The Guided System Development Framework

Carvalho Quaresma, Jose Nuno; Probst, Christian W.; Nielson, Flemming

Published in:
Proceedings of the 23rd Nordic Workshop Programming Theory

Publication date:
2011

The Guided System Development Framework

Jose Quaresma Christian W. Probst Flemming Nielson
Technical University of Denmark
{jncq,probst,nielson}@imm.dtu.dk

1 Introduction

The Service-Oriented Computing paradigm has had significant influence on the Internet, where an increasing number of companies are making their services available. It is now very common to develop systems by specifying the interaction between programs that provide a specific functionality in form of a service, which can be seen as a function that other programs can remotely execute. With the emergence of this paradigm, it is important to provide tools that help system designers to specify the system under development, to enable its easy integration with standard security suites used by industry, to generate code that implements the system's functionality, and to verify the security properties of those systems.

In fact, formal methods provide powerful tools that can be used to verify the security of these systems. However, these tools are not always integrated with the used development environment. That shortcoming is aggravated by the lack of expertise usually necessary to use formal methods and interpret their results. This obstacle can be overcome by a framework that aids and guides the developer on the specification of the system being developed, on choosing the appropriate standard protocols suites that achieve the required security properties, on providing an implementation of the specified system, and also on allowing the verification of its security properties.

In this article, we present the Guided System Development (GSD) framework that provides those functionalities and seamlessly integrates them. The initial idea behind this framework is strongly inspired by CaPiTo [3] and its main contribution: the successful connection of the abstract specification of Service-Oriented Systems with the usage of industry standard suites with the verification of the protocol and generation of code. CaPiTo uses the LySatool [1] to perform the verification of the system and generates system implementations in the programming language C.

2 The Framework

As mentioned above, one of the main ideas behind this framework is to apply CaPiTo's separation of concerns regarding the modelling of communication protocols — especially the separation between the message exchange view and the usage of standard communication suites used in industry — in a simple and intuitive way. This separation of concerns is an essential part of the GSD framework, which enables the specification, implementation and verification of Service-Oriented Systems by having different levels of abstraction and using a functional language similar to the Alice and Bob Notation [5].

As illustrated in Figure 1, the framework can be divided in two phases, the Specification Phase and the Realisation Phase. The former is composed by the Abstract Level and the Standards Level, while the latter is related to the different outputs and translations from the Standards Level.

2.1 The Specification Phase

As mentioned above, the Specification Phase is composed by the two top levels, the Abstract Level and the Standards Level.

In the Abstract Level, the system is described in a language similar to the Alice and Bob notation, with some extensions, such as Receiver Side Actions and Security Modules. Receiver Side Action allow

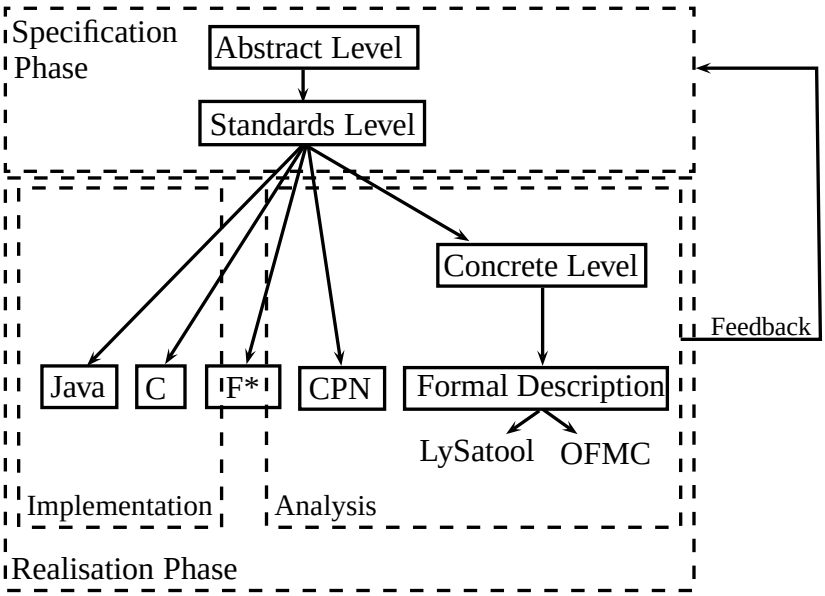

Figure 1: Overview of the Guided System Development framework including some examples of targeted languages and tools

the developer to specify additional actions that will be performed by the receiver of the message, for example, comparison and assignment of values. We believe this is an important extension of standard approaches, because it allows a more complete description of the system. The other main extension is the use of Security Modules, which specify the security requirements of the transmission of some elements of the message exchange. Such requirements are, for example, confidentiality, authenticity, and security. Using these, the developer is able to specify the goals of the system in an early stage of development. These goals, comparable to the ones is BAN logic [2], enable the reasoning about security properties of the described system.

The Abstract Level description of a system is translated into the Standards Level. This translation is performed by identifying Standard Suites that achieve the requirements expressed by the Security Modules. This can be realised by giving suggestions to the developer, depending on the Security Module used in the system description.

2.2 The Realisation Phase

The main goal of the Realisation Phase of the GSD framework is twofold; on one hand, it enables the translation of the system specification into different platforms such as Java, C, or CP-nets. On the other hand, it connects to verification tools, in order to reason about the security properties of the system.

By providing a translation from the system specification into executable code, the developer can easily obtain an implementation of the system, similar to earlier work [7].

Targeting Coloured Petri Nets [4] would add an alternative way of describing the system. CP-nets is a graphical oriented modelling language for the design, specification, simulation and verification of systems. It extends the original Petri nets with the capabilities of the high-level language, allowing the definition of data types — that can be seen as coloured tokens — and the manipulation of data values.

As for the reasoning about the security properties of the system, it can be achieved by translating the system to the Concrete Level — which represents the full message exchange — and verifying its security properties with tools such as LySatool [1] and OFMC [6]. The analyses results allow us to provide some

instant feedback to the system developer regarding the security of the system under development.

Another possibility that we are currently investigating is the translation into F* [8], a dependently typed language for secure distributed programming. This approach would allow to combine executable code with verification of security properties.

2.3 Example

In Figure 2, we show a simple message exchange between Alice and Bob to illustrate the functioning of the framework. We show it for the two levels that constitute the core of the framework and also for the Concrete Level.

While several Standard Suites could be used to implement the goal required in the Abstract Level, we are using TLS, an extremely wide-spread suite, to achieve confidentiality of the message. Confidentiality here means that the receiver (B) is authenticated and that the message can only be seen by the sender (A) and the receiver (B) in the modelled message exchange. The Concrete Level represents the full message exchange. We assume that Alice knows the public key of the certificate authority (CA) prior to the message exchange. Furthermore, both Alice and Bob calculate the resulting symmetric keys sk_{AB} and sk_{BA} (one for each direction of the communication) based on the values of n, m, N. Also, $Finished$ is generated based on these values together with the messages previously exchanged by Alice and Bob. For simplicity reasons, we are not showing the receiver side action and derivation of extra elements.

$$A \rightarrow B : confidential(M)$$

Abstract Level

$$A \rightarrow B : TLS(M)$$

Standards Level

$$\begin{aligned}
A &\rightarrow B : fresh(n) \\
B &\rightarrow A : fresh(m), \{B, pk_B^+\} : pk_{CA}^- \\
A &\rightarrow B : \{fresh(N)\} : pk_B^+, \ \{Finished\} : sk_{AB} \\
B &\rightarrow A : \{Finished\} : sk_{BA} \\
A &\rightarrow B : \{M\} : sk_{AB}
\end{aligned}$$

Concrete Level

Figure 2: Example of a system with a unique message that sends message M in a confidential way

3 Conclusion

With this framework, we provide the developer with an environment that allows a high-level specification of a system (and its goals), the use of Standard Suites to achieve those goals, and the interconnection to different runtime environments and formal approaches. The latter support verification of security properties, the result of which can be used to improve the system specification.

The next step in our research will be to define and implement the core of the framework and then investigate which of the alternatives are more suitable regarding the Concrete Level of the framework. We are currently defining the semantics of the language in the Abstract Level and the translation from that level to the Standards Level.

References

[1] Mikael Buchholtz. *User's Guide for the LySatool version 2.01*. DTU, April 2005.

[2] Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8:18–36, February 1990.

[3] H. Gao, F. Nielson, and H.R. Nielson. Protocol Stacks for Services. In *Foundations of computer security*, 2009.

[4] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9(3-4):213–254, 2007.

[5] S. Modersheim. Algebraic Properties in Alice and Bob Notation. In *Availability, Reliability and Security, 2009. ARES '09. International Conference on*, pages 433 –440, 2009.

[6] Sebastian Mödersheim and Luca Viganò. The open-source fixed-point model checker for symbolic analysis of security protocols. In *Foundations of Security Analysis and Design V*, volume 5705 of *Lecture Notes in Computer Science*, pages 166–194. Springer Berlin / Heidelberg, 2009.

[7] Jose Quaresma and Christian W. Probst. Protocol implementation generator. *Nordic Conference in Secure IT Systems (NordSec 2010)*, 2010.

[8] Nikhil Swamy, Juan Chen, Cedric Fournet, Pierre-Yves Strub, Karthikeyan Bharagavan, and Jean Yang. Secure distributed programming with value-dependent types. *The 16th ACM SIGPLAN International Conference on Functional Programming (ICFP 2011), to appear*, September 2011.