

Security and dynamic compositions of open systems

A. Bracciali* A. Brogi* G. Ferrari† E. Tuosto†

Dipartimento di Informatica
Università di Pisa
Corso Italia 40, I-56125
Italia

Abstract *Designing software by coordinating components is becoming a pervasive software development methodology. This practice of building (distributed) applications is currently supported by several industrial standards competing in the marketplace. Moreover, Internet facilitates the distributions of services to be embedded into applications. In this highly dynamic scenario, we discuss a methodology to formally describe the (behavioural) features of the single components and to reason about the properties of the assembled applications.*

Keywords: Formal Methods, Component Composition, Dynamic Aspects of Coordination, Property Verification.

1 Introduction

Open systems can be thought of as systems which can deal with dynamic addition/replacement of components, modification of the communication infrastructures without compromising the overall system behaviours. They can be built by composition of a set of components having a clear defined interface that may vary during time. The openness of the system manifests itself also in the computational environment: components interact without being able of making complete trustiness

*Author supported by the EU IST project *SOCS* IST-2001-32530

†Author supported by the EU project *PROFUNDIS* and the MIUR project *NAPOLI*

assumptions on the behaviour of all the other (possibly malicious) components.

The World Wide Web provides a paradigmatic example of an environment where open systems may be designed, developed and distributed. Current software technologies emphasize the notion of WEB SERVICE as a key idiom to control the design and the development of applications. Conceptually, WEB SERVICES [10] are stand-alone components that reside over the nodes of the network. Each WEB SERVICE has an interface which is network accessible through standard network protocols and describes the interaction capabilities of the service (e.g., the message format). Applications are developed by combining and integrating together WEB SERVICES, which do not have pre-existing knowledge of how they will interact with each other.

The stages of WEB SERVICE development can be summarized as follows:

1. *publishing* the services,
2. *finding* the required service,
3. *binding* the services inside the application,
4. *running* the application through the Web.

The use of WEB SERVICES, namely open systems over the World Wide Web, raises a number of interesting issues. First, security should be ensured: WEB SERVICES belonging to different service providers meet different security requirements. Second, dynamic adaptability

should be ensured: applications can reconfigure their structure (e.g. the security policy) and their WEB SERVICES at run-time to respond to dynamic changes of the network environment (new services may be offered and existing services may disappear and reappear later with more functionalities).

Current technologies provide limited solutions to some of the above issues. For instance, the Microsoft .NET architecture supplies a programming technology embodying general facilities for handling heterogeneity, but the coordination of WEB SERVICES is subtle and error-prone since one has to deal with the visible interfaces, the message formats, the underlying communication infrastructures and the application requirements such as security and quality of services.

The research activities in the field of coordination languages and models have improved the formal understanding of dynamically adaptable mobile components. However, a full-fledged foundational model is still missing. Some preliminary results in this direction can be found in [2] where a discipline for orchestrating WEB SERVICES is outlined. The approach of [2] is based on the idea of separating WEB SERVICE providers from *contracts* mechanisms (also known under the name of *connectors*), which regulate WEB SERVICE coordination.

Our goal is to contribute to providing a formal understanding of WEB SERVICE coordination, as a first step towards the development of proof techniques for the automated verification and certification of properties of web applications. Moreover, a formal understanding of WEB SERVICES may suggest the design of robust coordination patterns.

In this extended abstract we provide an informal introduction to the model. In particular, we discuss and motivate the main design issues, and we highlight some feature of the model. The proposed methodology has been applied [6, 5] to the static analysis of security protocols in a dynamically evolving computational environment.

The paper is organized as follows. In Section

2 we introduce a formal model for WEB SERVICE composition. The model exploits the notion of *naming* to handle coordination. Hence, the model takes the form of a name-passing calculus where WEB SERVICES are composed together by appropriately sharing their network references. Behavioural properties of applications are formulated as invariants that the application is required to guarantee. Only WEB SERVICES satisfying the invariants can join the application. For example, absence of deadlocks can be formulated as a notion of “up to now” correctness of an open session of the application: A new WEB SERVICE will not be admitted to the session if it can introduce a deadlock that will not be removed by any other WEB SERVICE possibly joining the session. In Section 3 we illustrate how the methodology facilitates reasoning about security properties. Finally, in Section 4 we argue how the methodology can support the validation of more general properties of the coordination of WEB SERVICES.

2 Interaction Patterns

The *Interaction Patterns* calculus (IP calculus) [7] has been specifically designed to describe composition of software components. The behaviour of a component is described by an *interaction pattern* $C = (X)[E]$ which consists of a behavioural expression E and of a set of references X , called *open variables*, that the component makes available to any environment in which it is executed. We require that an interaction pattern will be name disjoint from other interaction patterns.

Indeed, the IP calculus is a finitary fragment of the π -calculus [12] with the addition of variables and without recursion. We argue that this hypothesis may be assumed when dealing with component composition because interaction among components usually happens following recurrent finite patterns. Moreover, this simplification leads to a formal framework suitable for locally and uniformly reasoning on properties of both static and dynamic compo-

sitions.

A *session* \mathcal{S} is a set of (partially connected) interaction patterns. An Interaction pattern C accesses a session by means of a *join* operation $J(\mathcal{S}, \gamma, C)$. The join operation connects some of the open variables of C and \mathcal{S} , according to the declaration (name sharing) given by the binding γ . Instantiated variables are not open anymore. By communicating channel names, the connection topology of a session may dynamically change. A session whose interaction patterns do not have open variables is called a *closed* session, otherwise it is *open*. A pattern joining a closed session cannot interfere with the patterns already in the session.

Let us consider two simple interaction patterns. In the examples the combinator “+” denotes non deterministic choice, “and ‘.’” denotes sequential composition. Moreover, x, y, w are variables, c is a channel name and **abort**, **query** are basic data.

The first interaction pattern S describes a simple server, which offers to the environment a single channel (the open variable x). Once connected to a client, it can either send a reference to a service (the channel name c) or it can be terminated by receiving an interruption (**abort**). The second pattern C is a “compatible” client. It also offers one open variable (y). However, it may either receive dates or “autonomously” commit to abort:

$$\begin{aligned} S &= (x) [\text{out}(x, c) + \text{in}(x, \text{abort})] \\ C &= (y) [\tau.\text{in}(y, w) + \tau.\text{out}(y, \text{abort})] \end{aligned}$$

The internal action τ is used to model local choices: the choice $\tau._ + \tau._$ is not determined by an external synchronization, but by one of the two internal steps.

The two patterns join an (empty) session, by $J(\emptyset, [x \mapsto d, y \mapsto d], \{S, C\})$, yielding a closed session that never deadlocks:

$$\left\{ \begin{array}{l} () [\text{out}(d, c) + \text{in}(d, \text{abort})], \\ () [\tau.\text{in}(d, w) + \tau.\text{out}(d, \text{abort})] \end{array} \right\}$$

Besides the static incremental construction of a system, the same connecting mechanisms also models the dynamic interplay of

components that intend to join a running environment. Let us consider an open session containing a client that, after receiving the datum, forwards it along an open variable (k):

$$\left\{ (y, k) [\begin{array}{l} \tau.\text{in}(y, w).\text{out}(k, w) \\ + \tau.\text{out}(y, \text{abort}) \end{array}] \right\}$$

After the server S has joined the session, the session is still open.

Let us now consider the case where the session is joined, by exploiting the same mapping, by a server not capable of handling abort events S' :

$$\left\{ \begin{array}{l} () [\text{out}(d, c)], \\ (k) [\tau.\text{in}(d, w).\text{out}(d, w) + \tau.\text{out}(d, \text{abort})] \end{array} \right\}$$

Despite any pattern that will join the session, it appears as a session that may run in trouble in some points of its evolution. In fact, if the client commits to send **abort**, the session evolves in the permanently deadlocked session S'' :

$$\left\{ () [\text{out}(d, c)], (k) [\text{out}(d, \text{abort})] \right\}$$

where both the patterns are trying to send over a channel, but, since the channel is not anymore open, no other pattern will receive their messages.

Summing up, this short example shows how it is possible to state strong (correctness) properties about closed systems. It also points out how the same model can uniformly deal with open systems, for which it is possible to state weaker (correctness) properties, as acceptability [7], depending on the so far specified system.

3 Verifying security protocols

Cryptographic (security) protocols consist of a number of participants, called *principals*, which behave following a finite and deterministic rôle. Principal instances interact together, possibly interleaving more instances of the same protocol, by appropriately sharing their cryptographic keys. A protocol is hence

verified against the presence of an *intruder*, which, intercepting potentially all the communications exchanged by principals in an untrusted environment, violates a security property of the protocol, [9].

The IP calculus has been extended to deal with secure composition of components. The IP approach to protocol verification can be roughly summarized as follows:

- (a) Protocol principals are represented as interaction patterns;
- (b) Security properties are specified by logical formulas ϕ ;
- (c) An automatic procedure verifies whether there exists a session, formed by (multiple) principal instances together with an intruder, having a trace which invalidates the security property ϕ .

Hereafter, we will briefly discuss these points. As a running example we will consider the Needham-Shroeder public key protocol (NS). The NS protocol is used to authenticate a principal A to a principal B . In the protocol principal A speaks first by sending to principal B a nonce na and its name encrypted with the public key of B : $\{na, A\}_{B^+}$. Principal B replies encrypting na and a new nonce nb with the public key of A : $\{na, nb\}_{A^+}$. Principal A confirms to be the principal who started the protocol with B and has received the nonce nb generated by B by sending $\{nb\}_{B^+}$. The standard specification of the NS protocol is given below.

$$\begin{aligned} A &\rightarrow B : \{na, A\}_{B^+} \\ B &\rightarrow A : \{na, nb\}_{A^+} \\ A &\rightarrow B : \{nb\}_{B^+} \end{aligned}$$

- (a) The IP calculus has been extended with spi-calculus-like [1] cryptographic primitives. Communication happens along an untrusted channel that can be accessed by any of the principals. Only the principals

which have the correct key can read an encrypted message. The join operation models the sharing of keys. The IP calculus specification of the NS protocol is:

$$\begin{aligned} A &= (y)[\text{out}(\{na, A\}_{y^+}).\text{in}(\{na, u\}_{A^-}).\text{out}(\{u\}_{y^+})] \\ B &= ()[\text{in}(\{x, z\}_{B^-}).\text{out}(\{x, nb\}_{z^+}).\text{in}(\{nb\}_{B^-})] \end{aligned}$$

Notice that in the NS specification the pattern for principle B exploits the open variable y to acquire the public key of (an instance) of A .

To distinguish multiple instances in the same session, principal instances are obtained by assigning a unique index to the corresponding pattern. For instance, assume that session \mathcal{S} contains only an instance A_1 of the principal A . An instance B_2 of B joins \mathcal{S} , by the operation $J(\mathcal{S}, [y_2 \mapsto A_1^+, B_2])$:

$$\begin{aligned} \{ & () [\text{out}(na_1, \{A_1\}_{B_2^+}). \text{in}(\{na_1, u_1\}_{A_1^-}). \\ & \text{out}(\{u_1\}_{B_2^+})], \\ & () [\text{in}(\{x_2, z_2\}_{B_2^-}). \text{out}(\{z_2, nb_2\}_{z_2^+}). \\ & \text{in}(\{nb_2\}_{B_2^-})] \} \end{aligned}$$

The join operation keeps track of the sharing of keys among the principals involved in the protocol sessions.

- (b) The logic allows one to make statements about values exchanged, variables and their relations. The key issue of the approach is that the logic makes explicit the indexing of instances and quantifiers range over such indexes.

In the logical language, integrity properties are expressed by predicating about values of variables. For instance, the formula $\forall i. x_i = na_i$ reads as “for all instances indexed with i containing x_i , at the end of the protocol, the value of x_i must be na_i ”.

Similarly, secrecy properties can be expressed by explicitly mentioning the knowledge κ that the intruder may have acquired by intercepting data exchanged in the protocol, e.g. $\forall i. \neg(na_i \in \kappa)$.

Finally, authentication properties are expressed as relations among values exchanged and variables, in the style of the correspondence relations, [8, 3].

An authentication property for NS may be expressed by the logical formula $\phi \equiv \forall i. \exists j. x_i = A_j \rightarrow y_j = B_i$, that reads as: “if (a responder) B_i believes that he is talking with (an initiator) A_j , then A_j has intended to talk with B_i ”.

- (c) The verification process considers a session, that can be joined by a bound number of instances of the protocol and by an undefined attacker. By exploring the state space of the possible traces, it is checked if the intruder can drive the session along a trace that violates the security property ϕ . If any, the pattern of the intruder is returned.

Notice that this is a “static” verification, since model-checking the property, i.e. “certifying” the protocol, is done by means of an off-line simulation.

In the case of the NS protocol, with respect to ϕ , the intruder B_0 performing the Lowe attack [11], is returned:

$$B_0 = (y_0) [\text{in}(\{w, w'\}_{B_0^-}) . \text{out}(\{w, w'\}_{y_0^+}) . \\ \text{in}(\{w''\}_{B_0^-}) . \text{out}(\{w''\}_{y_0^+})]$$

4 WEB SERVICES: Coordination and Verification

In our approach, the join mechanism is the basic building block to assemble together highly dynamic systems. It appears hence natural to augment the expressiveness of the join mechanism with constraints enforcing the desired coordination properties web applications must satisfy.

As far as security issues are considered, let us assume that a security policy for a session \mathcal{S} is given as a constraint ψ on the sharing of references. Hence, $J(\mathcal{S}, \gamma, P)$ yields the new

session \mathcal{S}' only if the mapping γ respects the policy ψ , namely $\mathcal{S}' \models_{\gamma} \psi$.

Notice that the relation \models_{γ} is not meant to require a model checking of traces, but rather the checking for the satisfiability of ψ with respect to the mapping γ . Indeed, this can be dynamically verified efficiently.

Examining the intruder of the NS example, one could argue that the attack is made possible by an erroneous, unintended sharing of keys. In other words, by an improper use of the functionalities, of the components involved in the protocol. This observation can be used for avoiding the exhaustive analysis of all the possible traces of a protocol session joined by a new principal. Instead, access will be granted on the basis of a constraint on the key sharing.

For example, the constraint $\phi_* \equiv \exists i, j. y_i = B_j \rightarrow \forall h. (y_j \neq A_h \vee y_j \neq B_h)$ reads “any responder B_j (whose public key has been linked to the open variable y_i of another principal) cannot require any public key to be linked to its open variable y_j ”. By associating this constraint to all the mappings of the session, any principal attempting to concurrently playing more rôles in the protocol would be rejected. Hence the NS intruder would not be able to join the session. Notice that ϕ_* is “more restrictive” than ϕ , since also non-malicious components can be refused.

Interesting properties of WEB SERVICE coordination can be formally expressed by exploiting constraints over the sharing of references. For example, the interaction pattern of a WEB SERVICE offering a generic chat service may be thought as

$$\{ (x_1, \dots, x_n) [\dots \text{ chat room } \dots] \}$$

where x_1, \dots, x_n give the chat connections. The chat manager aims at preventing users to directly communicate each other, by refusing users that provide (and hence know) a connection already in the session. This property can be enforced by a constraint of the form $\psi \equiv \forall i, j. x_i \neq x_j$.

We can also equip WEB SERVICE specifications with logical formulas expressing non-functional requirements of components, like for

instance quality of services (e.g. a component can join a session only if a pool of needed WEB SERVICES can be guaranteed).

5 Concluding Remarks

In this paper we have illustrated a formal model to specify and verify coordination properties of distributed (web) applications. A more detailed presentation, more extensively discussing the features of the framework, can be found in [4]. The novelty of our proposal is given by the combination of the following ingredients:

- A name-passing process calculus is exploited to describe coordination policies of interacting components. In particular the join operator models the dynamic composition of components inside an evolving application (e.g. WEB SERVICES).
- Properties of the coordination policies are specified in a declarative style as constraints (invariants) of the join operation.
- Property verification is reduced to the dynamic solution of the constraint problem associated to the session.

We are currently investigating the expressivity of the logic to formally classify which classes of properties can be stated and checked. Moreover, we plan to exploit efficient constraint solving techniques to improve the effectiveness of the proposed methodology.

References

- [1] M. Abadi and A.D. Gordon. A calculus for cryptographic protocols: The spi-calculus. *Information and Computation*, 148(1):1–70, 1999.
- [2] L. Andrade and J.L. Fiadeiro. Coordination for orchestration. In F. Arbab and C. Talcott, editors, *Proc. 5th International Conference on Coordination Models and Languages (Coordination'02)*, volume 2315 of *LNCS*. Springer, 2002.
- [3] M. Boreale. Symbolic trace analysis of cryptographic protocols. In *28th Colloquium on Automata, Languages and Programming (ICALP)*, LNCS. Springer, July 2001.
- [4] A. Bracciali. *Behavioural patterns and software composition*. PhD thesis, Università di Pisa, Italy, 2002. Forthcoming.
- [5] A. Bracciali, A. Brogi, G. Ferrari, and E. Tuosto. Formal intruder identikit for open security protocols. Available at <http://www.di.unipi.it/~etuosto>, 2001.
- [6] A. Bracciali, A. Brogi, G. Ferrari, and E. Tuosto. Security issues in component-based design. *ENTCS*, 54, 2001.
- [7] A. Bracciali, A. Brogi, and F. Turini. Coordinating interaction patterns. In *Proceedings of the ACM Symposium on Applied Computing, Las Vegas, USA*. ACM, 2001.
- [8] E. Clarke, S. Jha, and W. Marrero. Using state space exploration and a nautural deduction style message derivation engine to verify security protocols. In *IFIP PROCOMET*, 1998.
- [9] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
- [10] IBM Software Group. Web services conceptual architecture. In *IBM White Papers*, 2000.
- [11] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, November 1995.
- [12] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40,41–77, September 1992.