CoPS - Checker of Persistent Security*

Carla Piazza, Enrico Pivato, and Sabina Rossi

Dipartimento di Informatica, Università Ca' Foscari di Venezia,
{piazza,epivato,srossi}@dsi.unive.it

Abstract. CoPS is an automatic checker of multilevel system security properties. CoPS can be used to check three different bisimulation-based non-interference properties for systems expressed as terms of the Security Process Algebra (SPA) language. The considered properties are persistent, in the sense that they are preserved at each execution step. Moreover, they imply the Bisimulation-based Non Deducibility on Composition (*BNDC*) property, whose decidability is still an open problem.

1 Introduction

The tool CoPS, available at `http://www.dsi.unive.it/~mefisto/CoPS/`, is an automatic checker of multilevel system security properties. It implements the polynomial algorithms described in [1] to check three security properties named

- *SBNDC*, i.e., *Strong Bisimulation-based Non Deducibility on Composition*,
- *P_BNDC*, i.e., *Persistent BNDC*,
- *PP_BNDC*, i.e., *Progressing Persistent BNDC*.

These are *Non-Interference* [8] properties for processes expressed as terms of the *Security Process Algebra* (SPA) [5] which is a variation of Milner's CCS [10] with actions partitioned into security levels. They imply the *Bisimulation-based Non Deducibility on Composition* (*BNDC*) property, whose decidability is still an open problem. If a system E satisfies one of the three properties checked by CoPS, then what a low level user sees of the system is not modified (in the sense of the bisimulation semantics) by composing any high level (possibly malicious) process with E, i.e., high level users cannot send confidential information down to low level users. The properties are *persistent* in the sense that if a process is *SBNDC* (resp., *P_BNDC* and *PP_BNDC*), then every reachable state is still *SBNDC* (resp., *P_BNDC* and *PP_BNDC*). As far as *P_BNDC* is concerned, in [7] persistency has been proved to be fundamental to deal with processes in dynamic contexts, i.e., contexts that can be reconfigured at runtime. Moreover, in [2] it is shown how *P_BNDC* can be used to prove properties (e.g., fairness) of cryptographic protocols. The three properties are compositional with respect to

* Partially supported by the MIUR Project "Modelli formali per la sicurezza", the EU Contract IST-2001-32617 "MyThS", and the FIRB project (RBAU018RCZ) "Interpretazione astratta e model checking per la verifica di sistemi embedded".

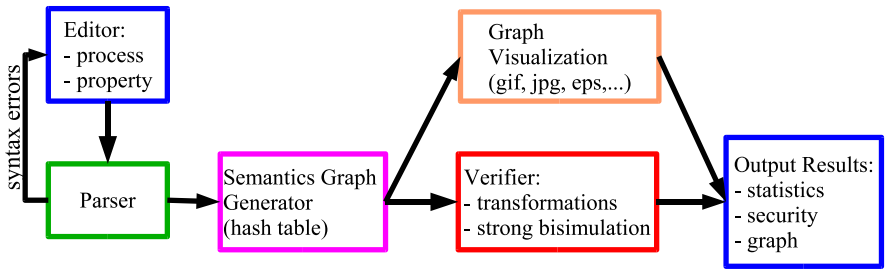

Fig. 1. CoPS Architecture

the parallel composition operator. CoPS exploits this compositionality to speed up the computation and drastically reduce the space complexity.

CoPS consists of a *graphical interface* and a *kernel module*. The graphical interface has been implemented in JAVA to get a large portability and allows to:

- *Insert the process(es)* to be checked in the editor pane. The process(es) can be either typed or loaded from a file. A tree is automatically drawn to facilitate the navigation among processes. The syntax is highlighted to get a better readability. Both fonts and colors can be changed by the user.
- *Select the security property* to be checked and start the verification. It is also possible to check whether two processes are strongly or weakly bisimilar.
- *Read the verification results.* Some time/space statistics are shown together with the security result. Moreover, syntax errors are reported.
- *View the graph* representing the semantics of the process(es). This can be also saved in a file whose type (e.g., jpg, gif, eps) can be chosen by the user.

The kernel module, whose architecture is shown in Figure 1, has been implemented in standard C to obtain good performances and consists of:

- A *parser* which checks for syntax errors and builds the syntax tree out of the SPA process.
- A *semantics graph generator* which elaborates the syntax tree to generate an adjacency-list representation of the graph associated to the process.
- A *verifier* which transforms the graph in order to use a strong bisimulation algorithm to perform the security check.

2 Persistent Security Properties

The *Security Process Algebra* (SPA) [5] is a variation of Milner's CCS [10], where the set of visible actions is partitioned into high level actions and low level ones in order to specify multilevel systems. The syntax of SPA *processes* is as follows:

$$E ::= \mathbf{0} \mid a.E \mid E + E \mid E|E \mid E \setminus v \mid E[f] \mid Z$$

The semantics is the same as in CCS. In particular, as in CCS, we denote by τ the silent (invisible) action.

As an example, a binary memory cell which initially contains the value 0 and is accessible by both high and low level users through the read and write operations (e.g., $r_h 0$ represents the high read of 0) can be formalized as follows:

$$M0 = \overline{r_h 0} \,.\, M0 + w_h 0 \,.\, M0 + w_h 1 \,.\, M1 + \overline{r_l 0} \,.\, M0 + w_l 0 \,.\, M0 + w_l 1 \,.\, M1$$
$$M1 = \overline{r_h 1} \,.\, M1 + w_h 0 \,.\, M0 + w_h 1 \,.\, M1 + \overline{r_l 1} \,.\, M1 + w_l 0 \,.\, M0 + w_l 1 \,.\, M1$$

$M0$ and $M1$ are totally insecure processes: no access control is implemented and a high level malicious entity may write confidential information into the memory cell which can be then read by any low level user. Our security properties will aim at detecting this kind of flaws, even in more subtle and interesting situations.

The three security properties *SBNDC*, *P_BNDC* and *PP_BNDC* can be defined in terms of unwinding conditions: if a state F of a secure process performs a high level action moving to a state G, then F also performs a sequence of silent actions moving to a state K which is equivalent to G for a low level user. We denote by $(\stackrel{\tau}{\rightarrow})^*$ a sequence of zero or more silent actions, by $(\stackrel{\tau}{\rightarrow})^+$ a sequence of at least one silent action and by $(\stackrel{\tau}{\rightarrow})^0$ a sequence of zero actions. We also use $\approx$ for weak bisimulation (see [10]) and $\approx^p$ for progressing bisimulation (see [11]).

Definition 1 ([1]). *A process E is* SBNDC *(resp.,* P_BNDC *and* PP_BNDC*) if for all F reachable from E, if $F \stackrel{h}{\rightarrow} G$, then $F(\stackrel{\tau}{\rightarrow})^0 K$ (resp., $F(\stackrel{\tau}{\rightarrow})^* K$ and $F(\stackrel{\tau}{\rightarrow})^+ K$) and $G \setminus H \approx K \setminus H$ (resp. $G \setminus H \approx K \setminus H$ and $G \setminus H \approx^p K \setminus H$).*

The memory cell defined above does not satisfy any of the three security properties. In fact, there is a direct information flow from high to low level. We can redefine the cell by eliminating any low level read operation as follows:

$$M0 = \overline{r_h 0} \,.\, M0 + w_h 0 \,.\, M0 + w_h 1 \,.\, M1 + w_l 0 \,.\, M0 + w_l 1 \,.\, M1$$
$$M1 = \overline{r_h 1} \,.\, M1 + w_h 0 \,.\, M0 + w_h 1 \,.\, M1 + w_l 0 \,.\, M0 + w_l 1 \,.\, M1$$

Now the memory cell is both *SBNDC* and *P_BNDC*, but not *PP_BNDC*.

Both *SBNDC* and *P_BNDC* are compositional with respect to the parallel operator, but not with respect to the non-deterministic choice operator. On the other hand, *PP_BNDC* is fully compositional.

In [1] efficient polynomial algorithms to verify the three security properties are described. These algorithms are based on the reduction of the problems of checking the security properties to the problem of checking a strong bisimulation between two graphs. CoPS implements such algorithms. As far as the strong bisimulation underlying algorithm is concerned, CoPS allows the user to choose between the Paige and Tarjan's algorithm [12] and the fast bisimulation algorithm described in [4]. This choice does not affect the worst-case complexities.

3 Tool Overview and Experimental Results

A screen-shot of CoPS is shown in Figure 2: a process has been typed in the edit pane on the right (the syntactic convetions are very similar to the ones

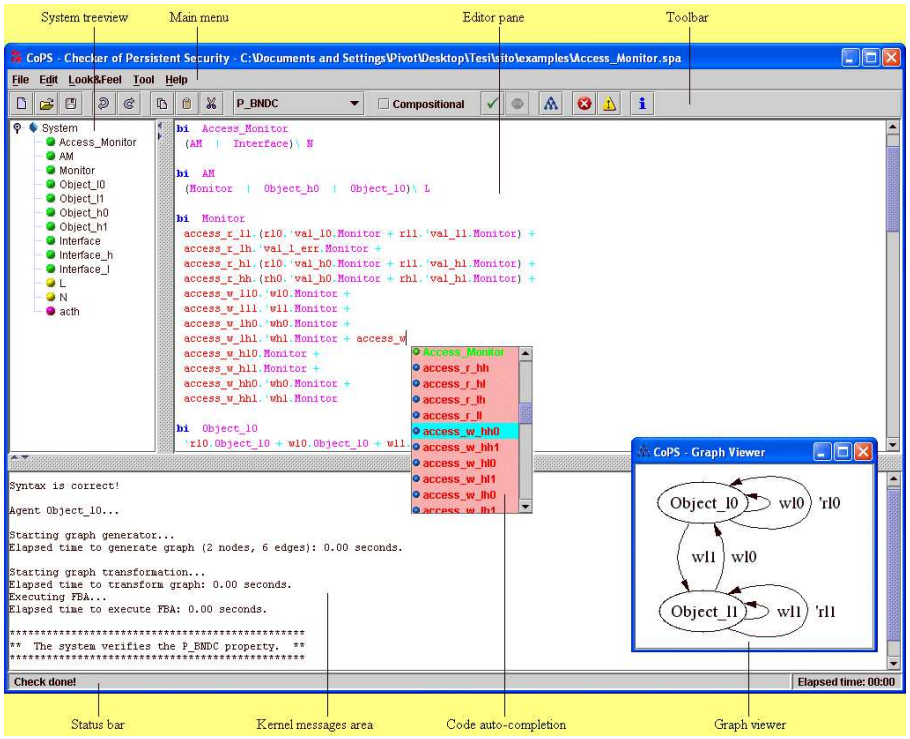

Fig. 2. A screen-shot CoPS : the process is *P_BNDC*

used on CCS processes in the Concurrency Workbench model checker [3]); the sub-processes occurring in its definition are automatically listed on the left; the verification results are shown in the bottom window. By selecting a process on the left, the editor moves on its definition and it allows one to verify it. The toolbar allows one to choose the property to be checked, stop the computation, see the graph representing the semantics of the process. The graph visualization requires the installation of GRAPHVIZ which can be downloaded at `http://www.research.att.com/sw/tools/graphviz/`. The SETTINGS option in the EDIT menu allows one to personalize the kernel execution by, e.g., setting the path of GRAPHVIZ and the format of the generated graph, choosing the bisimulation algorithm to be used (the Paige and Tarjan's one [12] or the one presented in [4]), avoiding the graph generation, setting the use/dimension of an hash table which speeds up the graph generation.

It is possible to avoid the use of the graphical interface and use directly the kernel via command line (`checker --help` shows the help).

CoPS has been successfully used on a number of medium-sized case studies. It has been compared with the tool CoSeC [6], which allows one to check a bisimulation-based property equivalent to *P_BNDC*. The experiments have been

carried out on a PC with a AMD Athlon XP 1800+ processor and 256M RAM. For medium size processes with a number of states smaller than 2000 CoPS takes one third of the time with respect to CoSeC. For processes with a greater number of states (around 6.000) CoPS takes half of the time with respect to CoSeC. We also checked a complex system: the Access_Monitor described in [5]. By exploiting the compositionality of *P_BNDC*, CoPS takes 55 sec while CoSeC didn't produce any answer after 12 hours. Notice that the main differences between CoPS and CoSeC consist of: (1) the use of the Paige and Tarjan algorithm for strong bisimulation [12] instead of the Kannellakis and Smolka's one [9]; (2) exploiting the *P_BNDC* characterization presented in [1] CoPS performs only one strong bisimulation test, while CoSeC repeats the test over all the reachable states.

References

1. A. Bossi, R. Focardi, C. Piazza, and S. Rossi. Verifying Persistent Security Properties. *Computer Languages, Systems and Structures*, 2003. To appear. Available at `http://www.dsi.unive.it/~srossi/cl03.ps.gz`.
2. M. Bugliesi, A. Ceccato, and S. Rossi. Context-Sensitive Equivalences for Non-Interference based Protocol Analysis. In *Proc. of the International Symposium on Fundamentals of Computing (FCT'03)*, volume 2751 of *LNCS*, pages 364–375. Springer–Verlag, 2003.
3. R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: A semantics-based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 15(1):36–72, 1993.
4. A. Dovier, C. Piazza, and A. Policriti. A Fast Bisimulation Algorithm. In *Proc. of Int. Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *LNCS*, pages 79–90. Springer-Verlag, 2001.
5. R. Focardi and R. Gorrieri. A Classification of Security Properties for Process Algebras. *Journal of Computer Security*, 3(1):5–33, 1994/1995.
6. R. Focardi and R. Gorrieri. The Compositional Security Checker: A Tool for the Verification of Information Flow Security Properties. *IEEE Transactions on Software Engineering*, 23(9):550–571, 1997.
7. R. Focardi and S. Rossi. Information Flow Security in Dynamic Contexts. In *Proc. of the 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, pages 307–319. IEEE Computer Society Press, 2002.
8. J. A. Goguen and J. Meseguer. Security Policies and Security Models. In *Proc. of the IEEE Symposium on Security and Privacy (SSP'82)*, pages 11–20. IEEE Computer Society Press, 1982.
9. P. C. Kannellakis and S. A. Smolka. CCS Expressions, Finite State Processes, and Three Problems of Equivalence. *Information and Computation*, 86(1):43–68, 1990.
10. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
11. U. Montanari and V. Sassone. CCS Dynamic Bisimulation is Progressing. In *Proc. of the 16th International Symposium on Mathematical Foundations of Computer Science (MFCS'91)*, volume 520 of *LNCS*, pages 346–356. Springer-Verlag, 1991.
12. R. Paige and R. E. Tarjan. Three Partition Refinement Algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

A Appendix

A.1 CoPS Site

CoPS is freely available at `http://www.dsi.unive.it/~mefisto/CoPS/`. In particular, in the site you can find:

- a short description of CoPS and its features;
- a tutorial which illustrates how to use CoPS;
- installation and configuration instructions;
- the downloadable versions together with a directory of examples;
- some references to theoretical papers on which CoPS is based;
- a form to contact us for any problem/suggestion.

CoPS, which is partially supported by the MIUR Project "Mefisto: Modelli formali per la sicurezza", the EU Contract IST-2001-32617 "MyThS", and the FIRB project (RBAU018RCZ) "Interpretazione astratta e model checking per la verifica di sistemi embedded" has been mainly tested by other participants of these projects on different case studies. Some of these case studies have been included in a directory of examples.

A.2 System Requirements and Installation Instructions

In the web pages of CoPS we put four compiled versions (for Windows, Linux, Sun, and MacOS) and a setup program for Windows.

In order to use CoPS with its graphical interface it is necessary to install the Java Runtime Environment (JRE) version 1.3.1 or above. We recommend the use JRE version 1.4.2, because the previous versions contain a bug which can cause a malfunctioning of CoPS. However, it is possible to use the kernel, named `checker`, via command line (`--help` provides all the details).

To view a graphical representation of the semantics of the system under analysis it is necessary to install Graphviz, which can be freely downloaded at `http://www.research.att.com/sw/tools/graphviz/`. If you are not interested in this feature you can disable the graph generation.

The installation of CoPS only requires the download and decompression of a file containing the compiled kernel and the graphical interface. Windows users can also choose to download a setup program providing a menu icon group in the program menu and an uninstall program. Files with `.spa` extension are automatically associated with CoPS.

The Settings option in the Edit menu allows to change the default settings, such as the Graphviz path, the underlying bisimulation algorithm, the graph generation/format, the use/dimension of an hash table, and others.

More detailed instructions and suggestions can be found in CoPS' site.

A.3 An Illustrating Example

A guided tour concerning the functionalities of CoPS and the use of its graphical interface can be found in the TUTORIAL section of our site. There we briefly recall the syntax of the SPA processes accepted by CoPS and illustrate the meaning of buttons, menus, and settings of the graphical interface through some snapshots.

Here we model a case study in order to give an intuition about the meaning of our security properties and the potentialities of CoPS .

Let us consider the E-commerce Processing System described in "Information Flow in Operating Systems: Eager Formal Methods" by J.D. Guttman, A.L. Herzog, and J.D. Ramsdell, presented at the Workshop on Issues in the Theory of Security 2003 (WITS'03). The system represents a process in which:

- an order is submitted electronically by a `Client`;
- an `E_sale` process ensures that the order is correct (e.g., the prices and discounts are correct), and, if so, passes it to the process `A_receiv` (Account Receivable);
- `A_receiv` interacts with a credit card clearing house and, if everything is ok, passes the order to the `Ship` process;
- the `Ship` process sends the order to the `Client`.

In the paper presented at WITS'03 the authors use Linear Temporal Logic to specify information flow policies for SELINUX, which can then be checked via model-checking. The `E-commerce` example is used to illustrate the technique. In particular, in this example it is important to ensure that, if the internal channels of communication are secure, then the casual chain is always the same (e.g., it is not possible that an unpaid order is shipped).

Let us model the E-commerce Processing System in the SPA language and use CoPS to check that the casual chain remains the same even in presence of a malicious attacker. To do this, all the interactions (including the ones with the client) have to be modelled as high level actions. Since we are assuming that the channels are secure these actions will be under the scope of a restriction, i.e., an attacker cannot synchronize on these actions. Then, different low level signals have to be sent out at different execution points. We have to check that also in presence of an high level attacker the low level signals are sent out in the same order, i.e., the casual chain is always the same. Hence, using the syntax[1] of CoPS we get the following processes.

```
bi E_Commerce (Client|E_sale|A_receiv|Ship)\Hc

bi Client 'sock_price_ok_and_pay_ok.shipped_order.0

bi E_sale sock_price_ok_and_pay_ok.'oklow1.'new_order_pay_ok.E_sale
        + sock_price_ok_and_pay_no.'oklow1.'new_order_pay_no.E_sale
        + sock_price_no_and_pay_ok.'nolow1.E_sale
        + sock_price_no_and_pay_no.'nolow1.E_sale
```

[1] In CoPS given an action `a`, `'a` stands for the output action $\overline{a}$.

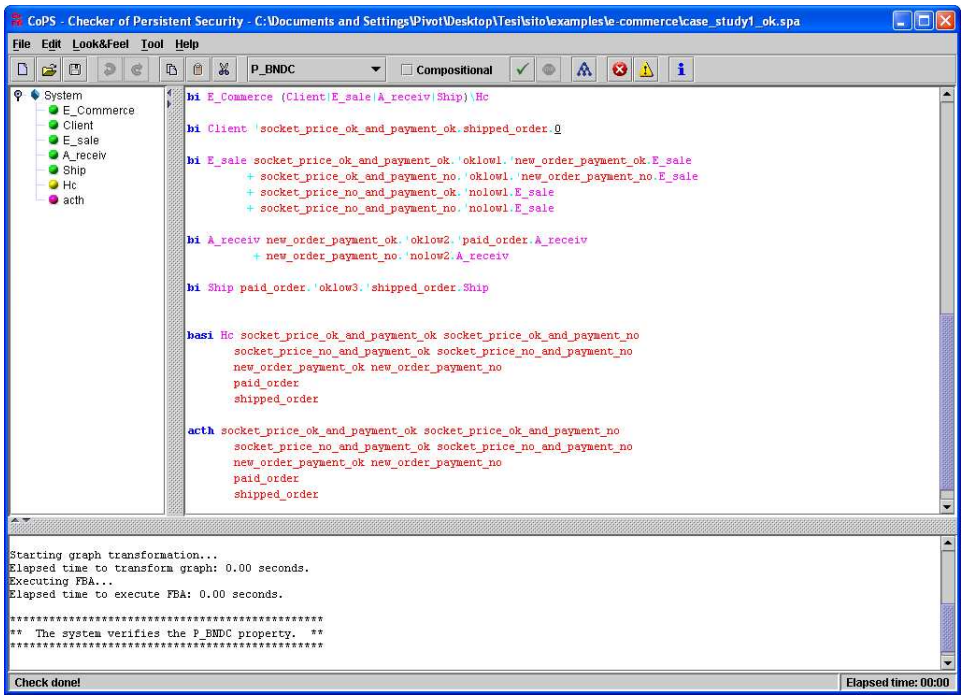

Fig. 3. The process `E_commerce` is *P_BNDC*

```
bi A_receiv new_order_pay_ok.'oklow2.'paid_order.A_receiv
          + new_order_pay_no.'nolow2.A_receiv

bi Ship paid_order.'oklow3.'shipped_order.Ship

basi Hc sock_price_ok_and_pay_ok sock_price_ok_and_pay_no
        sock_price_no_and_pay_ok sock_price_no_and_pay_no
        new_order_pay_ok new_order_pay_no
        paid_order shipped_order

acth sock_price_ok_and_pay_ok sock_price_ok_and_pay_no
     sock_price_no_and_pay_ok sock_price_no_and_pay_no
     new_order_pay_ok new_order_pay_no
     paid_order shipped_order
```

The process `E_commerce` satisfies the three security properties, i.e., the casual chain order is always respected. In Figure 3 we show the positive answer of CoPS relatively to the *P_BNDC* property. Some of its sub-components (e.g., the process `E_sale`) are not secure. This is due to the fact that the high level channels are not locally restricted, i.e., an attacker interacting directly with a

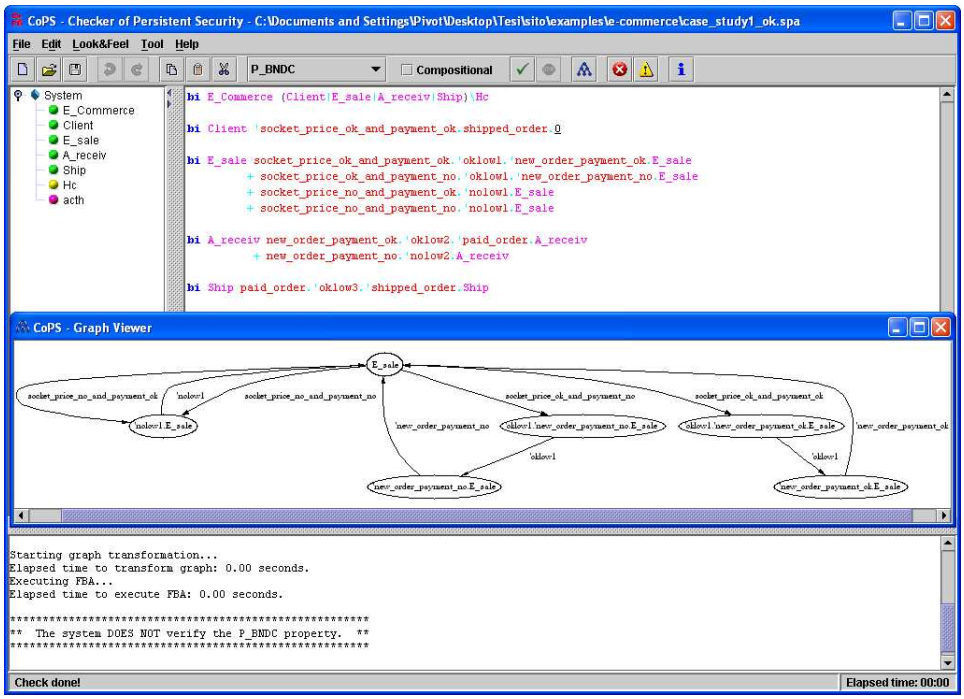

Fig. 4. The process `E_sale` is not *P_BNDC*

sub-component can change the casual chain. In Figure 4 we show the negative answer of CoPS relatively to *P_BNDC* for `E_sale`, together with its graph representation.

This example is modelled in the file `case_study1_ok.spa` in the subdirectory `e-commerce` of the directory of examples downloadable from our site. In the same directory, the files `case_study2_ok_dead.spa` and `case_study2_ok_nodead.spa` contain a variation of the E_commerce process in which the client can query the system to know the status of its order. In this case it is necessary to add timeouts to avoid that an attacker blocks the system.