Integrating a Security Requirement Language with UML

H. Abie[1], D. B. Aredo[1], T. Kristoffersen[1], S. Mazaher[1] and T. Raguin[2]

[1] Norwegian Computing Center
P. O. Box 114 Blindern, N-0314 Oslo, Norway
[2] NetUnion sarl, Avenue de Villamont,
19-1005 Lausanne, Switzerland

Abstract. We present an approach that integrates a language for precise and high-level specification of application security requirements, the Security Requirement Language (SRL), with an existing modeling technique, namely, the Unified Modeling Language (UML). SRL is based on first-order logic extended with a small set of modal operators and a syntactic abstraction mechanism. It offers extensibility in that new application/domain-specific requirements can be defined and reused. The focus of SRL is the security of communication in distributed systems. The integrated framework enables developers to add to system models security requirements, such as confidentiality, non-repudiation, and authentication, at an early stage of development, making security an integral part of the system development process. We illustrate the practical usability of our approach by presenting an example, and discuss the experiences that the users of our approach, i.e., system developers, have reported.

1 Introduction

E-work systems, be it for commerce, government, learning, etc., have made their way into our everyday lives, and therefore, we are more vulnerable to the malfunctioning of these systems than ever before. One of the important aspects common to all these systems is that they are security-critical. Security attacks against, e.g., e-commerce systems, have already caused huge financial damage, and much confidential information has been compromised.

One of the major reasons behind the failure of many critical systems is that security mechanisms are added to them as afterthoughts and not integrated into them in the early phases of the development process. Moreover, it is seldom checked, if at all, whether the security mechanisms used indeed satisfy the security requirements of the systems. It is necessary to capture security requirements at an early stage and integrate the requirements into system specification and propagate them further to the design and implementation phases. Lastly, security analysis methodologies must be used to ensure that the provided security mechanisms satisfy the specified security requirements.

This paper presents the results of our work in the EU IST project CASENET [6], whose overall objective was to develop and implement a tool-supported integrated framework and methodology for the formal and systematic specification of security requirements, modeling and analysis of security protocols, and implementation of security-critical e-work systems. The focus of the work described is the *specification* of security requirements based on formal methods and its integration with the Unified Modeling Language (UML) [16]. UML is widely used by the software community for modeling software applications. UML is effective in modeling systems, and its graphical notation is intuitive to users. In contrast, formal methods are not so user-friendly but offer a well-defined semantics that enables them to precisely capture security requirements, thus paving the way for formal verification. Systematic integration of mathematically-based methodologies with semi-formal analysis and design techniques into a single development framework bridges the gap between the practical application of security requirements engineering and the formal methods used in design and analysis of security protocols. Such an integration is shown to be an efficient approach to formal development of critical systems as it pulls together the strengths of the mathematical foundation of formal methods and the user friendliness of UML and exploits their synergy effect [18].

To specify security requirements in a precise way, we have designed the Security Requirement Language (SRL) [2] based on first-order logic. There have been several goals guiding the design of SRL:

– When dealing with security requirements, our concern is to capture just the *what* - specification of the requirements - and not the *how* - mechanism for realization of the requirements.
– Security requirements specifications must be useful in the development of security critical systems, i.e., it must be possible to translate/refine the specified requirements automatically to input used in the formal design/analysis of security protocols. This makes it possible to show that the system indeed provides the specified requirements.
– Most users involved in specifying system requirements and functionalities are not familiar with mathematical notations and concepts underlying formal specification languages, and hence are reluctant to use them. A third goal is therefore to be able to specify security requirements in a way that is clear and easy to understand for end-users.

The first two goals have been addressed by the formal nature of SRL. First-order logic both allows the precise specification of the *what* and lends itself to an automated translation/refinement for use in the formal design/analysis of security protocols, i.e., the *how*. That is, requirements can be translated into goals/constraints to be satisfied in the formal design/analysis of security protocols. We have addressed the third goal by providing abstraction mechanisms that hide the mathematical complexity of the specification behind concepts familiar to the end-users, and by providing a methodology to integrate the language with UML.

The rest of the paper is organized as follows. In Sect. 2, we briefly introduce SRL. Section 3 discusses our methodology for integrating SRL with UML. In Sect. 4, we present an example from a real world problem - a *Document Approval*

Workflow for Public Administration, to illustrate the practical usability of our approach. In Sect. 5, other related approaches are discussed. Section 6 gives a summary of the work described and discusses future work.

2 The Security Requirement Language (SRL)

SRL is based on first-order logic extended with a small set of relations and modal operators. Its focus has so far been on security of communication in distributed systems. For a detailed description of the language the reader is referred to [2].

From the point of view of the specification, at any given time, the world consists of a set of *objects*, each object being an entity of a *type*, such as principal (people, computers, systems) or message. Some objects are *constant* and have a fixed value, while objects of type *variable* may change *value* over time.

SRL's primitives are comprised of:

a set of **logical connectives**, such as the usual propositional connectives in first-order logic $\wedge$, $\vee$, $\neg$, and $\rightarrow$, logic quantifiers $\forall$ and $\exists$, and equality operators $=$ and $\neq$.

the **relations** $\mathrm{Writes}(P, m)$ and $\mathrm{Reads}(P, m)$, to convey the sending and the receiving of messages m by principal P, respectively.

the **functions** $Binding(P, X)$ that returns the value of X in the context of principal P (if P does not know the value of X it returns the null object, ϵ), and $Values(X)$ that returns the set of values that it is possible for an object X to be bound to.

the **modal operators** believes and can_prove. Intuitively, P believes s means that P believes that s is true, and P can_prove s means that P believes that s is true and is able to prove s, i.e., it has a proof of the truth of s, denoted $proof(s)$, that it can present whenever necessary.

a **macro facility** that allows for the definition of a shorthand for a formula, namely, a *macro*. A macro has a name and a body, and it may take typed parameters. In this way, mathematical formulae can be hidden behind concepts familiar to users.

the **epoch** construct that allows for the definition of a period of time during which given security requirements apply to the interactions that take place during the defined period. The related **within** construct is used to specify the interactions that may occur during the epoch.

the **sequence** construct that allows for the modeling of the application in terms of its interactions.

a concept of **time** that allows to express a particular point in time in which a particular relation holds. For example, if P reads a message, m, at time t_1, we write $\mathrm{Reads}_{t_1}(P, m)$. This makes it possible to place temporal constraints on relations.

The Macro Facility. The following example clarifies the concept of *macro*:

$$\text{MessageAuthentication}(A : principal, B : principal, m : message) \equiv$$
$$\text{Reads}_{t_2}(B, m) \rightarrow B \textsf{ believes } (\text{Writes}_{t_1}(A, m))) \wedge t_1 < t_2$$

introduces a macro by the name of MessageAuthentication, taking three parameters: the first two being principals and the last one a message. It expresses the requirement that m be authenticated for principal B. In other words, when receiving m, B knows it comes from A.

Macros are expanded by replacing their formal parameters with the given arguments. For example, the macro,

$$\text{MessageAuthentication}(Manager, Employee, CallForMeeting)$$

expands to

$$\text{Reads}_{t_2}(Employee, CallForMeeting) \rightarrow$$
$$Employee \textsf{ believes } (\text{Writes}_{t_1}(Manager, CallForMeeting))) \wedge t_1 < t_2$$

where $CallForMeeting$ is the message sent (an object in SRL).

A library of macros, called the *standard* library, is included in SRL. This library contains, among others, macros corresponding to common security requirements such as confidentiality, message authentication, non-repudiation, etc. Note that users can define their own macros, customized to their needs, and put them in a library, which can then be used in addition to or instead of the standard library. Libraries also facilitate reuse of requirements.

The Epoch Construct. An epoch is a period of time during which a given security requirement is in effect. It has associated with it an establishment phase and a termination phase that define the sequences of actions that establish and terminate the epoch, respectively. These phases delimit the duration of the epoch but are not part of the epoch themselves. The security requirement that is associated with an epoch applies to all interactions occurring during the corresponding period of time.

The interactions that may occur during a given epoch are specified using SRL's *within* construct which refers to the relevant epoch by its name.

The Sequence Construct. All the SRL primitives presented so far have to do with the specification of security requirements. The context of the security requirements is the application and especially the interactions to which they apply. To model the application, SRL provides the *sequence* construct that allows to group together related interactions of the application in a temporal order. A sequence can have a *name* and can be marked, using *within*, to take place within a given epoch.

3 The Integration Methodology

3.1 Design Decisions

To integrate SRL with UML, we identified constructs and concepts in the two worlds that best fit together and based the integration methodology on those.

Security requirements expressed in SRL are requirements on the interactions between the different entities in a system, and the *sequence* construct is used to model those interactions. Among the different types of UML diagrams, it is the sequence diagram that depicts these interactions and their temporal order explicitly; it is therefore the most appropriate type of diagram, with respect to SRL, for specifying security requirements.

On the SRL side, it is the macro facility that provides end-users with familiar terminology, which hides the mathematical formulae. To specify a requirement, the user only needs to know the name of the corresponding macro and its parameters; the body of the macro is of no concern to him. The arguments passed to the macro are taken from the context of the requirement, which is the application whose security requirements are being specified. The arguments will therefore be entities taken from the UML model of the application. Our integration methodology is therefore primarily based on combining SRL macros with UML sequence diagrams.

UML does not directly offer any constructs for the specification of security requirements, but it offers extension mechanisms that enable addition of new kinds of modeling elements. We wanted a flexible and uniform way to attach security-related properties, from requirements (by means of macros) to epoch definitions, to some of the elements of UML. The *Tagged-Value* extension mechanism was selected because of the flexibility it offers, especially for the values of the tags. In brief, SRL macros representing different security requirements, and other security-related information are attached to different elements of UML using the UML *Tagged-Value* mechanism.

Definition of epochs with their related *within* constructs, names for the sequence diagrams, definition of new macros, and security related assumptions, such as trust, comprise the other information that may be necessary when specifying the security requirements of a system. A special tag by the name CASENET is used for the purpose of supplying security information. Depending on their nature, the different types of information must be conveyed by tags on different elements of a UML model.

3.2 Applying the Methodology

Figure 1 illustrates the integration framework. In SRL security requirements

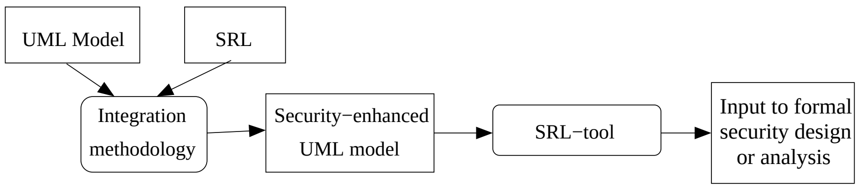

UML Model
SRL
Integration methodology
Security−enhanced UML model
SRL−tool
Input to formal security design or analysis

Fig. 1. The Integration Framework

apply to the interactions of a system. In UML, the corresponding parameterized macros are assigned to the CASENET tags of the appropriate interactions in sequence diagrams. That is, if an interaction in a sequence diagram has the requirement of *message authentication*, a CASENET tag whose value is the corresponding macro is used for that interaction. Note that the parameters of the SRL's `Message_Authentication` macro are replaced with values from the context of the macro, i.e., the corresponding interaction. Figure 2 illustrates the use of SRL macros in UML sequence diagrams where the comment box visualizes the value of the CASENET tag, and the dashed line indicates to which element it is attached (the tags themselves do not have a graphical notation). As for the

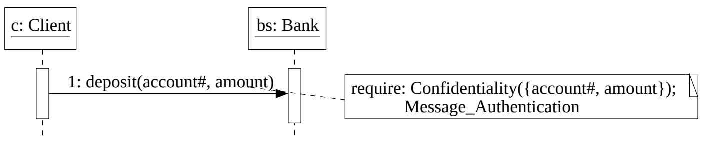

c: Client
bs: Bank
1: deposit(account#, amount)
require: Confidentiality({account#, amount});
Message_Authentication

Fig. 2. Use of SRL Macros in UML

other types of information mentioned above, some apply to a sequence diagram as a whole, such as a name for a sequence diagram or the *within* construct, or to the system as a whole, such as the definition of an epoch or that of a new macro.

The natural place for the sequence diagram-related and the system-related information is the sequence diagram and the system model (within which all the diagrams are contained), respectively. But tagged-values are neither supported for UML sequence diagrams nor for the system model. We therefore had to make some compromises and these types of information were attached to other elements than where they belong.

For sequence diagram-related information, the use of the corresponding *collaboration* was considered; but, since the corresponding *collaboration* may contain several sequence diagrams, the information pertaining to a specific sequence diagram cannot therefore be used as the tag value for the *collaboration.* A completely different option was to use UML comment boxes instead of the *Tagged-Value* mechanism, but UML comments are not always carried through to the output generated by the UML tools, such as `.xmi` files. We therefore have opted for a less than elegant solution, i.e., to attach the information to one of the object elements of a sequence diagram.

Similarly, the system-related information is attached to one of the object elements of a sequence diagram. The rationale behind this choice was to make it easier for end-users by limiting the number of places for tags. Since that kind of information cannot be placed in its natural place, we chose to gather all types of security-related properties in one place such that the end-user can concentrate only on sequences diagrams. Figure 3 illustrates how a sequence diagram-related

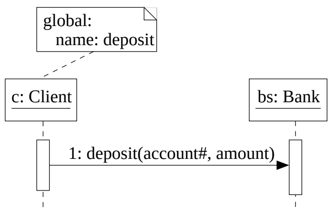


Fig. 3. An Example of Defining Sequence-related Information in UML

information is specified in UML. Some of these problems, namely, the naming of sequence diagrams, will be solved by the upcoming UML2.0 [17] specification, but this is only one of the types of information needed at the sequence diagram level. We believe that UML should support tags at the diagram and model levels, as illustrated by our work on SRL.

SRL is extensible in that, in addition to the predefined macros, new macros can always be defined for requirements specific to a system, as part of the specification of the system in SRL. To define new macros, a deeper knowledge of the language is required. But, once defined and tested, the macros can be reused thereafter. This extensibility is conveyed through the described integration methodology to the UML model. That is, new macro definitions can be tag-values and be used as requirements wherever needed.

Note that all requirements in comment boxes are valid SRL statements. A detailed syntax for the possible values of the CASENET tag is defined and can be found in [5].

3.3 Tool Support

The integration methodology described in the previous section, is supported by the SRL-tool. The input to the tool is the `.xmi` file generated by a UML tool from an application's model augmented with security requirements by means of tags. So far, our tool only supports the `.xmi` files generated by the PoseidonCE1.6.1 tool [3]. Figure 4 shows the SRL-tool, consisting of two components, and its input and output. Both components of the tool are implemented in Java. The front-end component uses an XML parser to extract the necessary information about both the sequence diagrams and the security requirements to generate a specification in SRL. The SRL file is input to the back-end of the tool, basically a compiler, to generate input for a given formal security design/analysis methodology. The code-generation part of the compiler must be rewritten for each new target formalism.

The SRL specification can set the constraints for the formal design or selection of appropriate security protocols for the application, or it can serve as the basis for specifying the goals of formal security protocol analysis methodologies.

[3] PoseidonCE1.6.1 is a product from Gentleware: http://www.gentleware.com

That is, SRL specifications can be translated to formal notations used in design or analysis processes of the security protocols. In the context of our work, the back-end of the SRL-tool translates SRL specifications to a notation used by the formal design methodology used in CASENET. Consequently, the process

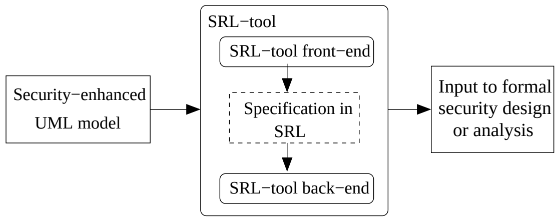

SRL−tool
SRL−tool front−end
Security−enhanced UML model
Specification in SRL
Input to formal security design or analysis
SRL−tool back−end

Fig. 4. The SRL-tool

from using SRL in connection with UML for the specification of the security requirements of an application, to generating input for a formal design/analysis process is supported by tools. This makes it possible to ensure that the security requirements are indeed satisfied by the implementation of the system.

4 Case Study: a Document Approval Workflow

4.1 Description of the System

As a proof of concept, our methodology has been applied to parts of the applications of the user-partners of the CASENET consortium. This section reports on one such trial where the example used is taken from a case study that NetUnion conducted for an online contracting application for public administration. The approach described in Sect. 3 is applied, by one of the members of NetUnion's design and development team, to the part of the process where an end-user digitally signs a previously submitted document.

4.2 Process Description

Once the to-be-signed document is stored on the server, the end-user can proceed with the *signDocument* process

The signing itself has always to take place on the end-user side where any appropriate security device for signing (SmartCard, software certificate, etc.) can be accessed. This implies the transmission of confidential data (the document to be signed and the created signature) over the Internet. The sequence diagrams shown in Fig. 5 illustrate the *signDocument* process. The user logs on to the server (5a), retrieves the document to be signed, does the signing, returns the signature to the server (5c), and logs off (5b).

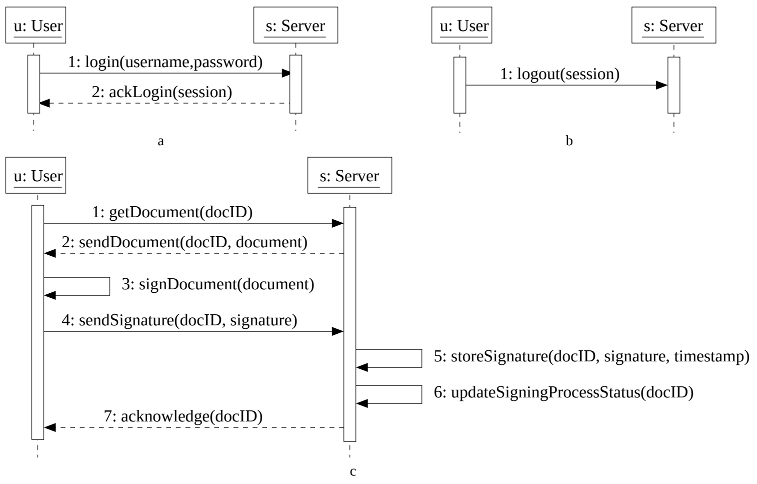

u: User
s: Server
1: login(username,password)
2: ackLogin(session)
a
u: User
s: Server
1: logout(session)
b
u: User
s: Server
1: getDocument(docID)
2: sendDocument(docID, document)
3: signDocument(document)
4: sendSignature(docID, signature)
5: storeSignature(docID, signature, timestamp)
6: updateSigningProcessStatus(docID)
7: acknowledge(docID)
c

Fig. 5. Sequence Diagrams of the Signing Process

4.3 The Identified Security Requirements

NetUnion identified the following security requirements for the signing process:

1. authenticity of the user u to the server s;
2. authenticity of the server s to the user u;
3. authenticity of all messages sent by user u to the server s;
4. confidentiality of *password*;
5. confidentiality of *session*;
6. confidentiality of *document*;
7. confidentiality of *signature*;
8. non-repudiation of receipt for the *signature*.

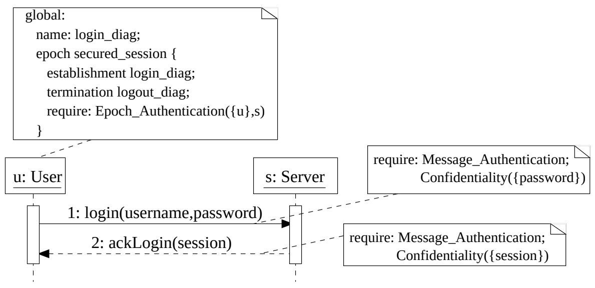
4.4 Specification of the Security Requirements

The security requirements for the signing process being identified, the next step was to specify them on the sequence diagrams of Fig. 5. The appropriate macros were used as values of CASENET tags and attached to the relevant interactions by NetUnion.

The first two requirements on the list in the previous section imply that the messages exchanged in the login phase, depicted in Fig. 6, must be authenticated. This is specified by means of the *Message_Authentication* macro as the tag-value for both messages, namely, *login* and *ackLogin*, of *login_diag* as shown in Fig. 6. Furthermore, the confidentiality of *password* and *session* were required, the fourth and fifth requirements on the list above. This is achieved by using the *Confidentiality* macro as shown in Fig. 6.

The third requirement on the list above can be specified by means of SRL's epoch construct. We define an epoch, called the *secured_session*, which is established by the *login_diag* (Fig. 6) and terminated by the *logout_diag* sequence diagrams (Fig. 7), respectively. The predefined macro *Epoch_authentication*[4] expresses the desired requirement that all messages sent by u must be authenticated by s. This requirement applies to all interactions between u and s taking place in the period of time defined by the corresponding epoch, i.e., between login and logout. Note that the requirement does not apply to the interactions in the *login_diag* and *logout_diag* sequence diagrams.

Now that we have defined the epoch *secured_session*, we have to say what are the interactions that may occur within that epoch. This is done by means of SRL's *within* construct, which is used in the *signDocument* sequence diagram, as depicted in Fig. 8. The *within* construct expresses the fact that all of the interactions in the sequence diagram are permitted to occur within the epoch *secured-session.* Therefore, the epoch requirement, *Epoch_Authentication*, applies to all of those interactions. Fig. 8.



global:
name: login_diag;
epoch secured_session {
establishment login_diag;
termination logout_diag;
require: Epoch_Authentication({u},s)
}
u: User
s: Server
require: Message_Authentication;
Confidentiality({password})
1: login(username,password)
2: ackLogin(session)
require: Message_Authentication;
Confidentiality({session})

Fig. 6. Sequence Diagram of the Login Process with its Security Requirements

As mentioned earlier, the requirement of the epoch *secured_session* does not apply to the interactions of the *logout_diag* sequence diagram. Therefore, in order to completely specify the third requirement in the list, *Message_Authentication* is explicitly required for the *logout* message. Confidentiality of *session* being one of the identified requirements, the corresponding macro is also applied to that message.

In addition, the other identified requirements, namely,

- confidentiality of the *signature* and the *document*, and
- non-repudiation of receipt for the *signature*,

are explicitly specified by means of the corresponding predefined macros for the relevant interactions.

[4] This macro is part of the standard macro library that is defined for SRL and can be used only within an epoch definition.

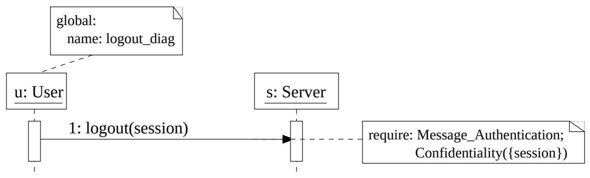


Fig. 7. Sequence Diagram of the Logout Process with its Security Requirements

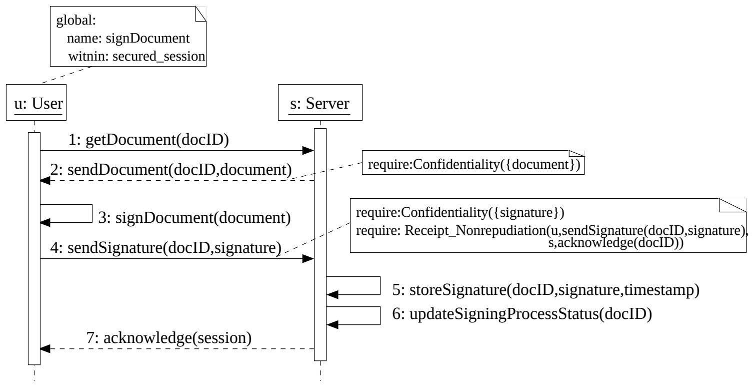

global:
name: signDocument
witnin: secured_session
u: User
s: Server
1: getDocument(docID)
2: sendDocument(docID,document)
require:Confidentiality({document})
3: signDocument(document)
4: sendSignature(docID,signature)
require:Confidentiality({signature})
require: Receipt_Nonrepudiation(u,sendSignature(docID,signature),
s,acknowledge(docID))
5: storeSignature(docID,signature,timestamp)
6: updateSigningProcessStatus(docID)
7: acknowledge(session)

Fig. 8. Sequence Diagram of the Signing Process with its Security Requirements

An excerpt of the `.xmi` file generated by the Poseidon tool for these sequence diagrams and the corresponding class diagram is shown in Fig. 9. The different CASENET tags appear as instances of the `UML:TaggedValue` element. One such instance is shown in Fig. 9, and has the requirements for the *logout* message as its *dataValue*. The SRL specification generated by the SRL-tool for the example is presented in Fig. 10. This specification is further transformed by the back-end of the SRL-tool to the design formalism used in the CASENET project.

4.5 Experience with the Integration Methodology

The system designers at NetUnion reported that SRL is easy to use. Since SRL is combined with a software engineering standard, specifying security requirements for a complete application can be done quickly. SRL macros are easy to understand and to use; the syntax is clear and simple, and they cover a wide range of security requirements. In addition, it is possible to extend the language by creating new macros. When using the predefined macros, SRL only requires a basic knowledge of security engineering and therefore the learning curve is extremely low. For defining new requirements however a thorough knowledge of the language is necessary and some security expertise is needed.

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<XMI xmi.version = '1.2' xmlns:UML = .... >
  <XMI.header> ... </XMI.header>
  <XMI.content>
    ...
    <UML:CallAction xmi.id = 'a21' name = 'logout(session)' ... >
      ...
    </UML:CallAction>
    <UML:Stimulus xmi.id = 'a23' name = 's1' isSpecification = 'false'>
      <UML:ModelElement.taggedValue>
        <UML:TaggedValue xmi.id = 'a25' ... dataValue =
         'require: Message_Authentication; Confidentiality({session})'>
          ...
        </UML:TaggedValue>
      </UML:ModelElement.taggedValue>
      ...
      <UML:Stimulus.dispatchAction>
        <UML:CallAction xmi.idref = 'a21'/>
      </UML:Stimulus.dispatchAction>
    </UML:Stimulus>
    ...
  </XMI.content>
</XMI>
```

Fig. 9. Excerpt of the xmi File.

5 Related Work

To put our work in context, we give some background information and a brief overview of related work.

Work done on formalization of security has been mainly concerned with the formal specification of security protocols for the purpose of analysis. A few examples are [3], [19], [14], [4], [1], [11] and [7]. Protocols are specified in some formal notation, which are then input either directly or after undergoing some transformation to a suitable analysis methodology. Their correctness is established with respect to some goals or invariants defined by the specification.

Several of these efforts are based on modal logic, of which [4], better known as BAN logic, is perhaps the most widely known. SRL has some operators, e.g., *believes*, that are close to the ones used in BAN logic. It therefore should be investigated whether BAN logic is a particularly suitable target formalism for SRL.

None of the work cited above deals directly with the specification of security requirements at a high, abstract level suited for application modeling. Integrating security engineering into the software development process is of paramount importance [8], [15]. Instead of an after-thought, security requirements must be an integral part of the requirements of the system to be built. Work has been done on capturing the security aspects of a system when modeling software by using graphical notations, such as the Unified Modeling Language (UML) [16]. Some of the major efforts in this direction are secureUML [10], AutoFocus [20], and UMLsec [9].

These approaches are similar to ours in the sense that they all introduce security related elements (concepts) into existing graphical modeling notations.

```
import(standard);
environment {
  u, s: principal;
}
epoch secured_session() {
  establishment (u,s) {
    u -> s: login(username,password);
      require: MessageAuthentication(u,s,login(username,password));
      require: Confidentiality(u,s,{password});
    s -> u: ackLogin(session);
      require: MessageAuthentication(s,u,ackLogin(session));
      require: Confidentiality({session});
  }
  termination (u,s) {
    u -> s: logout(session);
      require: MessageAuthentication(u,s,logout(session));
      require: Confidentiality(u,s,{session});
  }
  EpochAuthentication({u},s);
}
sequence signDocument(within secured_session) {
  u -> s: getDocument(docID);
  s -> u: sendDocument(docID,document);
    require: Confidentiality(s,u,{document});
  u -> s: sendSignature(docID,signature);
    require: Confidentiality({signature});
    require: ReceiptNonrepudiation(s,sendSignature(docID,signature),
                                   u,acknowledge(docID));
  s -> u: acknowledge(docID);
}
```

Fig. 10. The SRL Specification of the Example

The major difference is that our security related elements are based on a formal language with a well-defined semantics. This enables the automated refinement of the requirements for use in formal security design/analysis methodologies. secureUML is concerned only with access control requirements. AutoFocus and UMLsec mainly deal with confidentiality and authentication requirements. AutoFocus uses its structure diagrams, the equivalent of UML collaboration diagrams, to introduce security requirements. In UMLsec, different UML diagrams are used to capture security relevant information using extension mechanisms such as *Tagged-Value* and *stereotype*. A major limitation of these two approaches is that each deals with a limited set of security requirements and that it is not possible for the user to introduce new requirements. In contrast, SRL covers a wide range of requirements, and allows new application-specific requirements to be defined and reused; this capability is extended to UML through the integration methodology.

6 Conclusion

We have designed a language based on first-order logic to express systems' security requirements. A methodology for its integration with UML is also defined. We have shown the practical usability of our approach by presenting a real world example and discussing user experiences.

SRL and its integration with UML have a number of advantages:

- Security requirements can be specified precisely and at a high level of abstraction, independently of the implementation mechanisms.
- SRL is extensible in that it allows experts to define new, application/domain-specific security requirements, which can be reused by end-users.
- The library of predefined security macros makes it possible to easily reuse security requirements.
- The formal nature of the language with its well-defined semantics makes it possible to transform a high-level security requirement specification systematically and automatically into input for formal security design/analysis methodology. This permits to verify whether the security requirements are satisfied by the implementation of the system. It also paves the way for automatic generation of executable security-preserving code.
- The integration of SRL into UML exploits the complementary properties of these languages: formality and usability.
- The integration of SRL into UML encourages focusing on security requirements at an early stage, raising developer's awareness of security.
- The integration of SRL into UML makes possible the automated processing of security requirements from an early stage of system development.

In our future work, we plan to make SRL easier to use for end-users and apply SRL to other types of security requirements.

As an example, one way to improve the user-friendliness of SRL would be to make it possible to specify global requirements, such as to express that some data, e.g., *pinCode*, is always to be kept confidential. This would express that whenever *pinCode* is used in an communication, then there is a confidentiality requirement for *pinCode*. This is not possible in the current version of SRL. In an extended, future version of SRL, the obvious place for such a requirement in the UML model of a system would be the class diagram; the security requirement will be given as a tag-value for the attribute in question.

As mentioned earlier, the focus of SRL has been to specify security requirements related to communication. We will continue our work by using SRL to express security requirements for other contexts, such as access management, and by including the resulting new concepts into the integration methodology described in this paper.

Acknowledgments

This work has been partially funded by the European Commission through the CASENET project, IST-2001-32446, in the Fifth Program Framework.

References

1. Abadi, M., Gordon, A. D.: A Calculus for Cryptographic Protocols, The Spi Calculus. Research Report, digital Systems Research Center, January 1998.

2. Aredo, D. B., Kristoffersen, T., Mazaher, S.: Abstract Security Requirement Specification. Technical Report DART/03/04, Norsk Regnesentral, Oslo, Norway, March, 2004.
3. Bieber, P.: A Logic of Communication in a Hostile environment. *Proceedings of the Computer Security Foundations Workshop III.* IEEE Computer Society Press. June, 1990.
4. Burrows, M., Abadi, M., Needham, R.: A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1), February 1990.
5. CASENET, IST project 2001-32446: User Trial Progress Report. Deliverable CASENET/WP5/D5.2, June, 2003.
6. CASENET, IST project IST-2001-32446: *http://www.casenet-eu.org/*,
7. Denker, G., Millen, J., Rues, H.: The CAPSL Integrated Protocol Environment. Technical Report SRI-CSL-2000-02, SRI International, Computer Science Laboratory, October 2000.
8. Higginbotham, M. D., Maley, J. G., Milheizler, A. J., Suskie, B. j.: Integrating Information Security Engineering with System Engineering with System Engineering Tools. *Proceedings of WETICE '98*, July, 1998.
9. Jurjens, J.: UMLsec: Extending UML for Secure Systems Development. *Proceedings of the 5th International Conference on the United Modeling Language* (UML 2002). Lecture Notes in Computer Science, vol. 2460. Springer Verlag, 2002.
10. Lodderstedt, T., Basin, D., Doser, J.: SecureUML: A UML-based modeling language for model-driven security. *Proceedings of the 5th International Conference on the United Modeling Language* (UML 2002). Lecture Notes in Computer Science, vol. 2460. Springer Verlag, 2002.
11. Lowe, G.: Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. *Proceedings of TACAS 95.* Lecture Notes in Computer Science, vol. 1055. Springer Verlag, 1996.
12. Meadows, C., Syverson, P.: A Formal Language for Cryptographic Protocol Requirements. *Designs, Codes and Cryptography*, 7(1-2), January 1996.
13. Meadows, C.: The NRL Protocol Analyzer: An Overview. *Journal of Logic Programming*, 26(2), 1996.
14. Moser, L.: A Logic of Knowledge and Belief for Reasoning about Computer Security. *Proceedings of the Computer Security Foundations Workshop II.* IEEE Computer Society Press, June, 1989.
15. Mouratidis, H., Giorgini, P., Manson, G.: Integrating Security and Systems Engineering: Towards the Modeling of Secure Information Systems. *Proceedings of the 15th Conference on Advanced Information System Engineering* (CAISE*03), 2003.
16. OMG: The Unified Modeling Language Specification V1.5., Object Management Group, Needham, MA, U.S.A.", March, 2003.
17. OMG: *http://www.omg.org/technology/documents/modeling_spec_catalog.htm*
18. Shroff, M., France, R.: Towards a Formalization of UML Class Structures in Z. *Proceedings of COMPSAC'97*, August, 1997.
19. Syverson, P.: Formal Semantics of Logics of Cryptographic Protocols, *Proceedings of the Computer Security Foundations Workshop III.* IEEE Computer Society Press, June, 1990.
20. Wimmel, G., Wißpeintner, A.: Extended Description Techniques for Security Engineering. In M. Dupuy and P. Paradinas, editors, *Trusted Information, The New Decade Challenge*, Proceedings of the IFIP 16th International Conference on Information Security (Sec'01). Kluwer Academic Publishers, 2001.