A Formal Approach to the Integrated Analysis of Security and QoS

Alessandro Aldini *, Marco Bernardo

Università di Urbino "Carlo Bo"
Istituto di Scienze e Tecnologie dell'Informazione
Piazza della Repubblica 13, 61029 Urbino, Italy

Abstract

Concurrent and distributed systems are subject to several requirements of different nature. Among them security and quality of service (QoS) are two fundamental aspects, which can have a profound impact on the system performability. Unfortunately, the study of the tradeoff between security guarantees and performance needs is hard to accomplish, because the related analysis activities are usually carried out separately. In this paper we present an integrated and tool-supported methodology encompassing both activities, which can provide insights about how to trade the QoS delivered by a system with its security guarantees. The methodology is illustrated by assessing the effectiveness and the efficiency of the securing strategy implemented in the NRL Pump, a trusted device proposed to secure the replication of information from a low-security level enclave to a high-security level enclave.

Key words: QoS, security, noninterference, formal methods, process algebra, tools, case studies

1 Introduction

Security is a critical requirement of dependable systems and networks. In particular, in the last decades many efforts in the security community have been done to properly ensure data access control in multilevel secure systems. In essence, in such systems sensitive information is classified into access levels

* Corresponding author.
Email addresses: aldini@sti.uniurb.it (Alessandro Aldini), bernardo@sti.uniurb.it (Marco Bernardo).

and users are assigned clearances, such that users can only access information classified at or below their clearances.

The recent trends to open and real-time computing have shown that following such an approach is not enough to successfully solve the security problem. On the one hand, distributed systems and mobile code increase the vulnerability of network systems to attacks and of confidential data to information leaks. It is well known from real cases that a controlled sharing of information does not protect against direct and indirect illegal information flows. In particular, indirect leaks of data, called covert channels, are hard to be revealed and, even in the case well-established formal techniques can be applied to capture them, in practice many covert channels cannot be eliminated because of the intrinsic nature of the system [MK94,R+01,AG02,ABG03]. The presence of unavoidable covert channels is not only critical with respect to the security problem, because such channels may also indicate the existence of undesired interferences from the environment that are responsible for compromising properties related to safety, reliability, and availability [SD98].

On the other hand, the application of strong strategies aiming at minimizing each unwanted information flow is sometimes made impractical by hard QoS constraints. Therefore, trading QoS with information leaks is of paramount importance in the system design process. This activity involves both security analysis and performance evaluation, two tasks that – unfortunately – are usually carried out separately.

In order to achieve a reasonable balance between the expected QoS and the absence of illegal information flows, in this paper we advocate the adoption of an integrated view of security verification and performance evaluation. This is accomplished by proposing a tool-supported methodology that combines both analyses on the same formal system description. In brief, the integrated methodology we propose works in two phases as follows.

The first phase requires to provide a functional description of the system at hand, to which a security check is applied in order to reveal all the potential nondeterministic covert channels from the high-security level to the low-security level. Such an analysis is based on the noninterference approach to information flow theory [GM82] and is essentially carried out through equivalence checking [FG95]. Diagnostic information provided by the security check can be exploited to remove the unwanted covert channels or to reveal the causes of the unavoidable information leaks. On the other hand, the nature of the information flow captured in this phase can be also useful to detect the impact of the interference causing the information flow on other aspects like, e.g., system safety and availability. Indeed, applications of the noninterference approach outside of the original security setting can provide a framework for verifying properties related to safety, reliability, and availability. As an ex-

ample, the diagnostic information returned by the noninterference check can be useful to analyze the influence of faults triggered by the environment or non-trusted components upon the behavior of system components performing safety-critical applications.

Afterwards, in the second phase of our methodology the bandwidth of the covert channels detected in the first phase can be quantitatively assessed. This is carried out by enriching the functional description of the system with information about the temporal delays and the frequencies of the system activities. The second system description considered in the methodology thus relies on a performance model that can be analyzed through standard numerical techniques or simulation [Ste94,Lav83]. The output of this performance analysis is given by the value of some relevant efficiency measures of the system together with the bandwidth of its covert channels, expressed as the amount of information leaked per unit of time. Such performance figures are then used as a feedback to properly tune the system configuration parameters in a way that lowers the covert channel bandwidth under a tolerable threshold without jeopardizing the QoS delivered by the system.

Although the proposed methodology is independent of the specific description language and companion tool – provided that the basic ingredients needed by the methodology itself are supplied – in this paper the application of the methodology is illustrated using the process-algebraic, performance-oriented, architectural description language Æmilia [BDC02] and a suitably extended version of the related software tool TwoTowers [Ber06] that encompasses security analysis.

In this paper the application of our methodology is exemplified by means of a case study: the Network NRL Pump [KMM98]. This is a trusted device used in multiple single-level security architectures to offer replication of information from low-security level systems (Low, for short) to high-security level systems (High, for short) with high-assurance security guarantees. Data replication is needed in this framework to strengthen both availability and security. On the one hand, data replication is a proven approach for increasing the availability for distributed systems [OSS02]. On the other hand, data replication minimizes multilevel secure accesses to shared resources from processes at different security levels.

Although at first sight illegal information leaks seem to be absent in a message exchange from Low to High, some subtle behaviors must be paid attention to in order to prevent unauthorized users from obtaining access to confidential information. In fact, in order to offer reliable communications, an acknowledgement (ack) is usually required for each message that is successfully received. The transmission of an ack from High to Low is more than enough to set up a covert communication channel if the timing of the ack is under the control of

High. The NRL Pump, which basically acts as a delaying buffer between High and Low, makes such a timed covert channel negligible (see, e.g., the security analysis conducted in [L+04]) with a minor impact on the QoS.

However, some information can still be sent from High to Low through the NRL Pump. This is due to the feedback forwarded by the NRL Pump to notify Low that a connection is up/down. In fact, High can manipulate the notification procedure to set up a 1-bit covert channel related to the connection status. To mitigate the effect of such an unavoidable covert channel, the NRL Pump architecture is designed in such a way that a minimum delay is enforced between connection setup and connection closing/abort and between the connection reestablishment and the auditing of any connection that behaves in a suspicious way.

Here the question is no longer whether the NRL Pump is secure, but how much data per unit of time can be leaked by exploiting the backward information flow. By applying our methodology, we formally verify that such an information leakage is the unique functional covert channel suffered by the NRL Pump. We also emphasize the relation between the output of the security check and properties related to safety and availability that the NRL Pump should satisfy in spite of the interference of non-trusted parties. Afterwards we provide useful information about the tradeoff between the bandwidth of the unavoidable covert channel and the NRL Pump configuration parameters. In particular, we emphasize the impact of the NRL Pump securing strategy both on the QoS delivered by the system, expressed as the number of connection requests that are served per unit of time, and on some dependability-related properties.

The paper, which is an extended and revised version of [AB04a,AB04b], is organized as follows. In Sect. 2 we describe our methodology, which we illustrate throughout the paper by means of the NRL Pump. This mechanism is presented in Sect. 3 through the support of the Æmilia/TwoTowers technology, which is introduced in Sect. 4. In Sect. 5 we apply our methodology to the integrated analysis of the NRL Pump security and QoS. Comparison with related work and some concluding remarks are finally reported in Sect. 6.

2 Predicting the Tradeoff between Security and QoS

Trading QoS aspects with security requirements is a non-trivial operation involving both the description of the system functional behavior and the definition of the performance-related configuration parameters. In order to integrate in a balanced way these different aspects of the system design, we propose a tool-supported methodology that combines well-known formal techniques from

the security analysis and performance evaluation fields. The objective of our methodology is to guide the design of a system that should not significantly suffer from information leaks and that, at the same time, should deliver an adequate performance.

The methodology is illustrated in Fig. 1. As can be seen, the methodology requires to build two formal models of the system – a functional one and a performance one – where the latter is incrementally obtained from the former by adding further details.

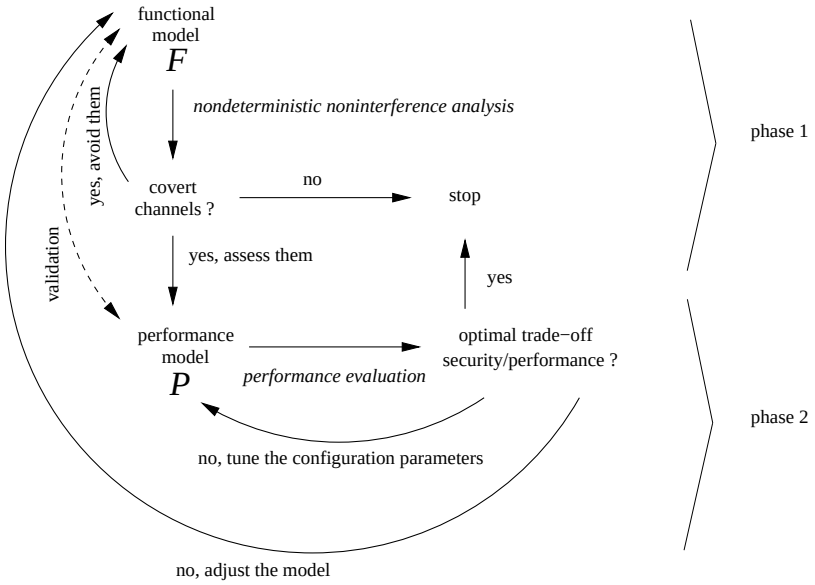

Fig. 1. Models and phases of the integrated methodology

The model in the first phase – the functional model F – refers to the formal description of the functional behavior of the system. A security check is applied to it in order to capture all the potential nondeterministic information leaks. The application of formal methods for the analysis of security properties (see, e.g., [Mea03] and the references therein) is a well-established approach accepted by the security community. In particular, our methodology relies on a technique based on the idea of nondeterministic noninterference [GM82]. Basically, supposing that low-security level users (Low) observe public operations only and high-security level users (High) perform confidential operations only, an interference from High to Low occurs if what High can do is reflected in what Low can observe. For instance, this approach has been formalized in [FG95], where noninterference properties such as strong nondeterministic noninterference and strong nondeducibility on composition are defined in a process algebraic setting. In particular, the first one consists of verifying whether the view of the system behavior as observed by Low in the absence of High interferences is the same as that observed when High interacts with the system. The second one, which is the strongest property described in [FG95], consists of verifying whether the view of the system behavior as observed by Low is

always the same before and after the execution of any interaction between the system and High.

The noninterference approach to security analysis can be reused in other areas in order to analyze, e.g., fault-tolerant systems or strong-partitioning mechanisms [DiV99]. The idea is to ensure that the behavior of the system observable by an external user, represented by Low, is independent of possible negative interferences from the environment or from faulty system components, represented by High. Recent examples (see, e.g., [SWD98,SD98,A+04] and the references therein) emphasize how different aspects of dependability can be handled by noninterference.

Formally, in order to apply the noninterference check, we divide the system activities into high-level and low-level actions, denoted *High* and *Low*, respectively, depending on their nature. Then, on the basis of the specific noninterference property, from the functional model F we derive the models that express the Low views to be compared. For instance, if the property is strong nondeterministic noninterference then we derive two models. On the one hand, the view of F without High operations, denoted $F \backslash High$, is obtained by preventing F from executing high-level actions. On the other hand, the low-level view of F with High interactions, denoted $F/High$, is obtained by turning all the high-level actions into invisible actions, since Low is not expected to observe them. Finally, $F \backslash High$ and $F/High$ are compared through equivalence checking. As shown in [FG95], a notion of equivalence relation that can be used to conduct the comparison is weak bisimulation equivalence [Mil89], as this captures the ability of two processes to simulate each other behaviors up to invisible actions. If the equivalence check is satisfied, then Low cannot infer the behavior of High by observing the public view of the system, which means that the system does not leak information from High to Low.

Whenever in the first phase a covert channel is revealed that leaks information from High to Low, the feedback returned by the equivalence check – typically in the form of a distinguishing modal logic formula – provides diagnostic information about the causes of the information flow. Based on this information, first of all it can be established whether the detected covert channel is avoidable or not. Then, if the covert channel can be eliminated with a minor impact on the functionalities of the system behavior, the same information can be exploited to suitably modify the functional model. In contrast, for all covert channels that are either unavoidable or tolerated as they would require a significant revision of the functional model, it is necessary to move on to the second phase of our methodology, in which the functional model F is refined into a performance model P by specifying the timing of each system activity.

When applying the methodology in practice, the consistency of the performance model P with respect to the functional model F – which ensures the

same noninterference outcome – depends on the precise way in which the functional model is extended with temporal information as well as on the expressive power of the formalism adopted to develop the models. For instance, if all the activity durations are expressed through exponentially distributed random variables, the derived performance model P turns out to be a Markovian model yielding a continuous-time Markov chain. This model does not need to be validated against F, since it is directly obtained from F by attaching exponential rates to the state transitions. In other words, the Markovian model is consistent by construction with the corresponding functional model, in the sense that the state space of the Markovian model is isomorphic (up to the transition rates) to the state space of the corresponding functional model. On the other hand, if the performance model contains general distributions to better characterize the actual system delays, then P may need to be validated against F. In fact, the use of general distributions no longer having infinite support may alter the state space of P with respect to the state space of F. In conclusion, to ensure consistency between the two models, the noninterference analysis must be repeated in the second phase only if some high-level actions are characterized through distributions with finite support.

Once the validation succeeds, the performance model can be analyzed through numerical techniques [Ste94] or simulated via standard techniques [Lav83]. In this phase, a quantitative estimation of the information leakage revealed during the first phase is provided by evaluating the performance metrics that are directly related to the bandwidth of the information flow from High to Low. Similarly, QoS-related metrics can be assessed by analyzing the same performance model. The resulting performance figures should reveal whether a reasonable tradeoff between security – in terms of bandwidth of each covert channel – and QoS – in terms of performance measures like system throughput and response time – is met or not. Such performance figures are then used as a feedback to guide the choice of the configuration parameters that affect the metrics of interest. The objective of this tuning activity is to lower the amount of information leakage under a tolerable threshold without jeopardizing the QoS delivered by the system. In the case a reasonable tradeoff cannot be obtained, it is necessary to adjust the functional model and restart the integrated analysis.

Since the developer may have different design requirements (strict vs. relaxed security needs and loose vs. tight QoS constraints) the adjustment activity can follow opposite strategies. If the main objective is to preserve information confidentiality one can modify the functional behavior of the system until the resulting model suffers only from unavoidable covert channels that are quantitatively negligible. In this respect, note that aiming at perfect security might compromise the service availability, which is one of the most critical factors for the success of network-based applications. Therefore, tuning the configuration parameters is needed to keep QoS as high as possible without

significantly altering the security constraints.

In the opposite limiting scenario, if ensuring QoS is more important than confidentiality issues, one can fix a QoS threshold and then modify the (functional and performance) behavior of the system until the resulting model meets the desired QoS. In such a case, tuning the configuration parameters is needed to keep the bandwidth of the potential covert channels as low as possible without jeopardizing the QoS. In this respect, controlling the kind and the amount of illegal information flows might be necessary to ensure dependability-related properties like, e.g., service availability.

Our methodology is adequate to cope with each possible intermediate scenario, by supporting at each step the kind of adjustment that is needed by the actual requirements.

3 An Overview of the NRL Pump

The NRL Pump is configured as a single hardware device that interfaces a high-security level LAN with a low-security level LAN. In essence, the Pump places a buffer between Low and High, pumps data from Low to High, and probabilistically modulates the timing of the ack from High to Low on the basis of the average transmission delay from High to the Pump.

As shown in Fig. 2, the low-level and high-level enclaves communicate with the Pump through special interfacing software called wrappers, which implement the pump protocol. Each wrapper is made of an application-dependent part, which supports the set of functionalities that satisfy application-specific requirements, and a pump-dependent part, which is a library of routines that implement the pump protocol. Each message that is received and forwarded by the wrappers includes 7 bytes of header field, containing information about the data length, some extra header, and the type of message (data or control).

The Pump should not be considered as a general-purpose network router that accepts a message from a low-level network and routes that message to a high-level network. Such an uncontrolled behavior would cause both security and availability problems. On the availability side, any low-level application may request to connect to any high-level LAN thus wasting the Pump resources. On the security side, a low-level Trojan Horse application could ask high-level Trojan Horse processes to reveal their presence. To avoid these problems, each process that uses the Pump must register its address with the Pump administrator, which is responsible for maintaining a configuration file that contains a connection table with registration information. The Pump provides both recoverable and non-recoverable services. Recoverability safely assumes

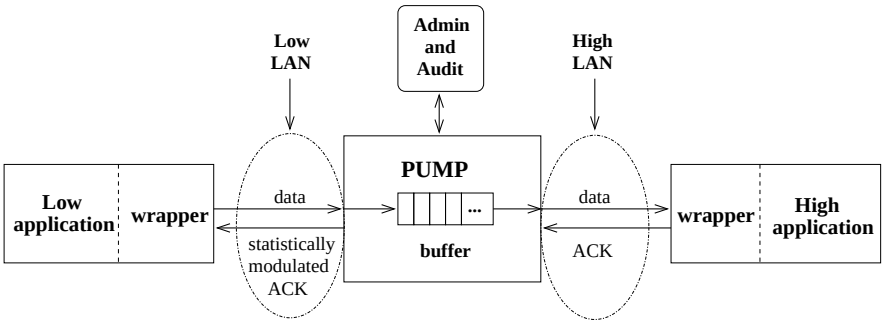

Fig. 2. Network NRL Pump architecture

that any sent message will be delivered to High, even if connection failures occur. Here, we concentrate on non-recoverable applications (like, e.g., FTP), which provide best-effort reliability of connection.

The procedure to establish a connection between Low and High through the Pump is as follows. Initially, Low sends a connection request message to the main thread (MT) of the Pump, which identifies the sending process and the address of the final destination. If both addresses are valid (i.e., they have been previously registered in the configuration file managed by the Pump administrator), MT sends back a connection valid message, otherwise it sends a connection reject message. In the first case, the connection is managed by a trusted low thread (TLT) and a trusted high thread (THT), which are created during the connection setup phase to interact with Low and High, respectively. Registered High processes are always ready to accept a connection from the Pump through the same handshake mechanism seen above.

Once the new connection is established, the Pump sends a connection grant message to both systems with initialization parameters for the communication. During the connection, TLT receives data messages from Low, then stores them in the connection buffer. Moreover, it sends back the acks (which are special data messages with zero data length) in the same order it receives the related data messages, by introducing an additional stochastic delay computed on the basis of the average rate at which THT consumes messages. On the other hand, THT delivers to High any data message contained in the connection buffer. The pump protocol also requires High to send back to THT the ack messages related to the received data messages. If High violates this protocol, THT aborts the connection. In such a case, as soon as TLT detects that THT is dead, it immediately sends all the remaining acks and a connection exit message to Low. Another special data message is connection close, which is sent at the end of a normal connection from Low to the Pump.

In general, the Pump is a reliable, secure, one-way communication device from Low to High, which minimizes the amount of (covert) communication in the opposite direction. During the connection, only THT directly communicates with High and only TLT directly communicates with Low. Moreover, TLT and

THT directly interact only through the connection buffer. However, even if the Pump minimizes any timed covert channel from High to Low [L+04], it cannot avoid some data leak in that direction. This is because the Pump notifies Low when a connection is down. Such a feedback is more than enough to set up a 1-bit covert channel from High to Low. In the following, we apply our methodology to formally verify the existence of such an unavoidable covert channel and we measure its bandwidth and its relation with some dependability and QoS properties.

4 Supporting the Methodology with Æmilia/TwoTowers

The application of our integrated methodology requires a sufficiently expressive specification language, in order to build the functional and performance models of interest. In addition to that, it requires a software tool equipped with the necessary analysis routines, so that the tradeoff between covert channel bandwidth and QoS can be assessed by verifying the properties of the models written in the language mentioned before.

Although the predictive methodology does not depend on a specific notation, in order to illustrate it we need to choose one. Here we use the architectural description language Æmilia [BDC02], together with its companion tool TwoTowers [Ber06], which has recently been extended to encompass security analysis, as they provide all the ingredients that are necessary to support the application of the methodology.

In the following, we give a brief overview of Æmilia and TwoTowers.

4.1 Æmilia

An Æmilia description represents an architectural type. This is an intermediate abstraction between a single system and an architectural style. It consists of a family of systems sharing certain constraints on the observable behavior of the system components as well as on the system topology. As shown in Table 1, the description of an architectural type in Æmilia starts with the name and the formal parameters of the architectural type and is composed of three sections.

The first section defines the types of components that characterize the system family. In order to include both the computational components and the connectors among them, these types are called architectural element types (AETs). The definition of an AET starts with its name and formal parameters and consists of the specification of its behavior and its interactions. The

Table 1
Structure of an Æmilia description

ARCHI_TYPE	◁*name and formal parameters*▷
ARCHI_ELEM_TYPES	
ELEM_TYPE	◁*def. of the first architectural element type*▷
⋮	⋮
ELEM_TYPE	◁*def. of the last architectural element type*▷
ARCHI_TOPOLOGY	
ARCHI_ELEM_INSTANCES	◁*decl. of the architectural element instances*▷
ARCHI_INTERACTIONS	◁*decl. of the architectural interactions*▷
ARCHI_ATTACHMENTS	◁*decl. of the architectural attachments*▷
[BEHAV_VARIATIONS	
[BEHAV_HIDINGS	◁*decl. of the behavioral hidings*▷]
[BEHAV_RESTRICTIONS	◁*decl. of the behavioral restrictions*▷]
[BEHAV_RENAMINGS	◁*decl. of the behavioral renamings*▷]]
END	

behavior has to be provided in the form of a list of sequential defining equations written in a verbose variant of the stochastic process algebra EMPA_{gr} [BB03]. The interactions are those EMPA_{gr} action types occurring in the behavior that act as interfaces for the AET. Each of them has to be equipped with two qualifiers, which establish whether it is an input or output interaction and the multiplicity of the communications in which it can be involved, respectively. All the other action types occurring in the behavior are assumed to represent internal activities.

The second section defines the architectural topology. This is specified in three steps. First we have the declaration of the instances of the AETs (called AEIs) with their actual parameters, which represent the real system components and connectors. Then we have the declaration of the architectural (as opposed to local) interactions, which are some of the interactions of the AEIs that act as interfaces for the whole system family. Finally we have the declaration of the directed architectural attachments among the local interactions of the AEIs, which make the AEIs communicate with each other.

The third section, which is optional, defines some variations of the observable behavior of the system family. This is accomplished by declaring some action

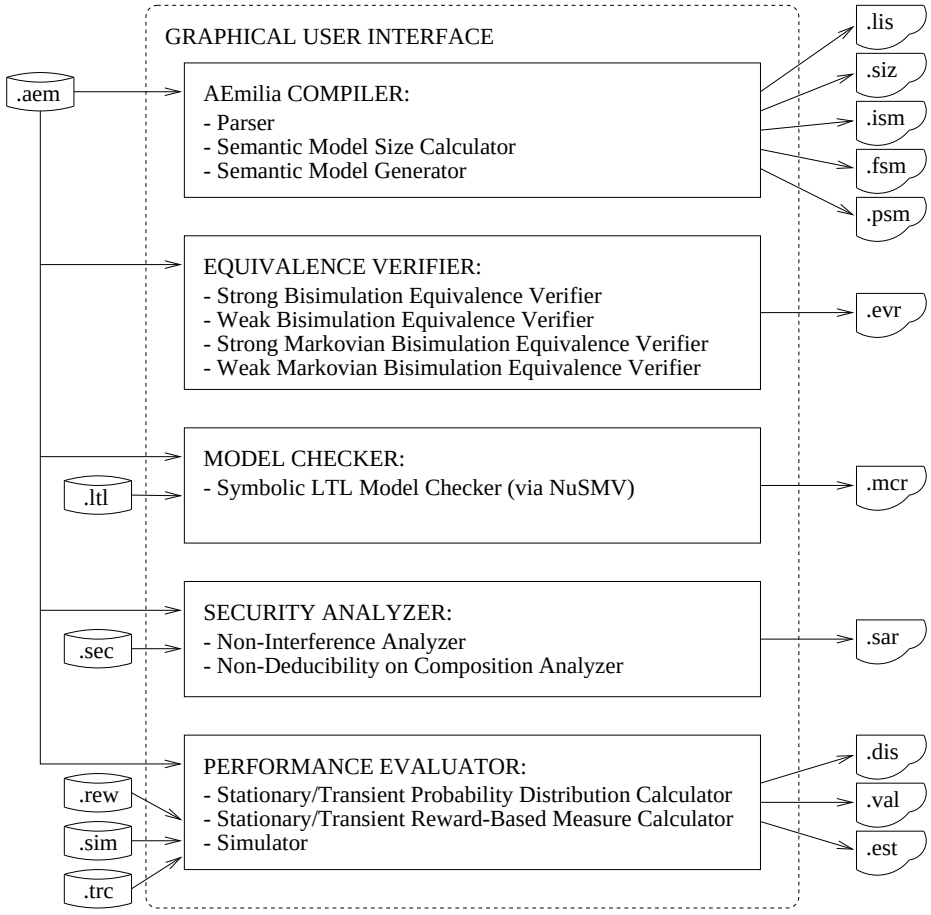

Fig. 3. Architecture of TwoTowers 5.1

types occurring in the behavior of certain AEIs to be unobservable, prevented from occurring, or renamed into other action types.

4.2 TwoTowers 5.1

Æmilia is the input language of the software tool TwoTowers, which has recently been extended to cope with security analysis. As shown in Fig. 3, version 5.1 of TwoTowers is equipped with a graphical user interface through which the user can invoke the analysis routines. Each routine needs input files of certain types and writes its results onto files of other types. The graphical user interface takes care of the integrated management of the various file types needed by the different routines.

The compiler is in charge of parsing system specifications stored in .aem files and signalling possible lexical, syntax and static semantic errors through a .lis file. If a specification is correct, the compiler can generate its integrated, functional or performance semantic model, which is written to a .ism, .fsm

or .psm file, respectively. As a faster option that does not require printing the state space onto a file, the compiler can show only the size – in terms of number of states and transitions – of the semantic model, which is written to a .size file.

The equivalence verifier checks whether two correct specifications are equivalent according to one of several different notions of equivalence. The result of the check, together with some diagnostic information in case of non-equivalence expressed through a modal logic formula, is written to a .evr file.

The model checker verifies through the BDD-based routines of the software tool NuSMV 2.2.5 [C+02] whether a set of functional properties expressed through verbose LTL formulas, which are stored in a .ltl file, are satisfied by a correct, finite-state Æmilia specification. The result of the check, together with a counterexample for each property that is not met, is written to a .mcr file.

The performance evaluator computes the performance characteristics of correct and performance closed specifications. First, it can calculate the stationary/transient state probability distribution of the performance semantic model of a specification, where the model is either a continuous-time or a discrete-time Markov chain. The distribution is written to a .dis file. Second, the performance evaluator can calculate a set of instant-of-time, stationary/transient performance measures specified through state and transition rewards stored in a .rew file. The values of the measures are written to a .mea file. In the stationary case the Gaussian elimination method and an adaptive variant of the symmetric SOR method are available, while in the transient case the method of uniformization is available. Third, the performance evaluator can estimate via discrete event simulation the mean, variance or distribution of a set of performance measures specified through an extension of state and transition rewards stored in a .sim file. The simulation is based on the method of independent replications and can be trace-driven, in which case the traces are stored in .trace files. The outcome of the simulation, which can be applied also to specifications whose underlying performance semantic model is not Markovian, is written to a .est file.

Finally, the newly added security analyzer verifies whether a correct specification possesses certain security properties establishing the absence of illegal information flows from high security components to low security components. Based on the noninterference approach, two security properties can be checked by the security analyzer of TwoTowers 5.1, i.e. strong nondeterministic noninterference and strong nondeducibility on composition [FG95]. In order to verify one of these two security properties, the user is required to specify in an additional .sec file the action names that are high and the action names that are low with respect to the security level. The result of the analysis, to-

gether with some diagnostic information in case of security violation expressed through a modal logic formula, is written to a .sar file.

5 Integrated Security and Performance Analysis of the NRL Pump

In this section we illustrate the use of our integrated methodology to assess the security/QoS tradeoff for the NRL Pump. The results we obtained with Æmilia/TwoTowers can be summarized as follows:

- The noninterference-based security analysis reveals the existence of an unavoidable covert channel caused by a connect/disconnect strategy. Diagnostic information is also provided to detect the functional behavior of the NRL Pump that is responsible for the information leakage. The relation between the detected interference and reliability/availability properties of the NRL Pump is also emphasized.
- Two metrics that are strictly related to the connect/disconnect strategy are evaluated. The derived figures constitute an estimation of the covert channel bandwidth and describe its relation with the NRL Pump configuration parameters.

In the following, we first present the Æmilia functional model of the NRL Pump. Then we describe the noninterference property we checked and we formally show that the success/failure of a connection can be coded into a 1-bit covert channel. Afterwards, we introduce the Æmilia performance model of the NRL Pump, which is obtained from the functional one by adding information about temporal delays and probabilistic behaviors of the NRL Pump activities. Finally, we specify the metrics related to the revealed covert channel on the basis of which we measured the information leakage under certain assumptions.

5.1 Æmilia Functional Model of the NRL Pump

The Æmilia specification of the functional model of the NRL Pump starts with its name and the indication that there is a formal parameter, representing the size of the connection buffer, with its initial value:

`ARCHI_TYPE NRL_Pump_Type(const integer buffer_size` $:= n$`)`

The Æmilia specification of the NRL Pump then proceeds with the definition of the AETs. In Table 2 we report the description of the low wrapper type, whose behavior is given by a single defining equation, which is built out of actions, action prefixes, choices, and behavior invocations. The notation `void`

Table 2
Æmilia functional specification of the low wrapper type

```
ARCHI_ELEM_TYPES
 ELEM_TYPE LW_Type(void)
  BEHAVIOR LW_Beh(void; void) =
    <send_conn_request, _>.
      choice {
        <receive_conn_valid, _>.<receive_conn_grant, _>.
          <send_msg, _>.<receive_low_ack, _>.
            choice {
              <receive_conn_exit, _>.LW_Beh(),
              <send_conn_close, _>.LW_Beh()
            },
        <receive_conn_reject, _>.LW_Beh()
      }
  INPUT_INTERACTIONS UNI receive_conn_valid;
                         receive_conn_grant;
                         receive_conn_reject;
                         receive_low_ack;
                         receive_conn_exit
  OUTPUT_INTERACTIONS UNI send_conn_request;
                          send_msg;
                          send_conn_close
```

in the definition of the architectural type `LW_Type` denotes the absence of data parameters, while the notation `void;void` in the definition of the equation `LW_Beh` represents the absence of data parameters and local variables. Actions are described as pairs of the form $<action\_name, action\_duration>$, where the second element is not specified in the case the Æmilia specification is purely functional.

The low wrapper sends a connection request to the Pump and then is ready to accept either a connection valid message or a connection reject message. If a connection is established, the low wrapper receives a grant message, sends a data message to TLT, and then waits for the related ack. For the sake of

Table 3
Æmilia functional specification of the main thread type

```
ELEM_TYPE MT_Type(void)
 BEHAVIOR MT_Beh(void; void) =
   <receive_conn_request, _>.
     choice {
       <conn_is_valid, _>.<wakeup_tht, _>.
         <send_conn_valid, _>.MT_Beh(),
       <conn_not_valid, _>.
         <send_conn_reject, _>.MT_Beh()
     }
 INPUT_INTERACTIONS UNI receive_conn_request
 OUTPUT_INTERACTIONS UNI wakeup_tht;
                         send_conn_valid;
                         send_conn_reject
```

simplicity, since the amount of data sent from Low to High does not alter the kind of communications between Low and High through the Pump, we considered a system configuration where the low wrapper tries to establish a connection during which a single message is sent to High. Hence, after the reception of the ack, the low wrapper can either receive a connection exit message in the case the connection is aborted by High, or send a connection close message in the case the connection is correctly terminated. The definition of the low wrapper type is concluded with the declaration of some of the action names occurring in its behavior as being input or output interactions, which act as the interfaces of the low wrapper with the other components of the system.

Each connection request sent to the Pump is managed by the main thread, whose type is defined in Table 3. The main thread monitors the port of the Pump to which Low sends connection request messages. In order not to have to introduce a definition of the Pump administrator, the verification of an incoming request is abstractly modeled by means of a nondeterministic choice between two actions. More precisely, in response to a request, either the main thread activates the trusted high thread and sends back a connection valid message, or it sends back a connection reject message.

The initialization of a new connection to High is conducted by THT, which is spawned by MT during the initial setup phase. The definition of the THT type

Table 4
Æmilia functional specification of the trusted high thread type

```
ELEM_TYPE THT_Type(void)
 BEHAVIOR THT_Beh(void; void) =
   choice {
     <receive_high_wakeup, _>.<init_high_conn, _>.
       <wakeup_tlt, _>.THT_Beh(),
     <read_msg, _>.<forward_msg, _>.
       choice {
         <receive_high_ack, _>.<delete_msg, _>.
           <send_ok_to_tlt, _>.THT_Beh(),
         <wait_for_timeout, _>.<comm_timeout, _>.
           <delete_msg, _>.<send_abort_to_tlt, _>.THT_Beh()
       }
   }
 INPUT_INTERACTIONS UNI receive_high_wakeup;
                        read_msg;
                        receive_high_ack
 OUTPUT_INTERACTIONS UNI wakeup_tlt;
                         forward_msg;
                         delete_msg;
                         send_ok_to_tlt;
                         comm_timeout;
                         send_abort_to_tlt
```

is reported in Table 4. Upon the initialization of THT, the connection setup handshaking between THT and the high wrapper is modeled by means of a single action of type `init_high_conn`. Afterwards, THT awakens the trusted low thread. When a connection is active, THT checks the buffer for new incoming data messages. Upon reading a message from the buffer, THT outputs it to the high communication channel. Then, THT waits for the reception of the related ack from High. The arrival of an ack message competes with the timeout fixed by THT. In particular, if the ack is received before the end of the timeout, THT removes the message from the buffer and informs TLT in order to allow the connection to be correctly closed. On the other hand, if

Table 5
Æmilia functional specification of the trusted low thread type

```
ELEM_TYPE TLT_Type(void)
 BEHAVIOR TLT_Beh(void; void) =
   <receive_low_wakeup, _>.<send_conn_grant, _>.
     <receive_msg, _>.<store_msg, _>.
       choice {
         <wait_delay, _>.<send_low_ack, _>.
           choice {
             <receive_abort_from_tht, _>.
               <send_conn_exit, _>.TLT_Beh(),
             <receive_ok_from_tht, _>.
               <receive_conn_close, _>.TLT_Beh()
             },
         <receive_abort_from_tht, _>.<send_low_ack, _>.
           <send_conn_exit, _>.TLT_Beh(),
         <receive_ok_from_tht, _>.<wait_delay, _>.
           <send_low_ack, _>.<receive_conn_close, _>.TLT_Beh()
       }
 INPUT_INTERACTIONS UNI receive_low_wakeup;
                        receive_msg;
                        receive_abort_from_tht;
                        receive_ok_from_tht;
                        receive_conn_close
 OUTPUT_INTERACTIONS UNI send_conn_grant;
                         store_msg;
                         send_low_ack;
                         send_conn_exit
```

the timeout expires before the reception of the ack, THT notifies the timeout expiration, removes the message from the buffer, and informs TLT about the aborted connection.

Table 6
Æmilia functional specification of the buffer type

```
ELEM_TYPE Buffer_Type(const integer buffer_size)
 BEHAVIOR Buffer_Beh(integer(0..buffer_size) msg_num := 0; void) =
   choice {
     cond(msg_num < buffer_size) ->
       <accept_msg, _>.Buffer_Beh(msg_num + 1),
     cond(msg_num > 0) ->
       choice {
         <read_msg, _>.Buffer_Beh(msg_num),
         <delete_msg, _>.Buffer_Beh(msg_num - 1)
       }
   }
 INPUT_INTERACTIONS UNI accept_msg;
                        delete_msg
 OUTPUT_INTERACTIONS UNI read_msg
```

In turn, the description of TLT is given in Table 5. TLT waits for THT to awaken it and then establishes the connection from Low to the Pump by sending a connection grant message to Low. At that moment, TLT is ready to receive a data message from Low. Upon receiving a data message, TLT stores it in the connection buffer and then sends the ack to Low after a certain delay. At any moment, TLT may receive a message from THT concerning the status of the connection. In particular, in the case of THT failure, TLT must send a connection exit message to Low. Alternatively, if THT is correctly working, TLT can accept a connection close message from Low. Note that if TLT detects the THT failure before sending the ack to Low, then TLT immediately transmits the ack and the connection exit message to Low.

TLT and THT share the communication buffer, through which the data messages coming from Low are forwarded to High. The buffer element type is described in Table 6. Its definition is parameterized with respect to the maximum size of the buffer, while its behavior is characterized by the number of messages that are currently stored, ranging from 0 to the maximum size. The buffer is initially empty and is accessed by TLT and THT only. When the buffer is not full, i.e. the condition `msg_num` $<$ `buffer_size` holds, a new data message can be accepted from TLT. When the buffer is not empty, i.e. the condition `msg_num` $>$ `0` holds, a data message can be read (deleted) from

Table 7
Æmilia functional specification of the high channel type

```
ELEM_TYPE HC_Type(void)
 BEHAVIOR HC_Beh(void; void) =
   <accept_msg, _>.
     choice {
       <receive_timeout, _>.HC_Beh(),
       <transmit_msg, _>.
         choice {
           <receive_timeout, _>.HC_Beh(),
           <accept_high_ack, _>.
             choice {
               <receive_timeout, _>.HC_Beh(),
               <transmit_high_ack, _>.HC_Beh()
             }
         }
     }
 INPUT_INTERACTIONS UNI accept_msg;
                        receive_timeout;
                        accept_high_ack
 OUTPUT_INTERACTIONS UNI transmit_msg;
                         transmit_high_ack
```

THT. Since we assumed that during a connection Low sends a single message to High, it is enough to consider a buffer with size $n = 1$.

The high channel type, described in Table 7, models the communication channel between THT and High. We need an explicit element type to express the transmission delay of messages in that link, because the round-trip delay of a communication between THT and High must compete with the timeout set by THT. Initially, the channel is ready to accept a data message from THT, which is then transmitted to High. After the delivery of the message, the channel waits for the related ack to be transmitted to THT. Such a handshake competes with the notification of the timeout from THT, which represents a connection abort. In this case, for the sake of simplicity, the channel loses all the pending messages.

Table 8
Æmilia functional specification of the high wrapper type

```
ELEM_TYPE HW_Type(void)
 BEHAVIOR HW_Beh(void; void) =
   <receive_msg, _>.<send_high_ack, _>.HW_Beh()
 INPUT_INTERACTIONS UNI receive_msg
 OUTPUT_INTERACTIONS UNI send_high_ack
```

Table 9
Æmilia functional specification of the NRL Pump topology (part I)

```
ARCHI_TOPOLOGY
 ARCHI_ELEM_INSTANCES
   LW : LW_Type();
   MT : MT_Type();
   THT : THT_Type();
   TLT : TLT_Type();
   B : Buffer_Type(buffer_size);
   HC : High_Channel_Type();
   HW : HW_Type()
```

At the high receiving site, the high wrapper is ready to interact with the high components of the Pump. The definition of the high wrapper type is given in Table 8. The high wrapper can accept a data message from the high channel and, in such a case, must transmit an ack message. Note that the high wrapper does not perform other operations, as we abstract away from the communications concerning the high connection initialization/termination.

Finally, the Æmilia specification of the NRL Pump contains the description of the system topology, in accordance with Fig. 2. In particular, as shown in Table 9, the Pump is composed of a main thread, a TLT, a THT, and a buffer. It interacts with a single low wrapper and a single high wrapper through a single high channel. Besides the declaration of all the instances of the AETs, the description of the system topology contains the declaration of the attachments involving the interactions of each instance, as shown in Table 10.

Table 10
Æmilia functional specification of the NRL Pump topology (part II)

```
ARCHI_INTERACTIONS void

ARCHI_ATTACHMENTS

   FROM LW.send_conn_request  TO MT.receive_conn_request;

   FROM MT.send_conn_valid    TO LW.receive_conn_valid;

   FROM MT.send_conn_reject   TO LW.receive_conn_reject;

   FROM MT.wakeup_tht         TO THT.receive_high_wakeup;

   FROM THT.wakeup_tlt        TO TLT.receive_low_wakeup;

   FROM TLT.send_conn_grant   TO LW.receive_conn_grant;

   FROM LW.send_msg           TO TLT.receive_msg;

   FROM TLT.store_msg         TO B.accept_msg;

   FROM TLT.send_low_ack      TO LW.receive_low_ack;

   FROM B.read_msg            TO THT.read_msg;

   FROM THT.forward_msg       TO HC.accept_msg;

   FROM HC.transmit_msg       TO HW.receive_msg;

   FROM THT.comm_timeout      TO HC.receive_timeout;

   FROM HW.send_high_ack      TO HC.accept_high_ack;

   FROM HC.transmit_high_ack  TO THT.receive_high_ack;

   FROM THT.delete_msg        TO B.delete_msg;

   FROM THT.send_abort_to_tlt TO TLT.receive_abort_from_tht;

   FROM THT.send_ok_to_tlt    TO TLT.receive_ok_from_tht;

   FROM TLT.send_conn_exit    TO LW.receive_conn_exit;

   FROM LW.send_conn_close    TO TLT.receive_conn_close

END
```

5.2 Noninterference-based Security Analysis

According to Sect. 2, the first phase of our methodology consists of applying the noninterference check to the Æmilia functional specification of the NRL Pump. For this purpose, the security analyzer of TwoTowers 5.1 allows the designer to describe in an auxiliary specification file (.sec) which actions belong to *High* and which belong to *Low*. All the other actions are simply disregarded

Table 11
Æmilia specification of the .sec file for the NRL Pump model

```
HIGH HW.receive_msg;
     HW.send_high_ack
LOW LW.receive_conn_valid;
    LW.receive_conn_grant;
    LW.receive_conn_reject;
    LW.receive_low_ack;
    LW.receive_conn_exit;
    LW.send_conn_request;
    LW.send_msg;
    LW.send_conn_close
```

by turning them into invisible actions.

As far as the NRL Pump is concerned, the low-level view of the system is represented by the communication interface between the low wrapper and the Pump, as they interact through low-level actions. Analogously, all the actions modeling communications between the high wrapper and the Pump are high-level actions. All the actions modeling communications among the internal components of the Pump (like, e.g., the synchronizations between MT and THT, or between TLT and the buffer) cannot be seen by an external observer. Therefore, as far as the security check is concerned, it is reasonable to assume that they are invisible, as they do not represent communications between the Pump and the wrappers. The content of the .sec file associated with the Æmilia specification of the NRL Pump is described in Table 11.

The security check we applied is based on strong nondeterministic noninterference [FG95]. In particular, the security analyzer of TwoTowers 5.1 automatically derives the two views to be compared from the Æmilia specification of the NRL Pump, i.e. `NRL_Pump_Type`$\backslash$*High* and `NRL_Pump_Type`/*High*, and performs the weak bisimulation equivalence check. The obtained result is that they cannot be weakly bisimulation equivalent. The distinguishing modal logic formula returned by TwoTowers 5.1 intuitively shows what follows: `NRL_Pump_Type`$\backslash$*High* aborts all the connections (each connection terminates with the occurrence of the low-level action modeling the transmission of a connection exit message), while `NRL_Pump_Type`/*High* is able to close connections between Low and High (a connection may terminate with the occurrence of the low-level action modeling the transmission of a connection close message). The related covert channel is caused by the unavoidable notification

feedback from the Pump to Low. Indeed, if we prevent Low from observing the result of each connection (by hiding the low-level actions modeling the connection close/exit message), we obtain that the system satisfies both strong nondeterministic noninterference and strong nondeducibility on composition. That means the covert channel described above is the unique nondeterministic information leakage that occurs in the NRL Pump.

As far as other dependability related issues are concerned, the noninterference check we conducted puts in evidence the reliability of the NRL Pump in terms of its ability to deliver the required service under intentional faults caused by High. In this respect, we point out that both $\texttt{NRL\_Pump\_Type}\backslash \mathit{High}$ and $\texttt{NRL\_Pump\_Type}/\mathit{High}$ are deadlock free. Therefore, the behavior of High, which can be either trusted or nontrusted, cannot compromise the Pump functionalities. Moreover, independently of the behavior of the current connection, the Pump is eventually available to accept new incoming connection requests. In particular, in the case High cheats by performing a denial-of-service attack, see $\texttt{NRL\_Pump\_Type}\backslash \mathit{High}$, the Pump is able to abort the connection by exploiting the timeout mechanism, thus becoming ready for new incoming requests. In other words, the service offered by the NRL Pump satisfies the availability property, even if the functional analysis conducted in this phase is not sufficient to evaluate efficiency issues. For this purpose, it is necessary to pass to the second phase of our methodology, which is based on the analysis of the performance model of the NRL Pump.

5.3 Æmilia Performance Model of the NRL Pump

The Æmilia performance model of the NRL Pump has two main differences with respect to the functional specification provided in Sect. 5.1. First, the description of the architectural type is parameterized with respect to a set of rates and probabilities concerned with the Pump activities, which are passed as actual parameters to the AEIs in the architectural topology section. The header of the NRL Pump architectural type is reported in Table 12. The formal parameters represent the size of the connection buffer, the rates modeling some exponentially distributed delays, and the probability that a connection request is valid. In particular, `conn_gen_rate` is the Low connection request generation rate, `conn_init_rate` is the High connection initialization rate, `data_trans_rate` (resp. `ack_trans_rate`) is the data (resp. ack) message transmission rate, `ack_delay_rate` is the inverse of the stochastic delay added by the Pump to the transmission of the acks to Low, and `timeout_rate` is the inverse of the maximum amount of time that the Pump waits for an expected ack.

Second, every action can now contain the specification of its duration. This is

Table 12
Æmilia performance specification of the NRL Pump header

```
ARCHI_TYPE NRL_Pump_Type(const integer buffer_size := n,
                         const rate conn_gen_rate := γ,
                         const rate conn_init_rate := η,
                         const rate data_trans_rate := δ,
                         const rate ack_trans_rate := κ,
                         const rate ack_delay_rate := θ,
                         const rate timeout_rate := μ,
                         const weight valid_prob := p)
```

given by `exp(_)` in the case of an exponentially timed action, while it is represented by `inf(_, _)` in the case of an immediate action. The two parameters of an immediate action are its priority level and its weight, whose default value is `1`. All the other actions are called passive and get a duration only if they are attached to an exponentially timed or immediate action. Actions that are not passive cannot be attached to each other.

In Tables 13, 14, and 15 we report the behavior of the AETs of the Æmilia performance specification of the NRL Pump. For instance, consider the main thread type described in Table 13. With respect to the functional specification of Table 3, the verification of an incoming request is modeled by a probabilistic choice between two immediate actions, which is governed by parameter `valid_prob`. Then, the activation of the trusted high thread is modeled through an immediate action, while the connection notification that is sent back to Low is modeled through an exponentially timed action whose duration is quantified by rate `data_trans_rate`. In general, all the transmission delays for the messages exchanged by the Pump and Low/High through the network are modeled as stochastic random variables guided by exponential distributions, while all the internal communications among the Pump components are modeled by immediate actions. This is because the duration of an activity internally performed by the Pump is negligible with respect to the transmission delay experienced by a message along the network.

5.4 Performance Evaluation and Tuning

According to the second phase of our methodology, whenever the performance model is validated against the functional model, the bandwidth of the covert channels that are not eliminated during the first phase is measured by eval-

Table 13
Æmilia performance specification of the NRL Pump (part I)

```
ELEM_TYPE LW_Type(const rate conn_gen_rate, const rate data_trans_rate)
 BEHAVIOR LW_Beh(void; void) =
   <send_conn_request, exp(conn_gen_rate)>.
     choice {
       <receive_conn_valid, _>.<receive_conn_grant, _>.
         <send_msg, exp(data_trans_rate)>.<receive_low_ack, _>.
           choice {
             <receive_conn_exit, _>.LW_Beh(),
             <send_conn_close, exp(data_trans_rate)>.LW_Beh()
           },
       <receive_conn_reject, _>.LW_Beh()
     }
     ...
ELEM_TYPE MT_Type(const rate data_trans_rate, const weight valid_prob)
 BEHAVIOR MT_Beh(void; void) =
   <receive_conn_request, _>.
     choice {
       <conn_is_valid, inf(1, valid_prob)>.<wakeup_tht, inf>.
         <send_conn_valid, exp(data_trans_rate)>.MT_Beh(),
       <conn_not_valid, inf(1, 1 − valid_prob)>.
         <send_conn_reject, exp(data_trans_rate)>.MT_Beh()
     }
     ...
ELEM_TYPE THT_Type(const rate conn_init_rate, const rate timeout_rate)
 BEHAVIOR THT_Beh(void; void) =
   choice {
     <receive_high_wakeup, _>.<init_high_conn, exp(conn_init_rate)>.
       <wakeup_tlt, inf>.THT_Beh(),
     <read_msg, _>.<forward_msg, inf>.
```

Table 14
Æmilia performance specification of the NRL Pump (part II)

```
      choice {
        <receive_high_ack, _>.<delete_msg, inf>.
          <send_ok_to_tlt, inf>.THT_Beh(),
        <wait_for_timeout, exp(timeout_rate)>.<comm_timeout, inf>.
          <delete_msg, inf>.<send_abort_to_tlt, inf>.THT_Beh()
      }
  }
    ...
ELEM_TYPE TLT_Type(const rate data_trans_rate,
                   const rate ack_trans_rate,
                   const rate ack_delay_rate)
 BEHAVIOR TLT_Beh(void; void) =
  <receive_low_wakeup, _>.<send_conn_grant, exp(data_trans_rate)>.
    <receive_msg, _>.<store_msg, inf>.
      choice {
        <wait_delay, exp(ack_delay_rate)>.
          <send_low_ack, exp(ack_trans_rate)>.
            choice {
              <receive_abort_from_tht, _>.
                <send_conn_exit, exp(data_trans_rate)>.TLT_Beh(),
              <receive_ok_from_tht, _>.
                <receive_conn_close, _>.TLT_Beh()
            },
        <receive_abort_from_tht, _>.
          <send_low_ack, exp(ack_trans_rate)>.
            <send_conn_exit, exp(data_trans_rate)>.TLT_Beh(),
        <receive_ok_from_tht, _>.<wait_delay, exp(ack_delay_rate)>.
          <send_low_ack, exp(ack_trans_rate)>.
            <receive_conn_close, _>.TLT_Beh()
      }
    ...
```

Table 15
Æmilia performance specification of the NRL Pump (part III)

```
ELEM_TYPE Buffer_Type(const integer buffer_size)
 BEHAVIOR Buffer_Beh(integer(0..buffer_size) msg_num := 0; void) =
   choice {
     cond(msg_num < buffer_size) ->
       <accept_msg, _>.Buffer_Beh(msg_num + 1),
     cond(msg_num > 0) ->
       choice {
         <read_msg, inf>.Buffer_Beh(msg_num),
         <delete_msg, _>.Buffer_Beh(msg_num - 1)
       }
   }
     ...
ELEM_TYPE HC_Type(const rate data_trans_rate, const rate ack_trans_rate)
 BEHAVIOR HC_Beh(void; void) =
   <accept_msg, _>.
     choice {
       <receive_timeout, _>.HC_Beh(),
       <transmit_msg, exp(data_trans_rate)>.
         choice {
           <receive_timeout, _>.HC_Beh(),
           <accept_high_ack, _>.
             choice {
               <receive_timeout, _>.HC_Beh(),
               <transmit_high_ack, exp(ack_trans_rate)>.HC_Beh()
             }
         }
     }
     ...
ELEM_TYPE HW_Type(void)
 BEHAVIOR HW_Beh(void; void) =
   <receive_msg, _>.<send_high_ack, inf>.HW_Beh()
```

Table 16
Æmilia specification of the .rew file for the NRL Pump model

```
MEASURE closed_connections_per_time_unit IS
      ENABLED(LW.send_conn_close) -> TRANS_REWARD(1);
MEASURE aborted_connections_per_time_unit IS
      ENABLED(TLT.send_conn_exit) -> TRANS_REWARD(1)
```

uating some related efficiency indicators. For this purpose, action durations have to be taken into consideration.

In the case of the Æmilia specification of the NRL Pump, first we observe that, when going from the functional model to the performance model, the verification of strong nondeterministic noninterference reveals the same covert channel described in Sect. 5.2. In particular, the security check revealed that the unique information leakage from High to Low is given by the occurrence of a connection exit event (in case High is absent) with respect to the occurrence of either a connection exit event or a connection close event (in case High is present). Hence, the number of connections that can be closed/aborted because of the behavior of High represents an estimate of how many bits High can pass to Low in a certain period.

Second, some delays of the NRL Pump activities, such as timeouts and transmission times, are modeled as stochastic random variables governed by exponential distributions. Thus, the stochastic model we obtain is a continuous-time Markov chain. To derive performance measures of interest, such a Markov chain can be analyzed by the performance evaluator of TwoTowers 5.1 through standard numerical techniques. To this aim, following [BB03] the designer describes in an auxiliary specification file (.rew) the rewards to be attached to specific actions of the Æmilia description. These rewards are then exploited to compute reward-based metrics, such as throughput and utilization measures.

Formally, the number of connections that are closed/aborted by the NRL Pump is estimated by measuring the throughput of the low-level actions modeling the transmission of the connection close and the connection exit messages that are observed by Low. As a consequence, the content of the .rew file associated with the Æmilia specification of the NRL Pump is as described in Table 16.

Before showing the analysis results, we explain some assumptions about the timing of the actions occurring in the Æmilia specification of the NRL Pump. All the delays are exponentially distributed with a certain rate expressed in sec^{-1}. The data (resp. ack) transmission rate and the round-trip propagation rate experienced during the connection setup phase between the Pump and

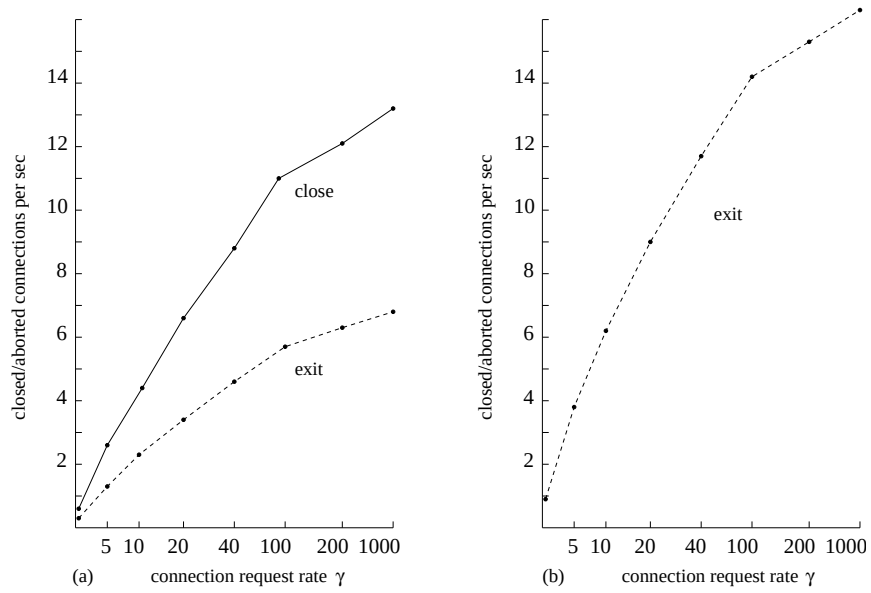

Fig. 4. Throughput of closed/aborted connections with and without High

High are δ (resp. κ) and η. We assume that the Pump uses two 64 Kbps full-duplex lines and the (mean) length of data (resp. ack) messages is 512 (resp. 49) bits, so that $\delta = 125$ (resp. $\kappa = 1306.12$) and $\eta = 62.5$. The connection request generation rate γ varies in the range $[1, 1000]$, i.e. from 1 request/sec to 1 request/ms. The rate of the stochastic delay added by the Pump before sending the ack to Low is θ. We assume such a delay to be equal to the transmission time of three ack messages, so that $\theta = 435.37$. This is long enough to hide the fluctuations of the transmission delays of the ack messages propagating from High to the Pump. The timeout delay used by the Pump when waiting for the ack from High varies from 2 sec to 10 ms. Therefore, the corresponding rate, denoted μ, varies in the range $[0.5, 100]$. Finally, for each connection request we abstract from the configuration file look-up and we assume that each incoming request is valid with probability $p = 0.99$.

Fig. 4 reports the number of connection close/exit messages observed per sec in the case $\mu = 57.04$, corresponding to double the average time needed to send a data message and to receive the related ack (i.e., about 17 ms). Fig. 4(a) refers to the scenario in which High correctly executes the protocol. Therefore, most connections are normally closed, while aborted connections can occur because of the expiration of the timeout set by the Pump. Fig. 4(b) refers to the scenario in which High is absent, i.e. all the connections abort. For both scenarios, we have that as the connection request rate γ increases, the number of closed/aborted connections increases as well. Note that abortions occur in both figures independently of the behavior of High. As a consequence, a connection exit message cannot reveal the presence/absence of High. Instead, Low deduces the presence of High if a connection is correctly closed, which is an event that occurs in Fig. 4(a) only. In particular, from Fig. 4(a) we derive

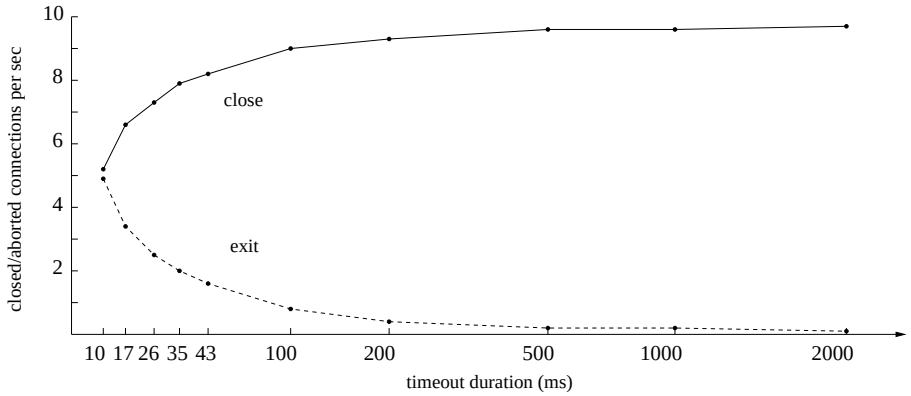

Fig. 5. Tradeoff between timeout duration and throughput of closed/aborted connections

that High succeeds in leaking its presence to Low up to 13 times/sec. Finally, note that the difference between the curve of Fig. 4(b) and the corresponding curve of Fig. 4(a) shows that the number of aborted connections observed per sec is appreciably altered by the absence of High. That means Low can deduce the presence of High by simply measuring the number of connection exit messages received per sec.

The number of connections that abort because of the timeout expiration can be limited by increasing the timeout duration. In Fig. 5 we show the tradeoff between the timeout duration and the Pump throughput in terms of number of connections served per sec. In particular, we consider a scenario where both Low and High correctly execute the protocol, $\gamma = 20$ (corresponding to a connection request every 50 ms), and the timeout duration varies in the interval $[10, 2000]$ ms (i.e., μ varies from 100 to 0.5). The curves show that as the timeout duration increases, the number of connection exit messages tends to zero, while the number of connection close messages rises up to 9 per sec. The most interesting result is that, whenever the timeout expires after at least 200 ms, it is very likely that an ack sent by High arrives before the expiration of the timeout. More precisely, for $\mu = 5$ we have 0.412051 abortions/sec, while in the limiting scenario where the timeout duration is 2 sec we observe 0.0425696 abortions/sec, corresponding to 2.554176 abortions/min. In other words, it is reasonable to predict with good approximation that an aborted connection occurs because of a misbehavior of High rather than a timeout expiration. Hence, High may exploit the connection exit message to leak a bit to Low, i.e. each connection really leaks a bit from High to Low (e.g., 0 if it succeeds and 1 if it fails).

In order to measure the bandwidth of such a 1-bit covert channel, in Fig. 6 we report the number of connections served per sec whenever High alternatively completes and blocks (with equal probabilities) the connections in order to express a sequence of bits to be sent to Low. As far as the configuration parameters are concerned, the connection request rate varies from 1 per sec

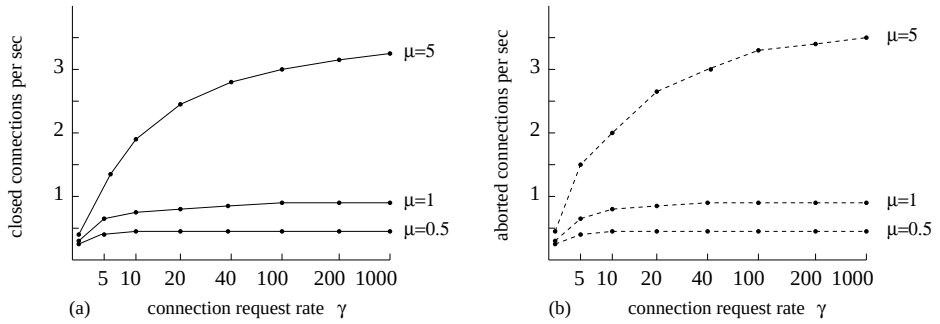

Fig. 6. Throughput of closed/aborted connections for different configuration parameters

($\gamma = 1$) to 1 per ms ($\gamma = 1000$) and the timeout duration is long enough to ensure an exact interpretation of the information leakage ($\mu \in \{0.5, 1, 5\}$). In particular, if the timeout duration is 200 ms, then we observe a number of closed connections between 0.41382 and 3.23043, and a number of aborted connections between 0.45022 and 3.51459. That means, if Low interpretes each connection termination as a leaked bit, in the worst case the maximum information leakage is 6.77633 bits/sec. However, in Fig. 5 we have seen that some abortion is not due to the High behavior, but it depends on the timeout expiration. As a consequence, a certain percentage of the bit sequence deduced by Low in a sec is wrong. In the scenario above ($\mu = 5$ and $\gamma \in [1, 1000]$) such a percentage is equal to 4.043% independently of the connection request frequency.

Obviously, there exists a tradeoff between the number of bits/sec that are deduced by Low and the accuracy of the deduction. For instance, in the case $\mu = 1$ the maximum information leakage is 1.82353 bits/sec with an error percentage equal to 0.862%, while in the case $\mu = 0.5$ the maximum information leakage is 0.95383 bits/sec with an error percentage equal to 0.435%. Another remark is in order about the comparison between Fig. 5 and Fig. 6, which is conducted by observing the curves of Fig. 6 for the value $\gamma = 20$ and by taking the same timeout duration. When $\mu = 5$, in Fig. 6 we observe 2.4219 closed connections per sec and 2.63493 aborted connections per sec, corresponding to 5.05683 bits/sec, which is appreciably less than the number of closed connections per sec in Fig. 5, i.e. 9.59341. The main difference is that in the scenario of Fig. 5 High completes all the connections, while in the scenario of Fig. 6 High alternatively completes and blocks the connections. Therefore, the bandwidth of the covert channel also depends on the sequence of bits that are leaked from High to Low.

In general, the Pump designer can quantitatively assess the relation between the amount of bits leaked from High to Low and the value of each configuration parameter that influences the QoS delivered by the NRL Pump. For instance, we have seen that covert channel bandwidth and Pump throughput (in terms of number of connections served per sec) are directly proportional.

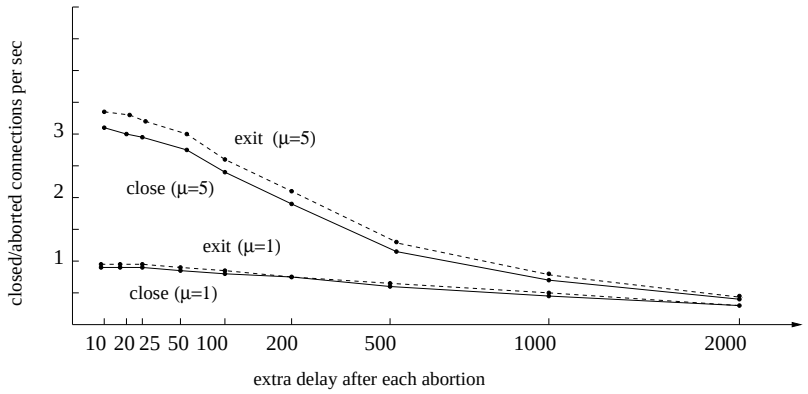

Fig. 7. Throughput of closed/aborted connections with extra delay

Therefore, the availability of the NRL Pump, expressed in terms of efficiency in responding to the incoming requests, is inversely proportional to the confidentiality degree offered by the NRL Pump. As another example, there exists an important relation between the timeout duration chosen by the Pump and the amount of information flowing from High to Low. In particular, the longer the timeout duration is, the more an aborted connection may be interpreted as a leaked bit with high accuracy.

A strategy to reduce the covert channel bandwidth consists of enforcing a minimum delay to elapse between subsequent connection establishments. Consider, e.g., the addition of an extra delay, exponentially distributed with rate λ, after the abortion of a connection and before its reestablishment. In Fig. 7 we report the effect of this extra delay in the case High alternatively completes and blocks (with equal probabilities) the connections, $\gamma = 200$, and $\mu \in \{1, 5\}$. The extra delay varies from 500 to 10 ms, i.e. $\lambda \in [2, 100]$. As an expected result, as the artificial delay increases the total number of closed/aborted connections per sec decreases. For instance, in the case $\mu = 5$, the covert channel bandwidth ranges from an upper bound of 6.48454 bits/sec to a lower bound of 0.84203 bits/sec. We recall that the covert channel bandwidth in the corresponding scenario of Fig. 6 is 6.56608 bits/sec. Hence, the bandwidth reduction is proportional to the extra delay duration. In the case $\mu = 1$, the connections are equally divided into aborted and closed. The information leakage ranges from 1.80413 bits/sec to 0.643281 bits/sec.

As a consequence of the obtained results, the covert channel bandwidth can be reduced under any desired threshold in spite of a reduction of the QoS, expressed in terms of number of requests served per sec. This is because each connection served by the NRL Pump leaks one bit. In practice, a tradeoff exists between the robustness against the 1-bit covert channel and the QoS delivered by the NRL Pump. To reduce the unfavorable impact of the proposed strategy on the QoS, which could be unacceptably burdensome, the extra delay mechanism should be carefully activated, e.g. only in the case of frequent abortions, which are an evidence of the misbehavior of High. Moreover, if the

Pump can handle several different connections at the same time, a good quality degree of the delivered service might be guaranteed by monitoring the kind of traffic. This can be done by reducing the waiting time for the trusted users that behave correctly and by adopting the delay mechanism for the suspicious connections that may try to exploit the 1-bit covert channel. In this way, the availability and the QoS offered by the NRL Pump are guaranteed in spite of a tolerable degree of interference causing the information leakage.

6 Conclusion: Related and Future Work

In this paper we have presented an integrated methodology – implemented through the Æmilia/TwoTowers technology – that combines noninterference-based security analysis and performance evaluation in order to trade QoS with covert channel bandwidth.

On the one hand, the need for both qualitative and quantitative security assessment stems from the fact that real systems like the NRL Pump suffer from unavoidable information leaks that have to be quantified. In particular, through the NRL Pump case study we have shown that the existence of an unwanted covert channel has an impact on both reliability- and security-related properties. Such a covert channel causes an information leakage which we have quantitatively estimated in terms of the number of bits leaked per unit of time.

On the other hand, performance evaluation allows for a quantitative estimation of the efficiency of the securing strategies implemented to reduce the covert channel bandwidth. For instance, through the NRL Pump case study we have shown that a tradeoff exists between the accuracy of the system in avoiding the information leakage and the QoS/reliability offered to the users.

The application of such a methodology represents an effective support to validating the security guarantees of real systems while preserving the expected QoS. For instance, audio/video applications based on real-time channels require both critical QoS constraints and privacy guarantees. Such applications often offer customized security (choice of the authentication and privacy methods, tolerance to replay attacks, use of caching and prefetching strategies) to achieve a customized tradeoff between security and performance, which can be formally analyzed and supported by the use of our methodology.

Although the methodology is presented by stressing on the security domain, it can be easily generalized to consider other dependability-related issues. For instance, in [SD98,SWD98,DiV99] different trace-based models of noninterference are used for specifying and verifying safety properties, while in [A$^+$04] an approach similar to the methodology proposed in this paper is employed

to assess QoS and reliability-related properties. In this respect, by adequately changing the roles of Low and High in the first phase of our methodology, the noninterference check may still provide useful results related to, e.g., safety. Accordingly, the performance analysis of the second phase would allow for an estimation of the efficiency of the strategies implemented to ensure safety. For instance, if we interpret Low as the portion of the system performing safety-critical functions and we interpret High as a set of faulty system components, the noninterference check can be useful to verify whether the system is tolerant against accidental rather than deliberate faults.

As far as the quantitative evaluation of covert channels is concerned, several formal approaches have been proposed to estimate the amount of information leakage, but none of them are concerned with the relation between security degree and QoS delivered by the system. For instance, an approach aiming at quantifying information flow has been proposed in [Lowe02], where the quantity is defined in terms of the number of different high-level behaviors that establish an information flow. This approach does not consider probabilistic behaviors and performance metrics. Instead, it relies on a worst-case analysis based on all possible ways in which the system can interact with the environment. As another example, [DHW04] estimates the leakage of information in terms of number of statistical tests needed to distinguish the illegal information flow. However, such an estimation is not related to information flow capacity issues.

As future work, we would like to strengthen the integration of security analysis and performance evaluation, in such a way that the illustrative modal logic formula returned in case of security violation automatically determines the performance metrics affecting the bandwidth of the information leakage.

Finally, it would be interesting to extend the methodology to consider not only nondeterministic covert channels, but also interferences caused, e.g., by temporal and probabilistic aspects of the system behavior [FGM03,ABG03,LMT05]. In this respect, a further step would be the integrated analysis on the same system model of each kind of information flow – nondeterministic, temporal, and probabilistic – in order to provide for each of them a quantitative estimation of the amount of information leakage.

References

[A+04] A. Acquaviva, A. Aldini, M. Bernardo, A. Bogliolo, E. Bontà, and E. Lattanzi, Assessing the Impact of Dynamic Power Management on the Functionality and the Performance of Battery-Powered Appliances, in *5th Conf. on Dependable Systems and Networks (DSN'04) – Performance and Dependability Symposium* (IEEE CS Press, 2004) 731–740.

[AB04a] A. Aldini and M. Bernardo, An Integrated View of Security Analysis and Performance Evaluation: Trading QoS with Covert Channel Bandwidth, in *23rd Conf. on Computer Safety, Reliability and Security (SAFECOMP'04)* LNCS **3219** (Springer Verlag, 2004) 283–296.

[AB04b] A. Aldini and M. Bernardo, TwoTowers 4.0: Towards the Integration of Security Analysis and Performance Evaluation, in *1st Conf. on Quantitative Evaluation of Systems (QEST'04)* (IEEE CS Press, 2004) 336–337.

[ABG03] A. Aldini, M. Bravetti, and R. Gorrieri, A Process-algebraic Approach for the Analysis of Probabilistic Noninterference, *Journal of Computer Security* **12** (IOS Press, 2004) 191–245.

[AG02] A. Aldini and R. Gorrieri, Security Analysis of a Probabilistic Non-repudiation Protocol, in *2nd Workshop on Process Algebra and Performance Modelling, Probabilistic Methods in Verification (PAPM-ProbMiV'02)* LNCS **2399** (Springer Verlag, 2002) 17–36.

[Ber06] M. Bernardo, *"TwoTowers 5.1 User Manual"* (2006) `http://www.sti.uniurb.it/bernardo/twotowers/`.

[BB03] M. Bernardo and M. Bravetti, Performance Measure Sensitive Congruences for Markovian Process Algebras, *Theoretical Computer Science* **290** (2003) 117–160.

[BDC02] M. Bernardo, L. Donatiello, and P. Ciancarini, Stochastic Process Algebra: From an Algebraic Formalism to an Architectural Description Language, *Performance Evaluation of Complex Systems: Techniques and Tools* LNCS **2459** (Springer Verlag, 2002) 236–260.

[C+02] R. Cavada, A. Cimatti, E. Olivetti, M. Pistore, and M. Roveri, *"NuSMV 2.1 User Manual"* (2002) `http://nusmv.irst.itc.it/`.

[DHW04] A. Di Pierro, C. Hankin, and H. Wiklicky, Approximate Non-Interference *Journal of Computer Security* **12** (IOS Press, 2004) 37-81.

[DiV99] Ben L. Di Vito, A Model of Cooperative Noninterference for Integrated Modular Avionics, *7th Conf. on Dependable Computing for Critical Applications (DCCA-7)* (IEEE CS Press, 1999) 269–286.

[FG95] R. Focardi and R. Gorrieri, A Classification of Security Properties, *Journal of Computer Security* **3** (IOS Press, 1995) 5–33.

[FGM03] R. Focardi, R. Gorrieri, and F. Martinelli, Real-Time Information Flow Analysis, *Journal on Selected Areas in Communications* **21** (IEEE CS Press, 2003) 20–35.

[GM82] J.A. Goguen and J. Meseguer, Security Policy and Security Models, in *Symposium on Security and Privacy (SSP'82)* (IEEE CS Press, 1982) 11–20.

[KMM98] M.H. Kang, A.P. Moore, and I.S. Moskowitz, Design and Assurance Strategy for the NRL Pump, *NRL Memo* 5540-97-7991 (Naval Research Laboratory, Washington, D.C., 1997) in *IEEE Computer Magazine* **31** (1998) 56–64.

[L+04] R. Lanotte, A. Maggiolo-Schettini, S. Tini, A. Troina, and E. Tronci, Automatic Analysis of the NRL Pump, *Selected Papers from MEFISTO Project "Formal Methods for Security"* ENTCS **99** (2004) 245–266.

[LMT05] R. Lanotte, A. Maggiolo-Schettini, and A. Troina, A Classification of Time and/or Probability Dependent Security Properties, in *Workshop on Quantitative Aspects of Programming Languages (QAPL05)* (2005) 33–47.

[Lav83] S.S. Lavenberg editor, *Computer Performance Modeling Handbook* (Academic Press, 1983).

[Lowe02] G. Lowe, Quantifying Information Flow, in *15th Computer Security Foundation Workshop (CSFW'02)* (IEEE CS Press, 2002) 18–31.

[Mea03] C. Meadows, What Makes a Cryptographic Protocol Secure? The Evolution of Requirements Specification in Formal Cryptographic Protocol Analysis, in *12th European Symposium on Programming Languages and Systems (ESOP'03)* LNCS **2618** (Springer Verlag, 2003) 10–21.

[Mil89] R. Milner, *Communication and Concurrency* (Prentice Hall, 1989).

[MK94] I.S. Moskowitz and M.H. Kang, Covert Channels – Here to Stay? in *9th Conf. on Computer Assurance (Compass'94)* (National Institute of Standards and Technology, 1994) 235–244.

[OSS02] G. On, J. Schmitt, and R. Steinmetz, On Availability QoS for Replicated Multimedia Service and Content, in *Joint Workshops on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems (IDMS-PROMS'02)* LNCS **2515** (Springer Verlag, 2002) 313–326.

[R+01] P.Y.A. Ryan, J. McLean, J. Millen, and V. Gligor, Non-interference: Who Needs It?, in *14th Computer Security Foundations Workshop (CSFW'01)* (IEEE CS Press, 2001) 237–238.

[SWD98] A. Simpson, J. Woodcock, and J. Davies, Safety through Security in *9th Workshop on Software Specification and Design (IWSSD-9)* (IEEE CS Press, 1998) 18–24.

[Ste94] W.J. Stewart, *Introduction to the Numerical Solution of Markov Chains* (Princeton University Press, 1994).

[SD98] V. Stravridou and B. Dutertre, From Security to Safety and Back, in *2nd Workshop on Computer Security, Fault Tolerance, and Software Assurance: From Needs to Solutions*, Williamsburg, VA (1998).