Security Constraints for Sequence Diagram and Code Generation

Abdeslam Jakimi[1,2] and Mohammed Elkoutbi[1]

[1]Mohammed V University, SIME, ENSIAS, Rabat, Morocco ,
[2]My Ismail University, FSTE, B.P 509, Boutalamin, Errachidia, Morocco
{ajakim@yahoo.fr}

***Abstract*:** The Unified Modeling Language, which has become a standard notation for object-oriented modeling, provides a suitable framework for scenario acquisition using use case diagrams and sequence diagrams. A sequence diagrams shows the interactions among the objects participating in a scenario in temporal order. It depicts the objects by their lifelines and shows the messages they exchange in time sequence. In this paper, we suggest to offer the extension of scenarios that describe a given system in a natural way based directly on sequence diagrams. We developed algorithm and tool support that can automatically produce a code of sequence diagram with security constraints.

Keywords: UML, sequence diagrams, security constraints, code generation.

1. Introduction

The Unified Modeling Language (UML) [1,2,3] is an expressive language that can be used for problem conceptualization, software system specification as well as implementation. UML is a graphical language for specifying the analysis and design of object-oriented software systems [2]. It provides several diagram types that can be used to view and model the software system from different perspectives and/or at different levels of abstraction. UML defines thirteen types of graphical diagrams. The four diagrams which become important in the design phase are a class diagram, use case diagram, state chart diagram and sequence diagram.

The emergence of UML as a standard for modeling systems has encouraged the use of automated software tools that facilitate the development process from analysis through coding. In UML, the static structure of classes in a system is represented by a class diagram while the dynamic behavior of the classes is represented by a set of usecase, sequence and statechart diagrams. To facilitate the software development process, it would be ideal to have tools that automatically generate or help to generate executable code from the models.

In the present study, an effort has been made to find methods to automatically generate executable security code from the UML sequence diagram. We propose to extend the UML with the following message constraints: security constraints.

2. Scenarios and UML

2.1. Scenarios

Scenarios have been evolved according to several aspects, and their interpretation seems to depend on the context of use and the way in which they were acquired or generated. In a survey, Rolland [4] proposed a framework for the classification of scenarios according to four aspects: the form, contents, the goal and the cycle of development

The form view deals with the expression mode of a scenario. Are scenarios formally or informally described, in a static, animated or interactive form?

The contents view concerns the kind of knowledge which is expressed in a scenario. Scenarios can, for instance, focus on the description of the system functionality or they can describe a broader view in which the functionality is embedded into a larger business process with various stakeholders and resources bound to it.

The purpose view is used to capture the role that a scenario is aiming to play in the requirement's engineering process. Describing the functionality of a system, exploring design alternatives or explaining drawbacks or inefficiencies of a system are examples of roles that can be assigned to a scenario.

The lifecycle view considers scenarios as artefacts existing and evolving in time through the execution of operations during the requirement's engineering process. Creation, refinement or deletion are examples of such operations.

2.2. Scenarios in UML

Object-oriented analysis and design methods offer a good framework for scenarios. In our work, we have adopted the UML, which is a unified notation for Object-oriented analysis and design. It directly unifies the methods of Booch, Rumbaugh and Jacobson.

Scenarios and use cases have been used interchangeably in several works meaning partial descriptions. UML distinguishes between these terms and gives them a more precise definition. A use case is a generic description of an entire transaction involving several objects of the system. A use case diagram is more concerned with the interaction between the system and actors (objects outside the system that interact directly with it). It presents a collection of use cases and their corresponding external actors. A scenario shows a particular series of interactions among objects in a single execution of a use case of a system (execution instance of a use case). A scenario is defined as an instance of a given use case. Scenarios can be viewed in two different ways through sequence diagrams or communication diagrams. Both types of diagrams rely on the same underlying semantics. Conversion from one to the other is possible.

2.2.1 Class diagram

A class diagram is a graphic view of the static structural model. It shows a set of classes, interfaces and their relationships. The main focus is on the description of the classes. Class diagrams are important for constructing systems through forward engineering. The ClassD is the central diagram of a UML model. The translation of class diagrams to an Object-oriented programming language is easy and provided by most CASE tools.

2.2.2 Sequence diagram

For our work, we chose to use sequence diagrams because of their wide use in different domains. A sequence diagram shows interactions among a set of objects in temporal order, which is good for understanding timing and interaction issues. It depicts the objects by their lifelines and shows the messages they exchange in time sequence. However, it does not capture the associations among the objects.

A sequence diagrams has two dimensions: the vertical dimension represents the time, and the horizontal dimension represents the objects. Messages are shown as horizontal solid arrows from the lifeline of the object sender to the lifeline of the object receiver (Figure 2). A message may be guarded by a condition, annotated by iteration or concurrency information, and/or constrained by an expression. Each message can be labeled by a sequence number representing the nested procedural calling sequence throughout the scenario, and the message signature. Sequence numbers contain a list of sequence elements separated by dots.

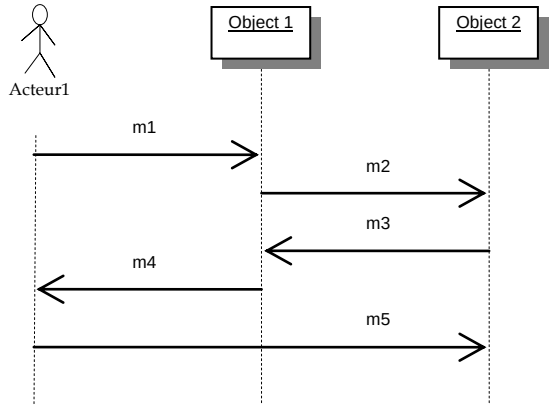

Figure 2. Example of a SequenceD.

2.2.3 Constraints for sequence diagram

UML defines two standard constraints for messages: vote and broadcast. The vote constraint restricts a collection of return messages, and the broadcast constraint specifies that the constrained messages are not invoked in any particular order.

Beyond the UML standard message constraints found in sequence diagrams, elkoutbi et al. [5,6] define the two additional constraints input Data and output Data. The input Data constraint indicates that the corresponding message holds input information from the user. The outputData constraint specifies that the corresponding message carries information for display. Both input Data and output Data constraints have a parameter that indicates the kind of user action. This parameter normally represents the dependency between the message and the elements of the underlying class diagram.

3. Description of the Approach

In this section, we describe the overall approach to add security constraints and generate code from scenarios (sequence diagram). The approach consists of three activities (see Figure 3), which are detailed below:

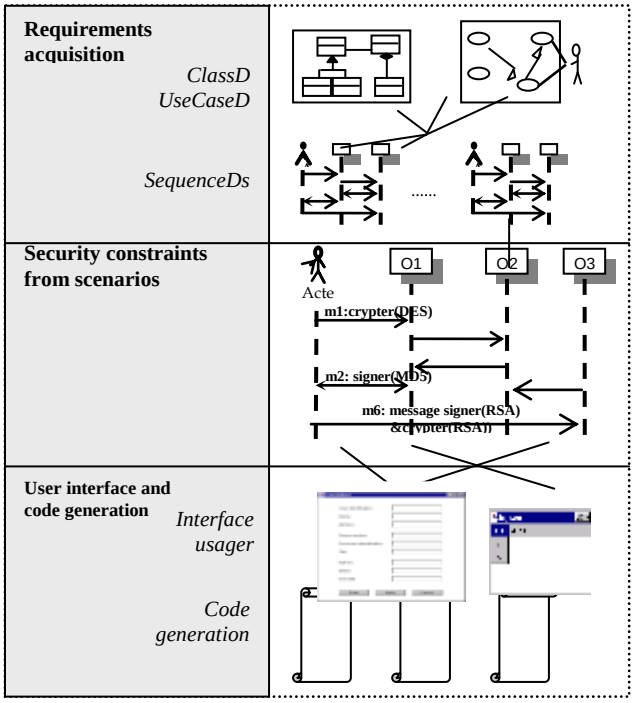

Figure 3. Overview of the proposed process

In the *Requirements Acquisition* activity, the analyst elaborates the UsecaseD, and for each use case, he or she elaborates several SequenceDs corresponding to the scenarios of the use case at hand.

The *Security Constraints activity* consists of extending interaction diagrams. We propose to extend the UML with the following message constraints: security constraints.

In the *User interface prototype and code generation activity*, we generate code from a sequence diagram and derive user interface prototypes for all the interface objects found in the system.

In the following, we will discuss these activities of the proposed process.

3.1 Security constraints for sequence diagram

Today, security has become a major issue for information systems (e-business, e-trade, etc). It will be convenient to be able to define and represent these constraints in the step of requirement engineering. We were interested in the major security aspects: authenticity and confidentiality. Authenticity means the proof of identity and confidentiality relates to the privacy of information. Using UML, when a message is sent from a source to a target object, it can carry some information (message parameters). We aim to express that the exchange is private using some encryption algorithms (RSA, AES, 3DES, etc). This can be specified as a parameter of the constraint. The two constraints defined to model

security aspects are given below:
m{Auth}: The message m must be signed by the sender object to proof its identity to the receiver object.
m{Crypt(algo)}: The message content (message parameters) must be encrypted using the algorithm (algo).
Figures 4 give an example a sequence diagram with security constraints for ATM (Automatic Teller Machine)

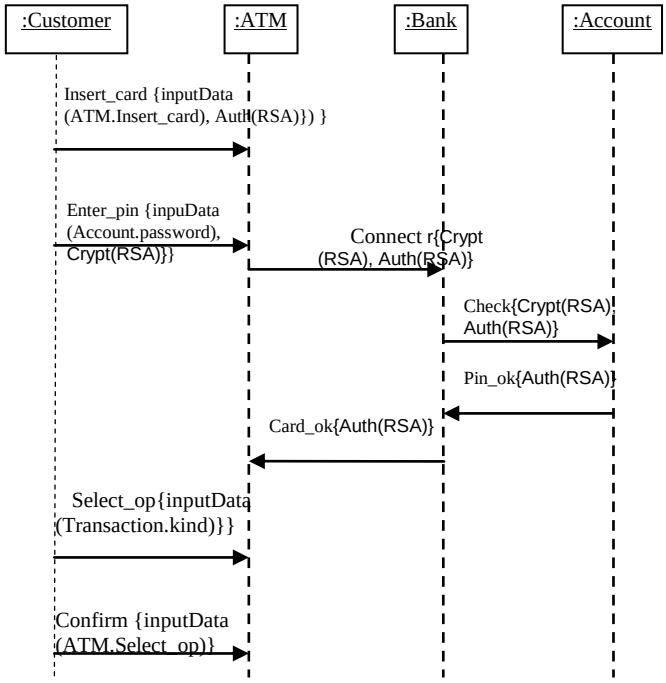

Figure 4. Sequence diagram with security constraints

3.2 Code generation

UML and Java [7], which is an OO model and design notation and an OO programming language respectively, are some of the tools widely used in many software development projects. However, these modeling and programming activities are mostly separated. And a gap exists between these models and programs.
This paper proposes an approach to narrow the gap between multiple UML models and an implemented system. The narrowing of a gap is achieved by generating Java source code directly from multiple UML model diagrams. The code generation is achieved by creating a mapping between UML and the Java programming language.

Many current OO CASE tools [8, 9, 10, 11, 12,13] generate limited skeleton code from such models. The main drawback of this approach is that there is no code generation for object behavior and thus the code generated is not complete. Chow et al. [14] developed two main steps in translating code from dynamic behaviour of the system. Translate an object's state diagram into Java code and Generate method body based on the pre/post condition of an operation and specify the order of language statements based on the message passing sequence in the interaction diagram. Jakimi et al. [15] generated automatically implementation code from the UML sequence diagrams in an object-oriented programming language such as Java.
The generation of Java code from the UML sequence diagram has been met with some degree of success. In addition to the generation of skeleton class code from class diagram, Java codes have been generated from the statechart, the sequence diagram and the component diagram. However, generation of Java source code from all UML diagrams is not yet achievable.
Figure 5 presents the types of the security constraints which we can associate an exchange of messages between the objects.

- Simple message
- Encrypted message
- Encrypted and signed message
- Signed message

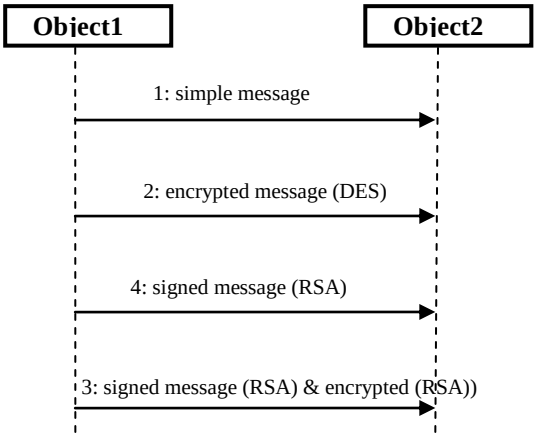

Figure 5. Sequence diagram with type's security constraints

The code generated relating to the diagram of figure 5 is arising according to the type of message sent (simple, signed, encrypted or signed and encrypted). The code generated by this approach for the figure 6 is:

```
class mySystem {
        public void service(){
    // simple message
          Object2.m1();
  //encrypted  messageé
     Object2.m2(encrypted(DES/RSA));
 // signed message
          Object2.m3(signed(MD5/RSA));
  //signed/encrypted message
           Object2.m3(signeg(RSA) & encrypted(RSA)));
       }
}
```

Figure 6. Code generation from figure 5

3.3 Tool support

For create the tool support for code generated from the sequence diagram with security constraints. We have used the Eclipse environment, the TogetherJ plug-in for UML modeling and the application programming interface (API) JDOM for XML manipulation.
We used the plug-in for UML diagrams (from Together) which makes it possible for us to create sequence diagrams. This diagram is first acquired to throw the UML diagram plug-in, and then there is transformed in form XML files.

This XML file can also be imported via the UML diagram plug-in for purposes of visualization and annotation. Finally we develop a code generator for automatic Java code security generation from sequence diagram.

4. Conclusion

In conclusion, we have proposed in this paper an UML-based code generation approach. In this work, we have presented a new approach that produces automatically code from the sequence diagram with security constraints. This approach helps developers to transit from the design to implementation phase and to shorten the software development cycle.

The future works of this research include the following areas: generate code from UML diagrams that describe dynamic and non-functional aspects of a system while remaining platform independent and optimize generated code, find a rigorous method to lower the abstraction level.

References

[1] Object Management Group (OMG), Unified Modeling Language (UML) specifications version 1.5, 2003. http://www.omg.org/

[2] G. Booch, J. Rumbaugh, and I. Jacobson, "The Unified Modeling Language: User Guide", Massachusetts: Addison-Wesley, 1999.

[3] J. Rumbaugh, I. Jacobson, and G. Booch, "The Unified Modeling Language: Reference Manual Guide", Massachusetts: Addison-Wesley, 1999.

[4] C. Rolland, C. Ben Achour, C. Cauvet, J. Ralyté, A. Sutcliffe, N.A.M. Maiden, M. Jarke, P. Haumer, K. Pohl, E. Dubois and P. Heymans. "A Proposal for a Scenario Classification Framework". The Requirements Engineering Journal, Volume 3, Number 1, 1998.

[5] M. Bennani., M. Elkoutbi., and K. Nafil; Modelling Real-time Aspects using UML Scenarios proceedings of the 3rd International Conference on Software Methodologies, Tools and Techniques, pp. 200-213, Leipzig, Germany, September 2004.

[6] M. Elkoutbi, Khriss I., R.K. Keller. "Automated Prototyping of User Interfaces Based on UML Scenarios". The Automated Software Engineering Journal, 13, 5-40, 2006.

[7] Sun Microsystems Inc., Java Technology, http://java.sun.com

[8] J. Ali, and J. Tanaka, "Converting Statecharts into Java Code", in Proc. IDPT'00, Dallas, Texas, USA, 2000.

[9] I-Logix Inc., Rhapsody, http://www.ilogix.com.

[10] D. Harel, and E. Grey, "Executable Object Modeling with Statecharts", in Proc. of 18th Inter. Conf. on Software Engineering, IEEE, March 1996, pp. 246-257.

[11] D. Harel, and E. Grey, "Executable Object Modeling with Statecharts",Computer, vol. 30, no. 7, 1997, pp. 31-42.

[12] I. Azim Niaz and J. Tanaka, An Object-Oriented Approach To Generate Java Code From UML Statecharts, International Journal of Computer & Information Science, Vol. 6, No. 2, June 2005

[13] J. Ali, and J. Tanaka, "Implementing the Dynamic Behavior Represented as Multiple State Diagrams and Activity Diagrams", Journal of Computer Science & Information Management , vol. 2, no. 1, 2001, pp.

[14] K.O. Chow, W. Jia, V.C.P. Chan and J. Cao, Modelbased generation of Java code, Proc. International Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), Las Vegas, USA, 2000.

[15] A. Jakimi and M. El Koutbi, "An Object-Oriented Approach to UML Scenarios Engineering and Code Generation" International Journal of Computer Theory and Engineering , vol.1, No 1, pp 35-41, April 2009.

Author Biographies

Abdeslam Jakimi is a professor at Faculty of Science and technology in myIsmail University, he received his Masters degree in software engineering in 2004. His current research interests include requirements engineering, user interface prototyping, design transformations, scenario engineering and code generation.

Mohammed Elkoutbi is a professor at National School of Computer and Systems Analysis in Agdal, Rabat, Morocco. His current research interests include requirements engineering, user interface prototyping and design, and formal methods in analysis and design. He earned a PhD in Computer Science from University of Montreal in 2000.