSL), based on Message Sequence Charts. In the next section we give an overview of the latter, and in the subsequent one we introduce the SRSL.

## 4 A Requirements Language for Communication Protocols: MSC

The ITU-T's Standardization Sector specifies *Message Sequence Charts* [2] (ITU-T Z.120) as the requirements language for the visualization of system runs or traces within communication systems. MSCs can be defined as a trace language for describing message interchanges among communicating entities. It is endowed with a graphical layout that gives a description of system behaviour in terms of message flow diagrams that is both clear and perspicuous. MSCs focus on the communication behaviour of system components and their environments, and are widely used as follows: for requirements definition; for specification of process communication and interface; as a basis for automatic generation of *Specification Description Language* (SDL) [1] skeletons; for selection and specification of test cases; and for documentation. It is used most frequently together with SDL.

The basic language constructs of MSCs are instances and messages. Instances are graphically represented by an axis, i.e. a vertical line or a column. An entity name and an instance name can be specified within an instance heading in the graph. A total ordering of the communication events is specified along each instance axis. Actions describing an internal activity of an instance, in addition to message exchange, may also be specified. The system environment is also represented by a frame symbol forming the boundary of the diagram.

Instances may also be created from a parent instance. Instance creation is described by a special symbol in the shape of a dashed arrow that can be associated with textual parameters. Termination of instances is also possible, and is represented by a stop symbol in form of a cross at the end of an instance axis.

It is also possible to specify conditions describing a state associated with a non-empty set of instances. Conditions can be also used for sequential composition of MSCs. MSCs can be used to describe the behavior of a subsystem or component intended to be combined in different ways into a more complex system.

In addition, MSCs may also be combined with the help of expressions consisting of composition operators and references to the MSCs. MSC references can be used either to reference a single MSC or a number of MSCs using a textual MSC expression. The MSC expressions are constructed from the operators *alt*, *par*, *loop*, *opt* and *exc*, described below.

The keyword *alt* denotes alternative executions of several MSCs. Only one of the alternatives is applicable in an instantiation of the actual sequence.

The *par* operator denotes parallel executions of several MSCs. All events within the MSCs involved are executed, with the sole restriction that the event order within each MSC is preserved.

An MSC reference with a *loop* construct is used for iterations and can have several forms. The most general construct, *loop<n,m>*, where "*n*" and "*m*" are natural numbers, denotes iteration at least *n* and most *m* times.

The *opt* construct denotes a unary operator. It is interpreted in the same way as an alt operation where the second operand is an empty MSC.

An MSC reference where the text starts with *exc* followed by the name of an MSC indicates that the MSC can be aborted at the position of the MSC reference symbol

and instead continued with the referenced MSC. MSC references with exceptions are used frequently.

High-level MSCs [2] provide a means to graphically define how a set of MSCs can be combined. An HMSC is a directed graph where each node is a start symbol, an end symbol, an MSC reference, a condition, a connection point, or a parallel frame. The flow lines are used to connect the nodes in the HMSC and indicate the sequencing that is possible among the nodes. The incoming flow lines are always connected to the top edge of the node symbols, whereas the outgoing flow lines are connected to the bottom edge. If there is more than one outgoing flow line from a node this indicates an alternative. The conditions in HMSCs can be used to indicate global system states or guards and impose restrictions on the MSCs that are referenced in the HMSC. The parallel frames contain one or more small HMSCs and indicate that the small HMSCs are the operands of a parallel operator, i.e. the events in the different small HMSCs can be interleaved.

The connection points are introduced to simplify the layout of the HMSCs and have no semantic meaning. High-level MSCs can be constrained and measured with time intervals for MSC expressions. In addition, the execution time of a parallel frame of an HMSC can be constrained or measured. The interpretation is similar to the interpretation of Timed MSC expressions.

## 5 The SRSL language

The specification language proposed here is an extension of ITU standard requirements language MSC and HMSC. SRSL [4] is a high level language intended to specify cryptographic protocols and secure systems. Such a language must be modular, easy to learn, and able to express security notions.

The main design criteria we have used during development of the SRSL language are the following:

- The language should provide visualization capabilities. A graphic representation offers perspicuous views of a system and clarifies the communication among customers, users and developers.
- It should be possible to produce specifications of a system at several levels of abstraction. Different stakeholders may have different views, and accordingly so we require that it be possible to describe a system at several levels of abstraction.
- The language should make available mechanisms for modularisation. Standard implementations of a part of the system are often used as components of the whole system. Standard modules may be defined, implemented and reused. For instance, it is common to implement a new system using well-defined standards protocols. Thus, if we need a server authenticated and encrypted connection, we can make use of a standard TSL protocol. In this way it becomes also easier to use a formerly implemented library in any program language.
- Standard languages should be reused as much as possible. Using standard languages have a lot of benefits. Usually it is difficult to learn a new language or

methodology. In the beginning there may be not enough tools to support it, or they may be too inefficient. Standard languages typically have tools that support them, and many users may have previous experience with them.

- The language should be suitable for validation purposes. When specifying a system we usually describe what the system is supposed to do. However, we need also be able to validate the system, i.e. to show that it is defined according to the specification.
- It should be possible to express security requirements in the language. Unfortunately, there is currently no standard notation to represent those requirements.

The SRSL has two levels of representation: multi-layer module scheme and security scenario description.

The multi-layer module scheme describes security systems in terms of a multi-layered structure. The first layer is the communication medium, also used to study attack strategies in the security analysis phase, i.e. the intruders' behaviour. The other layers depend on the security mechanisms defined during system development. This description is translated into a security scenario representation using standard package definitions.

As an example, we consider a system that uses SSL security mechanisms in order to achieve server authentication and confidential communication. Its design must ensure that an application server (the responder) is given evidence of the fact that a sender (the initiator) has previously sent some message, i.e. the protocol must be able to ensure non-repudiation of the origin of messages sent to the server. The module specification is depicted in figure 1.

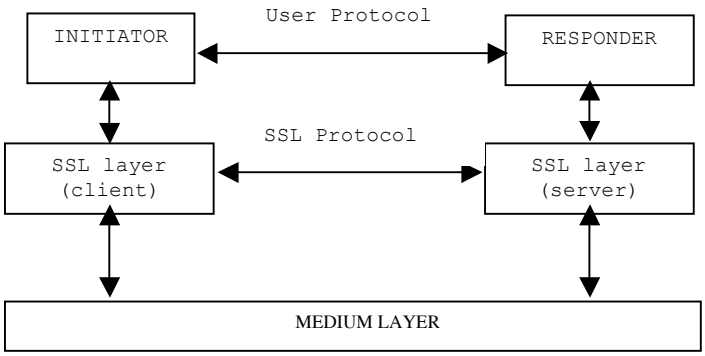


**Fig. 1.** SRSL Module Specification

The SSL layer is described in a standard security communications package. Therefore, part of medium layer is generated automatically. Consequently, we only have to specify the Initiator-Responder protocol. It is composed of simple scenarios described in MSC. The SRSL security scenario description is divided into three parts; the specification of protocol elements, the message exchange flow, and the security services requirements.

The extensions proposed concern the definition of entities, the definition of data types related to security aspects, and the security services that are going to be used or analysed. These are described in comment text boxes, and are intended to be examined during the security analysis phase. These elements required to define a security protocol can be divided into several categories. These are explained as follows (keywords are in cursive):

- Entities: *Agent* (*Initiator*/*Responder*), the principal's identification; *Key_Server*, which provides cryptographic keys; *Time_Server*, which provides time tokens; *Notary*, which registers the transaction; *Server Certification Authority* (*SCA*), which validates a certificate.
- Messages: *Text*, of type clear text; *Random_Number*, an integer; *Timestamp*, giving the actual time; *Sequence*, the count number.
- Keys: *Public_key*, e.g. in PKCS#12 format (aprostrophe symbol (') reference to private key); *Certificate*, a public key signed by CA; *Private_key*, used to sign documents; *Shared_key*, a secret key shared by more than one entity; *Session_key*, a secret key used to encrypt transmitted data.

In addition, SRSL may operate with previously defined data types. These operations are: *Concatenate*, composition of complex data (operator ','); *Cipher* (operator "{" "}"), which provides cipher data resp. cleartext data (e.g. RSAcipher/RSADecipher PKCS#1 format); *Hash*, the result of a one-way algorithm; *Sign* (operator "[" "]"), a message hash encrypted with signer's private key (e.g. RSAsign PKCS#7). Furthermore, user-defined functions are also considerated.

The message exchange is defined in MSC, the requirements language most widely utilized in telecommunications, and its extension HMSC, explained in section 4. The is known for its high degree of flexibility and is universally accepted in protocol engineering.

The security services requirements section are also described in comment text boxes. We use three different security statements: *Authenticated(A,B),* stating that B is certain of the identity of A; *conf(X)*, stating that the data *X* cannot be deduced (also called confidentiality); *NRO(A,X)* ( non-repudiation of origin ), stating that that data *X* (the evidence) must have originated in *A*. These statements have been formally defined in [5].

Furthermore, an automatic translator program [7] is used to produce the SDL version of the system from SRSL. The SDL system produced is subsequently used to analyse security requirements of the system during the analysis phase.

In the specification stage we must consider two different kinds of tasks. The first one concerns the specification of a system yet to be developed, in which case we have more freedom to choose what security mechanisms to use. The second one concerns the specification of an already implemented system.

In order to specify a system, we have to identify different functional parts. These are represented as composition of MSCs using HMSC. Each part is considered separately. Next, we specify possible scenarios without regard to security requirements, i.e. taking into consideration only the purely functional aspects. As an example, if we want to specify the access to a data bank account via the internet, we only describe the data request and the bank reply.

Figure 2 depicts a scenario consisting of a sequence of message exchanges, described in MSC. As we can see, we have two entities: the *User_browser* and the

*Bank_portal*. The user, by way of a browser, asks for access to the bank's portal in order to request his data bank.

We obtain two alternative scenarios according to whether the access request is accepted or rejected. First the user sends its account number and data request. If the request is accepted, the *Bank_portal* sends the *User_data_bank*. Otherwise, if the request is rejected, a rejection data is sent instead. The transmitted data is finally displayed in the *User's browser*.

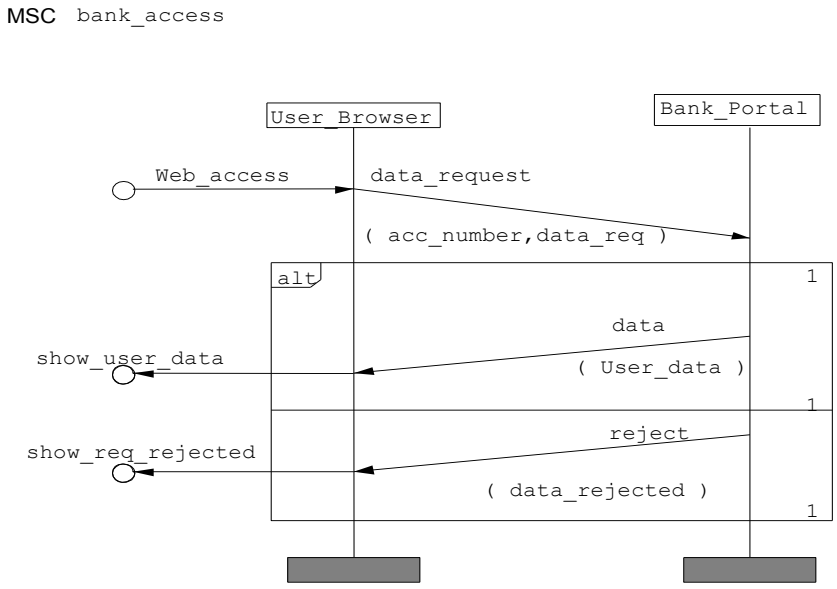


**Fig. 2.** MSC scenario of user's access to data bank

We can now analyse the specification in order to verify and validate the functional requirements, whereupon we may proceed to analyse the security requirements. These are defined in a comment text box. If a new system is being designed, we include here the security mechanisms needed to meet these requirements. Otherwise, we include the existing security mechanisms instead.

In Figure 3, we describe the process of integration of the security requirements into the specification. The security requirements are the following:

```
1- Authenticated(Bank_Portal,User_Browser);
2- Authenticated(User,Bank_Portal);
3- conf(account_number);
4- conf(data_req);
5- conf(user_data);
```

The first requirement means that user's browser must authenticate itself to bank's portal. This is achieved with the help of an HTTPS-connection.

The second requirement means that the user must authenticate itself to the bank's portal. This is accomplished by a mechanism that asks for the user's identification and password, and subsequently validates it.

The three last requirements mean that the data transmitted is confidential. This goal is accomplished by making use of the session key established during the https connection.

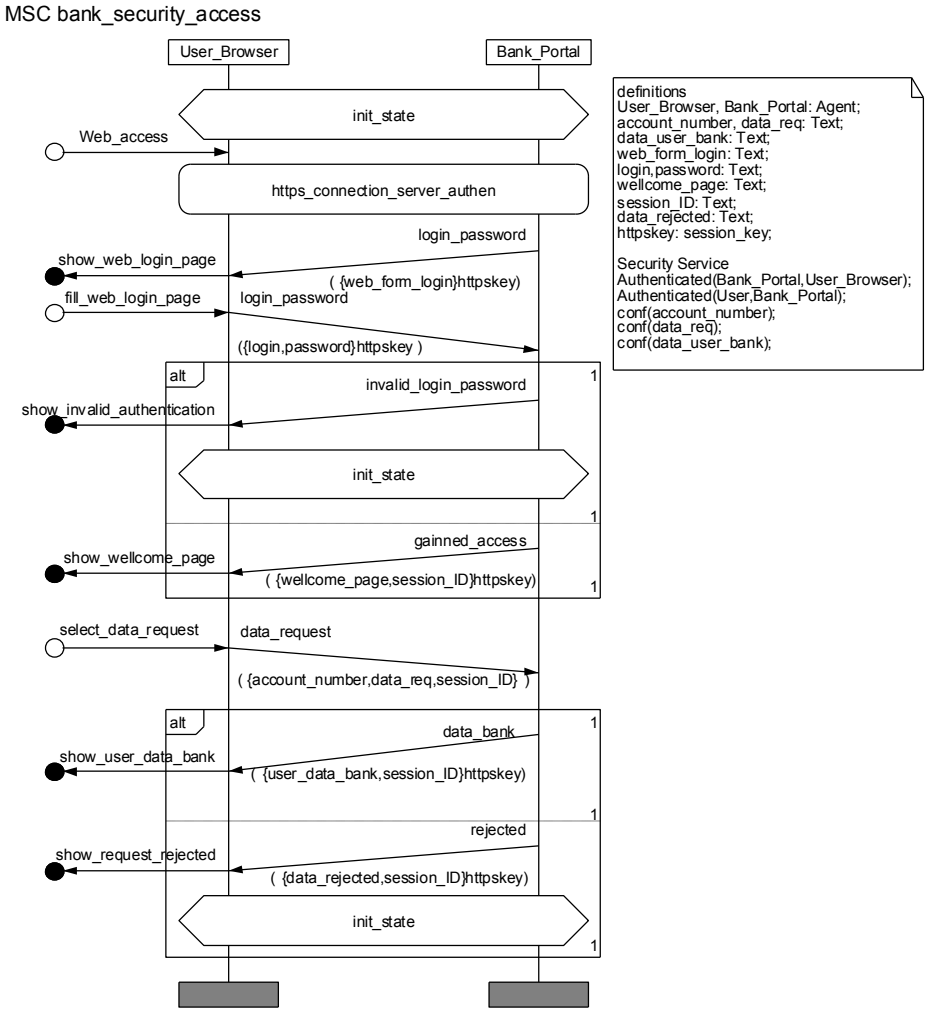


**Fig. 3.** SRSL security scenario of user's access to data bank

We might as well have considered alternative security mechanisms to meet the five requirements. The important point to note here is that we have chosen a form of specification that does not bind the developer to any particular security mechanisms, thus enhancing separation of concerns and modularity. This is accomplished by allowing

the security requirements to be defined at a high level of abstraction and independently of the definition of the functional requirements.

In the case that we have a system that is already implemented, i.e. a legacy system, and we want to analyse or document it, we describe instead the security mechanisms that have been implemented.

## 6 A case study: on-line contracting processes

We have applied our methodology to a system currently being developed by an IT company that plays the role of user partner in the EU-project where this work has been performed. This is working on a virtual enterprise business scenario implementing on-line contracting processes by integration of *Trusted Third Party* services (TTPs) such as electronic notary systems into a web-based multi-users services platform. The current on-line contracting process is rather complex and supports several activities such as contract creation, negotiation, signing and final archiving.

To start with, we focus on the contract signing process (managing the contract signing and notarisation process control). This procedure is part of the business-to-business scenario for setting up a virtual enterprise platform integrating technology components such as e-contracting, e-notary and role based authorization engines.

This section describes the existing electronic notary process within an e-business scenario. The central core of this set-up is the MESA platform, developed by the IT company. MESA provides web-based user interfaces and role based control mechanisms for accessing functions made available by the TTPs.

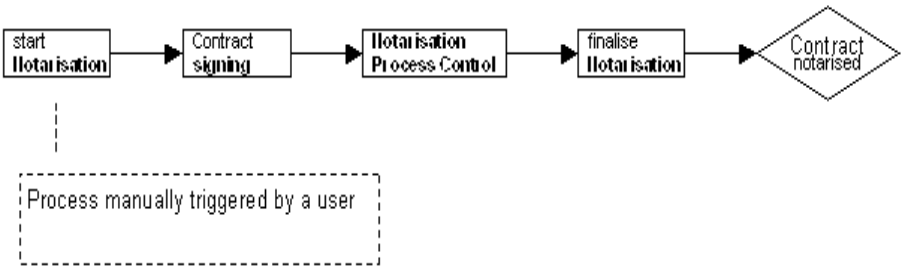


**Fig.** 4 Contract signing process

The following diagram (depicted in figure 5) describes the contract signing process implemented by an e-Notary reference application and used within the IT company scenario. A user intending to access a web-based user interface provided by the MESA platform triggers manually the contract signing process within the following business scenario:

In the sequel we describe the contract signing process, including the security requirements and the relationships among the users, the MESA platform, and the e-notary service.

Our methodology has been used to examine this process in terms of communication security issues. The intended goals have been to validate the model and evaluate both the current reference implementation and a proposed extension to an agent-based scenario for the reference implementation.

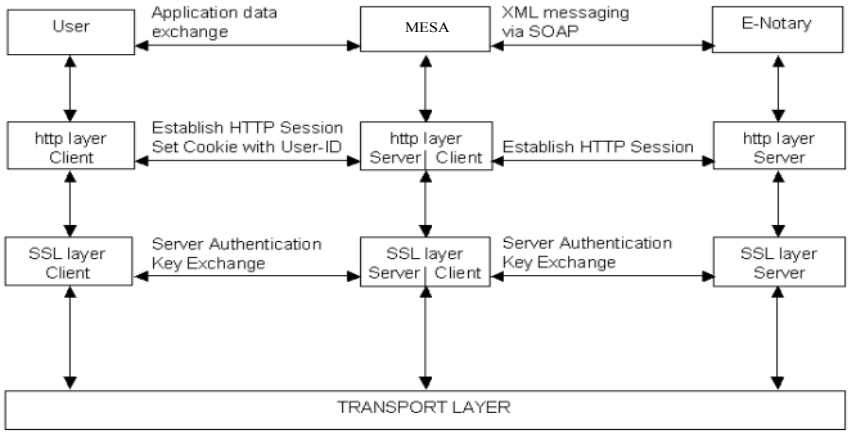


**Fig. 5.** Application structure in SRSL module description

This implementation is being used within the current business scenario. However, the current client/server implementation, based on traditional PKC technology, has inherent problems in terms of flexibility and scalability. While the reference scenario requires a certain infrastructure, compliance to the European directives concerning digital signatures, to alternative PKC technologies and to certificate infrastructures might be more suitable when adopting the e-notary process within other business scenarios (with different context of actors, contents, legal requirements and liability issues). In fact, changing the context of a recent e-Notary deployment scenario and identification of implications in terms of security are the most interesting challenges we face.

We have to pay attention to the fact that what we have specified here is a newly implemented system. Therefore the task has been to describe the behavior of current application in order analyze and improve the current implementation. We started by emphasizing for the developers the usefulness of elaborating a system specification intended to clarify the different scenarios in order to increase our understanding of them and to avoid certain ambiguities. We show now the definition of the system module description representing the different layers structured according to the security services.

The protocols described in each layer are specified in terms of MSC/HMSC diagrams. Many are standard protocols and so they are instances of generic specifications.

The system (see Figure 6) is divided into three parts: the contract creation process, the signing process, and the notarization process. A diagram in a lower abstraction level describes each MSC reference.

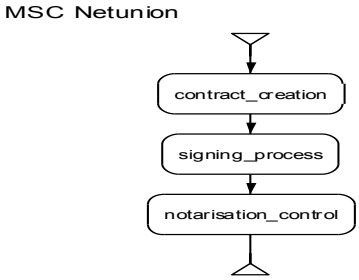


**Fig. 6.** HMSC application description

A representative part of the specification is the *create_contract* scenario (Figure 7).

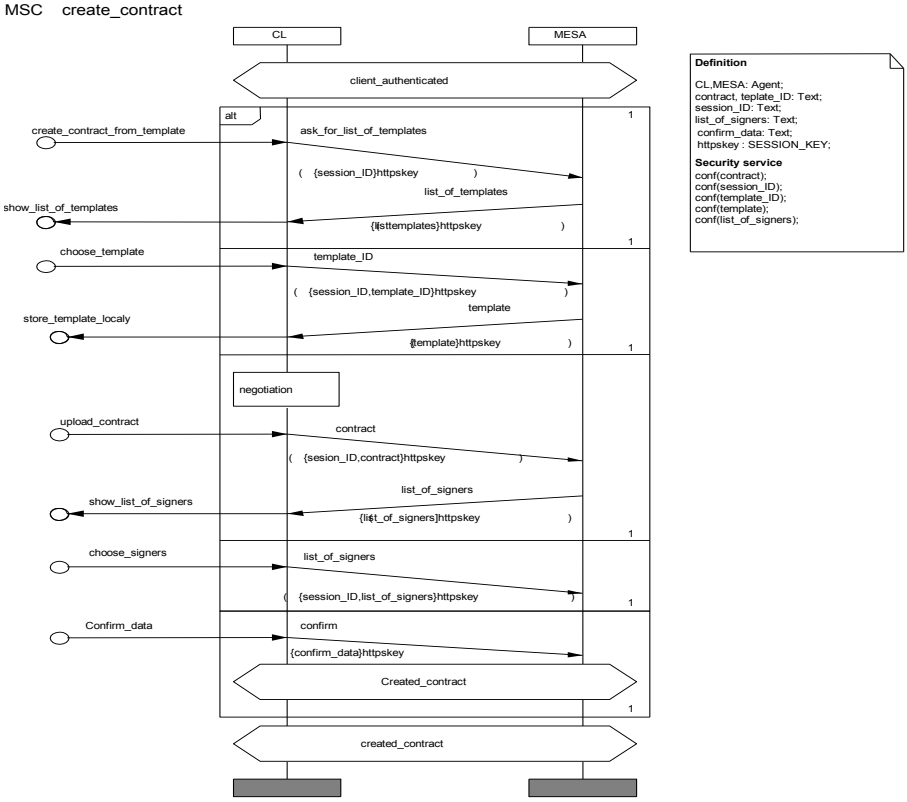


**Fig. 7.** SRSL create_contract security scenario

The *contract leader* (CL) triggers the contract creation process. Previously, the contract leader and the MESA platform had to be authenticated, and a HTTPS session key exchanged. This is represented by the initial state *client_authenticated*. The scenario is divided into four independent alternatives (*alt*-operator). In the third sub-

scenario we use the task MSC operator to specify the possibility of an external negotiation agreement that is not part of our system. The fourth sub-scenario ends the process by accepting the uploaded contract and starting the next scenario in the state *created_contract*.

Developers considered this methodology very useful for their purposes, especially with regard to the specification of the contract signing process (Figure 8). The notification was initially implemented by letting the E-notary service send an e-mail to each signer. However, this procedure is unreliable since it lacks any kind of security guarantees. When this fact was drawn to attention of the developers, they decided to modify the system in order to provide for security services, such as the signing of the e-mail by the e-notary service to ensure non-repudiation of origin (NRO). This has been appended to security services section, and a signing mechanism has been included in the definition of the e-notary. This mechanism is checked later in analysis phase.

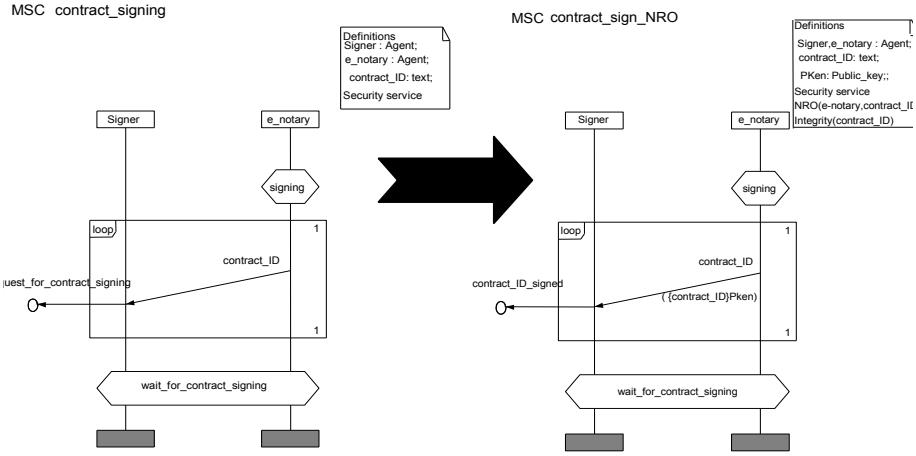


**Fig. 8.** SRSL *contract_signing* security scenario & non-repudiation improvement

The developers deemed this methodology easy to learn and to apply in real environments. They believed that it has been of great help for understanding the implementation and that it provided a method to improve the application with regard to the required security services and mechanisms. Furthermore, the methodology made available a formal method of analysis that increased the developers and users reliance on the system.

## 7 Conclusions

We have studied different methods for designing and analyzing a system containing communication protocols. We observed that in order to define a secure system, the developers need a unified framework that allows them to integrate the security aspects into both the software system itself, or at least relevant parts of it, and the communi-

cation protocols that constitute a part of the total system. The methodology consisted of an extension of the ITU standard requirements language MCS, called SRSL, a high level language for the specification of cryptographic protocols.

In order to illustrate the methodology, we have shown an application consisting of an electronic notary process scenario that the developers wanted to validate and improve. Moreover, we have described how this electronic Notary process can be inserted into a different scenario, given different input parameters. In this way, we were able to offer a framework within which it became possible to define and to evaluate different deployment options for rolling out the security services. We concluded by noting that the solutions proposed were very well received by the developers, who considered them easy to learn and to apply.

**Acknowledgements.** The work described in this paper has been supported by the European Commission through the IST Programme under Contract IST-2001-32446 (CASENET).

## References

1. ITU-T Recommendation Z.100 (11/99), *Specification and Description Language (SDL)*, Geneva, 1999.
2. ITU-T Recommendation Z.120 (11/99), *Message Sequence Charts (MSC-2000)*, Geneva,1999.
3. Jurjëns, J., *Towards development of secure systems using UMLsec*, Lecture notes in Computer Science 2029, 2001.
4. Lopez, J., Ortega, J.J. and Troya, J.M., *Protocol Engineering Applied to Formal Analysis of Security Systems*, Infrasec'02, LNCS 2437, Bristol, UK, October 2002.
5. Lopez ,J., Ortega, J.J. and Troya, J.M., *Verification of authentication protocols using SDL-Method*, Workshop of Information Security, Ciudad-Real- SPAIN, April 2002.
6. Meadows, C., *Open issues in formal methods for cryptographic protocol analysis*, Proceedings of DISCEX 2000,pages 237-250. IEEE Comp. Society Press, 2000.
7. Menezes, A., Van Oorschot, P.C., Vanstone, S., *Handbook of Applied Cryptography*, CRC Press, 1996.
8. Millen, J. and Denker, G., *CAPSL integrated protocol environment*, In DARPA Information Survivability Conference (DISCEX 2000), IEEE Computer Society, 2000.
9. Millen. J. Denker, G. *CAPSL and MuCAPSL*, J. Telecommunications and Information Technology, 2002
10. Object Management Group. http://www.omg.org/
11. Ryan, P. and Schneider, S., *The Modelling and Analysis of Security Protocols: the CSP Approach*, Addison-Wesley, 2001.