Manuscript Number: SCICO-D-11-00251

Title: Model Checking Adaptive Multilevel Service Compositions

Article Type: Special Issue: FACS 2010

Keywords: processalgebra; non-interference; model-checking; webservices

Corresponding Author: Dr Sabina Rossi,

Corresponding Author's Institution:

First Author: Michele Bugliesi

Order of Authors: Michele Bugliesi; Andrea Marin; Sabina Rossi

Abstract: In this paper we present a logic-based technique for verifying both security and correctness properties of multilevel service compositions. We define modal μ-calculus formulae interpreted over service configurations. Our formulae characterize those compositions which satisfy a non-interference property and are compliant, i.e., are both deadlock and livelock free. Moreover, we use filters as prescriptions of behavior (coercions to prevent service misbehavior) and we devise a model checking algorithm for adaptive service compositions which automatically synthesizes an adapting filter.

Highlights

>We present a technique for verifying security and correctness of multilevel service compositions. **>** We define modal μ-calculus formulae interpreted over service configurations. **>** We use filters as prescriptions of behavior, i.e., coercions to prevent service misbehavior. **>** We devise a model checking algorithm for adaptive service compositions. **>** We study a case-study of a federated login system.

Model Checking Adaptive Multilevel Service Compositions[☆]

M. Bugliesi[a], A. Marin[a], S. Rossi[a,1,*]

[a]Università Ca' Foscari Venezia, via Torino 155, 30172
Venezia (Italy)

Abstract

In this paper we present a logic-based technique for verifying both security and correctness properties of multilevel service compositions. We define modal μ-calculus formulae interpreted over service configurations. Our formulae characterize those compositions which satisfy a non-interference property and are compliant, i.e., are both deadlock and livelock free. Moreover, we use filters as prescriptions of behavior (coercions to prevent service misbehavior) and we devise a model checking algorithm for adaptive service compositions which automatically synthesizes an adapting filter.

Keywords: process algebra, non-interference, model-checking, web services

1. Introduction

Service Oriented Architectures (SOA) provide a software architectural style to connect loosely specified and coupled services that communicate with each other. Simple Object Access Protocol (SOAP)-based Web services are becoming the most common implementation of SOA. They are designed to support interoperable service-to-service interactions over a network. This interoperability is gained through a set of XML-based open standards, such as the *Web Service Definition Language* (WSDL). These standards provide a common approach for defining, publishing, and using web services.

[☆]Work partially supported by the MIUR Project IPODS "Interacting Processes in Open-ended Distributed Systems".

[*]Corresponding author.

Email addresses: `bugliesi@dais.unive.it` (M. Bugliesi), `marin@dais.unive.it` (A. Marin), `srossi@dais.unive.it` (S. Rossi)

[1]Full Contact details: Dipartimento di Scienze Ambientali, Informatica e Statistica (DAIS), via Torino 155 - 30172 Venezia (Italy), Tel. +39-041-2348411 Fax. +39-041-2348419

As is the case of many other applications, the information processed in web services might be commercially sensitive and it is important to protect this information against security threats such as disclosure to unauthorized parties. Indeed, many of the features that make web services attractive, including greater accessibility of data, dynamic connections, and relative autonomy (lack of human intervention) are at odds with traditional security models and controls. Difficult issues and unsolved problems exist, such as protecting confidentiality and integrity of data that is transmitted through web services protocols in service-to-service transactions.

Traditionally, security policies are used to specify the security requirements of a system and to control the access to confidential data and resources (see, e.g., [1, 2]). However, if this approach can be successfully applied to enforce the security requirements of a centralized system, it is less suited to formulate the security needs of a service composition. Indeed, there is no central authority in the web that is able to fix the security labels of all services and data. Rather than manipulating stored data, web services compute requested information from dynamic data available on the net that need to be dynamically classified according to their stored information.

In this paper we specify service compositions in terms of behavioral contracts which provide abstract descriptions of system behaviors by means of terms of some process algebra. Formal theories of contracts have first been introduced in [3], and then further developed along independent lines of research in [4, 5, 6], and in [7, 8].

Based upon such formal descriptions, we propose a notion of *non-interference* [9] for multilevel service compositions. This is motivated by the fact that SOAs are increasingly relying on complex distributed systems that share information with multiple levels of security. In these systems information with mixed security levels is processed and targeted to particular clients. For example, in an e-business system, some data will be privileged (e.g., credit card numbers and medical records) and some data will be public (e.g., stock market quotes). Such systems need to be equipped with appropriate security facilities to guarantee the security requirements (e.g., confidentiality or integrity) of the participants.

In this paper, we present an information flow security model [10, 11, 12, 13] for service compositions to control the flow of confidential data in web services. Security policies are used to specify the security requirements of service components. In order to capture the dynamic nature, heterogeneity and lack of knowledge which are intrinsic features of modern web services, we allow policies to be dynamically specified by the service participants. In our model, for example, customers may formulate their security requirements by dynamically assign types (that are security annotations) to individual service components.

We also consider the property of *compliance* which is widely used in the context of SOA as a formal device to identify well-formed service compositions, those whose interactions are free of synchronization errors. Our notion of compliance (see [7, 8]) is a strong condition that ensures the absence of deadlocks *and* livelocks, and the application setting is that of choreographies: compliant choreographies are those whose computations never get stuck or trapped into infinite loops without chances to exit.

We develop a method for verifying both security and correctness properties of multilevel service compositions based on the use of model-checking techniques [14].

Model checking is a promising formal method for automatically verifying properties of finite-state concurrent systems [14]. A model checker normally performs an exhaustive search of the state space of a system to determine if a particular property, expressed in terms of a logical formula, holds. Given sufficient resources, the procedure will always terminate with a yes/no answer.

We define modal μ-calculus [15] formulae, interpreted over service configurations, characterizing those compositions which satisfy a non-interference property and are compliant, i.e., are both deadlock and livelock free. Indeed, due to its expressiveness and its conciseness the μ-calculus is well-suited to model both the security and the correctness properties we are interested in. A model checker (like, e.g., NCSU Concurrency Workbench) can then be used to simultaneously check non-interference and compliance.

We also develop an algorithm for verifying adaptive service compositions. This is based on the use of filters, introduced in [4], as prescriptions of behaviour (coercions to prevent service misbehaviour). Security and correctness properties for adaptive multilevel service compositions are ensured by the automatic synthesis of an adapting filter.

Finally, we apply these results to a case-study in which a customer receives a security level according to its identity that may be certified by a federated login system. This consists of a set of providers that share an open standard to handle user accounts, such as *OpenId*. Google, facebook, Yahoo!, Wordpress are examples of providers that adhere to OpenId standard. When a web application allows for a federated login, a customer can use the account owned at any of the federated providers to obtain the required service. We show how the calculus presented here allows one to formally describe such a system, and we apply the compliance and non-interference analysis to the model. Particular attention is devoted to interpret the results of such analysis in terms of good developing practices.

1.1. Plan of the paper

The paper is organized as follows: Section 2 introduces the calculus for multilevel service compositions. Section 3 formalizes the notion of non-interference. Section 4 presents characteristic modal μ-calculus formulae for non-interference. Section 5 introduces the notion of compliance and defines a modal μ-calculus formula characterizing it. Section 6 presents an algorithm for adaptive service com-

Type environments	Δ	$::=$	$\emptyset \mid \Delta, u : \varsigma$	$u \in \mathcal{P} \cup \mathcal{V},\ \varsigma \in \Sigma$
Actions	φ	$::=$	$\bar{a}@u \mid a@u$	$a \in \mathcal{A},\ u \in \mathcal{P} \cup \mathcal{V}$
Contracts	σ	$::=$	$\mathbf{1} \mid \mathbf{x} \mid \varphi.\sigma \mid \sigma + \sigma$	
			$\mid \sigma \lfloor_{\Delta}\!\oplus_{\Delta}\!\rfloor \sigma \mid \mathtt{rec}(\mathbf{x})\,\sigma$	
Compositions	C	$::=$	$p[\sigma] \mid C \parallel C$	

Table 1: Syntax

positions. A case study is presented in Section 7. Finally, Section 8 concludes the paper.

2. The Calculus

We represent service contracts as terms of a value-passing CCS-like [16] process calculus that includes recursion and operators for external and internal choice. In the algebra, parallel composition arises in *contract compositions* that we define after [8, 17, 18] as the parallel (and concurrent) composition of a set of principals executing contracts. We presuppose a denumerable set of action names $\mathcal{A}$, ranged over by a, b, c, a denumerable set of principal identities $\mathcal{P}$, ranged over by p, q, r, and a denumerable set of variables $\mathcal{V}$, ranged over by x, y. The actions represent the basic units of observable behavior of the underlying services, while the principal names specify the peers providing the services.

In order to specify multilevel service compositions, we assign security levels to principal identities and express both contracts and compositions as typed terms of our calculus. Formally, we assume a complete lattice $\langle \Sigma, \preceq \rangle$ of security annotations, ranged over by ς, ϱ, where $\top$ and $\bot$ represent the top and the bottom elements of the lattice. We denote by $\sqcup$ and $\sqcap$ the join and meet operators over Σ, respectively. Type environments are used to assign security levels to principals. A type environment Δ is a finite mapping from principals and variables to security annotations. Closed type environments, ranged over by Γ, assign security levels to principals only, i.e., they do not contain variables in their domain. We define $\Gamma_1 \sqcup \Gamma_2$ (resp., $\Gamma_1 \sqcap \Gamma_2$) the type environment Γ such that $\Gamma(p) = \Gamma_i(p)$ if $p \notin dom(\Gamma_1) \cap dom(\Gamma_2)$ and $p \in dom(\Gamma_i)$, and $\Gamma(p) = \Gamma_1(p) \sqcup \Gamma_2(p)$ (resp., $\Gamma_1(p) \sqcap \Gamma_2(p)$) otherwise.

2.1. Syntax

The syntax of our calculus is presented in Table 1. Contracts have the following form. Term $\mathbf{1}$ denotes a contract that has reached a successful state. The contract

$\bar{a}@p.\sigma$ describes a service that sends a message on a to principal p and then behaves as σ; syntactically, the principal identity p may be a variable, but it must be a name when the prefix is ready to fire. Dually, the input prefix $a@u.\sigma$ waits for an input on a from a particular/any principal and then continues as σ. If u is a variable x, then the input form is a binder for x with scope σ: upon synchronization with a principal p, x gets uniformly substituted by p in σ. The contract $\sigma + \sigma'$ denotes an external choice, guided by the environment. The contract $\sigma\lfloor_{\Delta}\oplus\ _{\Delta'}\rfloor\sigma'$ represents the internal choice between σ in the type environment Δ and σ' in the type environment Δ' made irrespective of the structure of the interacting components. Syntactically, Δ and Δ' may contain variables in their domain, but they must be closed when the internal choice is ready to fire. The internal choice operator we adopt in this paper allows us to model the fact that a principal may dynamically change (upgrade) the security level of his interactions with other service components through the specific type environment associated with each choice. This will be explained in the definition of the semantics of our calculus and Example 2.1. Finally, $\mathtt{rec}(\mathbf{x})\,\sigma$ makes it possible to express iteration in the contract language. As usual, we assume a standard contractivity condition for recursion, requiring that recursive variables be guarded by a prefix.

Given a principal $p \in \mathcal{P}$, we say that a contract σ is *p-compatible* if for all $\bar{a}@q$ and $a@q$ occurring in σ, q is different from p.

A composition $p_1[\sigma_1] \parallel \cdots \parallel p_n[\sigma_n]$ of principals must be *well-formed* [17] to constitute a legal composition, namely: (i) the principal identities p_i's must all be pairwise different, and (ii) each contract σ_i, executed by principal p_i, is p_i-compatible. If $C = p_1[\sigma_1] \parallel \cdots \parallel p_n[\sigma_n]$ is a legal composition, we say that C is a $\{p_1, \ldots, p_n\}$-composition (dually, that $\{p_1, \ldots, p_n\}$ are the underlying principals for C). Throughout, we assume that contracts are closed (they have no free variables) and that compositions are well-formed. Also, we often omit trailing $\mathbf{1}$'s. We say that $\Gamma \rhd C$ is a *configuration* if Γ is a type environment and C is a well-formed $\{p_1, \ldots, p_n\}$-service composition such that $\{p_1, \ldots, p_n\} \subseteq dom(\Gamma)$.

2.2. Semantics

We define the dynamics of typed service compositions in terms of labelled transition systems (and a success predicate), with rules reported in Table 2. In the table, and in the whole paper, λ ranges over visible contract typed actions $\bar{a}@p$, $a@p$ and silent actions τ; δ ranges over service composition actions $a_{p\to q}$, $\bar{a}_{p\to q}$ and τ.

The first block of rules defines the successful states of a contract and a composition, which are those that expose the successful term $\mathbf{1}$ at top level, or immediately under an external choice (up-to recursive unfoldings). Notice that a composition is successful only when all its components are successful.

Contract and composition satisfaction: $\sigma \checkmark$

$$\mathbf{1}\checkmark \qquad \frac{\sigma_i \checkmark}{\sigma_1 + \sigma_2 \checkmark} \qquad \frac{\sigma\{\mathbf{x} := \mathtt{rec}(\mathbf{x})\,\sigma\}\checkmark}{\mathtt{rec}(\mathbf{x})\,\sigma \checkmark} \qquad \frac{\sigma \checkmark}{p[\sigma]\checkmark} \qquad \frac{C_1 \checkmark \quad C_2 \checkmark}{C_1 \parallel C_2 \checkmark}$$

Typed contract transitions: $\Gamma \triangleright \sigma \xrightarrow{\lambda} \Gamma' \triangleright \sigma'$

$$\Gamma \triangleright a@p.\sigma \xrightarrow{a@p} \Gamma \triangleright \sigma \qquad \Gamma \triangleright a@x.\sigma \xrightarrow{a@p} \Gamma \triangleright \sigma\{x := p\}$$

$$\Gamma \triangleright \bar{a}@p.\sigma \xrightarrow{\bar{a}@p} \Gamma \triangleright \sigma \qquad \Gamma \triangleright \sigma_1 \lfloor_{\Gamma_1} \oplus {}_{\Gamma_2}\rfloor \sigma_2 \xrightarrow{\tau} \Gamma \sqcup \Gamma_i \triangleright \sigma_i \quad (i = 1, 2)$$

$$\frac{\Gamma \triangleright \sigma_i \xrightarrow{\lambda} \Gamma' \triangleright \sigma}{\Gamma \triangleright \sigma_1 + \sigma_2 \xrightarrow{\lambda} \Gamma' \triangleright \sigma}\,(i = 1, 2) \qquad \frac{\Gamma \triangleright \sigma\{\mathbf{x} := \mathtt{rec}(\mathbf{x})\,\sigma\} \xrightarrow{\lambda} \Gamma' \triangleright \sigma'}{\Gamma \triangleright \mathtt{rec}(\mathbf{x})\,\sigma \xrightarrow{\lambda} \Gamma' \triangleright \sigma'}$$

Typed composition transitions: $\Gamma \triangleright C \xrightarrow{\delta} \Gamma' \triangleright C'$

$$\frac{\Gamma \triangleright \sigma \xrightarrow{a@p} \Gamma \triangleright \sigma'}{\Gamma \triangleright q[\sigma] \xrightarrow{a_{p\to q}} \Gamma \triangleright q[\sigma']}\; p \in dom(\Gamma),\; p \neq q \qquad \frac{\Gamma \triangleright \sigma \xrightarrow{\bar{a}@p} \Gamma \triangleright \sigma'}{\Gamma \triangleright q[\sigma] \xrightarrow{\bar{a}_{q\to p}} \Gamma \triangleright q[\sigma']}\; p \neq q$$

$$\frac{\Gamma \triangleright C_1 \xrightarrow{a_{p\to q}} \Gamma \triangleright C_1' \quad \Gamma \triangleright C_2 \xrightarrow{\bar{a}_{p\to q}} \Gamma \triangleright C_2'}{\Gamma \triangleright C_1 \parallel C_2 \xrightarrow{\tau} \Gamma \triangleright C_1' \parallel C_2'}$$

$$\frac{\Gamma \triangleright \sigma \xrightarrow{\tau} \Gamma' \triangleright \sigma'}{\Gamma \triangleright p[\sigma] \xrightarrow{\tau} \Gamma' \triangleright p[\sigma']}\; \Gamma(q) \prec \Gamma'(q) \preceq \Gamma(p),\; \forall q \in dom(\Gamma') : \Gamma(q) \neq \Gamma'(q)$$

$$\frac{\Gamma \triangleright C_1 \xrightarrow{\delta} \Gamma' \triangleright C_1'}{\Gamma \triangleright C_1 \parallel C_2 \xrightarrow{\delta} \Gamma' \triangleright C_1' \parallel C_2}$$

Table 2: Typed contract and composition transitions

The second block of rules defines the typed transitions for contracts, and are mostly self-explanatory. The rule for the internal choice ensures that a service component cannot downgrade the security level of other principals. Intuitively, this corresponds to the fact that downgrading the security level of a principal would mean downgrading the level of confidentiality of the information it manages. Notice that the type environments (Γ_1 and Γ_2) that are arguments of the internal choice are required to be closed when the choice is ready to fire.

Each typed contract transition yields a corresponding transition for the principal hosting the contract. The typed transitions for configurations are relative to the underlying set $dom(\Gamma)$ of principals. τ transitions for configurations arise from the execution of a contract internal choice. In this case, the principal performing the choice may modify the security level of its interactions with other components by assigning different security levels to the principals with which it is going to interact. In the rule we assume that a principal p cannot upgrade the level of its interactions with other components above its own level. This is the meaning of the condition associated with the rule for τ-typed composition transitions. This is the only constraint on the principal security levels we assume. Such a constraint ensures that information does not explicitly flow from high to low, but it does not deal with implicit flows. Instead, we will characterize non-interference in terms of the actions that typed service compositions may perform.

We will use the following shorthands. We write $\Longrightarrow$ to denote the reflexive and transitive closure of $\xrightarrow{\tau}$, and $\overset{\delta}{\Longrightarrow}$ for $\Longrightarrow\xrightarrow{\delta}\Longrightarrow$. We extend sequence of actions $w = \delta_1 \ldots \delta_n$, we write $\overset{w}{\Longrightarrow}$ to note $\overset{\delta_1}{\Longrightarrow} \cdots \overset{\delta_n}{\Longrightarrow}$. A computation for a configuration $\Gamma \rhd C$, is a sequence $\Gamma \rhd C = \Gamma_0 \rhd C_0 \xrightarrow{\tau} \Gamma_1 \rhd C_1 \xrightarrow{\tau} \ldots$ of internal actions.

Lemma 2.1 (Preserving well-formedness)**.** *Let Γ be a type environment and C be a service composition such that $\Gamma \rhd C$ is a configuration. If $\Gamma \rhd C$ is well-formed and $\Gamma \rhd C \xrightarrow{\tau} \Gamma' \rhd C'$, then $\Gamma' \rhd C'$ is well-formed.*

Proof. The only subtlety is that an output action within principal p, may have a variable, say x as a target. In any closed contract, x must be bound at an enclosing input prefix. Now, given that $\Gamma \rhd C$ is well-formed, that input may never synchronize with an output from p itself: hence x will never get bound to p. $\square$

Example 2.1. *Table 3 shows an example of a service contract composition. Let Σ contain two security annotations, $\mathtt{L}$ (*public*) and H (*confidential*), with $\mathtt{L} \preceq \mathtt{H}$. Let Γ be the type environment $C : \mathtt{H}, T : \mathtt{L}, A_1 : \mathtt{L}, A_2 : \mathtt{L}$. The typed composition $\Gamma \rhd S$ is well-formed and consists of four services: $C[\sigma_C]$, $T[\sigma_T]$ and $A_i[\sigma_A]$ representing a* customer, *a* travel agency, *and two* airline companies,

$$S = C[\sigma_C] \parallel T[\sigma_T] \parallel A_1[\sigma_A] \parallel A_2[\sigma_A]$$

$$\begin{aligned}
\sigma_C = \quad & \overline{\texttt{Req}}@T.\texttt{Lst}@T.(\,\overline{\texttt{Close}}@T.\mathbf{1}\lfloor_{\emptyset}\oplus\,_{T:\texttt{H}}\rfloor(\,\overline{\texttt{Buy1}}@T.\overline{\texttt{Pay}}@T.\\
& \texttt{Get}@A_1.\mathbf{1}\lfloor_{A_1:\texttt{H}}\oplus\,_{A_2:\texttt{H}}\rfloor\overline{\texttt{Buy2}}@T.\overline{\texttt{Pay}}@T.\texttt{Get}@A_2.\mathbf{1}\,)\,)\\[1ex]
\sigma_T = \quad & \texttt{Req}@x.\overline{\texttt{Inq}}@A_1.\overline{\texttt{Inq}}@A_2.\texttt{Price}@A_1.\texttt{Price}@A_2.\overline{\texttt{Lst}}@x.(\,\texttt{Close}@x.\mathbf{1}\\
& +\texttt{Buy1}@x.\overline{\texttt{Ord}}@A_1.\texttt{Pay}@x.\overline{\texttt{Conf}}@A_1.\mathbf{1}\\
& +\texttt{Buy2}@x.\overline{\texttt{Ord}}@A_2.\texttt{Pay}@x.\overline{\texttt{Conf}}@A_2.\mathbf{1})\\[1ex]
\sigma_A = \quad & \texttt{Inq}@x.\overline{\texttt{Price}}@x.(\,\texttt{Ord}@x.\texttt{Conf}@x.\overline{\texttt{Get}}@y.\mathbf{1}+\mathbf{1}\,)
\end{aligned}$$

Table 3: Example of a travel agency

respectively. The elementary actions represent business activities that result in messages being sent or received. For example, the action $\overline{\texttt{Req}}@T$ *undertaken by the customer results in a message being sent to the travel agency. In the example, the customer sends a request to the travel agency which then inquires the airlines to get the prices for the selected route. Each airline responds and the travel agency sends to the customer the list of the best prices. The customer decides whether to close the communication with the travel agency or to buy from one of the airlines. In the latter case the customer decides to assign a high security level (*$\texttt{H}$*) to both the travel agency and the chosen airline company in order to safeguard the confidentiality of the purchasing data. The travel agency orders the ticket from the selected airline and takes a deposit (or a full payment) from the customer. As soon as the airline receives the confirmation of the payment, the ticket is issued to the customer.* □

3. Non-Interference

The concept of *non-interference* [9] has been introduced to formalize the absence of information flow in multilevel systems. In the context of service compositions it demands that public interactions between service components are unchanged as secret communications are varied or, more generally, that the low level behaviour of the service composition is independent of the behaviour of its high components. In this way clients are assured that the data transmitted over the air to a web server remains confidential; in other words, sensitive data cannot be intercepted and understood by eavesdroppers.

The notion of non-interference we are going to introduce is relative to the internal behaviour of service compositions, i.e., we are interested in observing the

Typed internal composition transitions: $\Gamma \rhd C \overset{\alpha}{\hookrightarrow} \Gamma' \rhd C'$

$$\frac{\Gamma \rhd \sigma \xrightarrow{\tau} \Gamma' \rhd \sigma'}{\Gamma \rhd p[\sigma] \overset{\tau}{\hookrightarrow} \Gamma' \rhd p[\sigma']} \qquad \frac{\Gamma \rhd C_1 \overset{\alpha}{\hookrightarrow} \Gamma' \rhd C_1'}{\Gamma \rhd C_1 \parallel C_2 \overset{\alpha}{\hookrightarrow} \Gamma' \rhd C_1' \parallel C_2}$$

$$\frac{\Gamma \rhd C_1 \xrightarrow{a_{p\to q}} \Gamma \rhd C_1' \quad \Gamma \rhd C_2 \xrightarrow{\bar{a}_{p\to q}} \Gamma \rhd C_2'}{\Gamma \rhd C_1 \parallel C_2 \overset{\{a\}_{p\to q}}{\hookrightarrow} \Gamma \rhd C_1' \parallel C_2'}$$

Table 4: Typed internal composition transitions

synchronizations between service components. We thus refine the semantics of compositions in order to help (i) to distinguish a local contract move from a synchronization, and (ii) to identify the principals involved in every synchronization. This is captured by the rules collected in Table 4, where we use the relation $\hookrightarrow$ to represent typed synchronizations between service components. The τ label now indicates an internal action to a service component, while synchronizations between different peers in a composition are represented through a label of the form $\{a\}_{p\to q}$ meaning that principals p and q synchronize on action a. We let α range over the labels $\{a\}_{p\to q}$ and τ. We denote by $\overset{\tau}{\hookrightarrow\!\!\!\!\rightarrow}$ a possible empty sequence of $\overset{\tau}{\hookrightarrow}$ and define $\overset{\{a\}_{p\to q}}{\hookrightarrow\!\!\!\!\rightarrow} \overset{def}{=} \overset{\tau}{\hookrightarrow\!\!\!\!\rightarrow} \overset{\{a\}_{p\to q}}{\hookrightarrow} \overset{\tau}{\hookrightarrow\!\!\!\!\rightarrow}$.

The following lemma relates the two semantics for service compositions, one expressed in terms of $\longrightarrow$ and the other one expressed in terms of $\hookrightarrow$.

Lemma 3.1. *Let Γ be a typed environment and C be a service composition such that $\Gamma \rhd C$ is a configuration.*

- $\Gamma \rhd C \overset{\tau}{\hookrightarrow} \Gamma' \rhd C'$ *if and only if* $C = C_1 \parallel p[\sigma] \parallel C_2$, $C' = C_1 \parallel p[\sigma'] \parallel C_2$ *and* $\sigma \xrightarrow{\tau} \sigma'$*;*
- $\Gamma \rhd C \overset{\{a\}_{p\to q}}{\hookrightarrow} \Gamma \rhd C'$ *if and only if* $C = C_1 \parallel p[\sigma] \parallel C_2 \parallel q[\rho] \parallel C_3$, $C' = C_1 \parallel p[\sigma'] \parallel C_2 \parallel q[\rho'] \parallel C_3$, $\sigma \xrightarrow{\bar{a}@q} \sigma'$ *and* $\rho \xrightarrow{a@p} \rho'$.

Proof. The proof follows by induction on the derivations. $\square$

In order to define our notion of non-interference, we need to be able to distinguish the component interactions at a given security clearance. As transitions are typed, we can assign a security level to them as follows: the level of a synchronization depends on the level of the principals performing it. More precisely, by abuse of notation, we denote by $\Gamma(\{a\}_{p\to q})$ the level of the synchronization $\{a\}_{p\to q}$ in the type environment Γ and define it as:

$$\Gamma(\{a\}_{p\to q}) = \Gamma(p) \sqcap \Gamma(q).$$

Thus a ς-level synchronization between two components is performed by principals whose security clearance is higher or equal to ς.

3.1. Behavioural Observations

We define behavioural observations in terms of equivalences that are parametric with respect to the security level $\varsigma \in \Sigma$ of the behaviour we want to observe. Such equivalences are relations over configurations that equate service compositions exhibiting the same ς-level component interactions.

Our behavioural equivalences are defined as a variant of the notion of *weak bisimulation* [16], an observation equivalence which allows one to observe the non-deterministic structure of the LTSs and focuses only on the observable actions.

In the following, we write $\Gamma_1 =_\varsigma \Gamma_2$ whenever $\{p \in dom(\Gamma_1) |\, \Gamma_1(p) \preceq \varsigma\} = \{p \in dom(\Gamma_2) |\, \Gamma_2(p) \preceq \varsigma\}$.

Definition 3.1 (Weak bisimulation on ς-low actions). Let $\varsigma \in \Sigma$. A *weak bisimulation on ς-low actions* is the largest symmetric relation $\approx_\varsigma$ over configurations such that whenever $\Gamma_1 \rhd C_1 \approx_\varsigma \Gamma_2 \rhd C_2$ with $\Gamma_1 =_\varsigma \Gamma_2$

- if $\Gamma_1 \rhd C_1 \xhookrightarrow{\alpha} \Gamma'_1 \rhd C'_1$ with $\alpha = \tau$ or $\Gamma_1(\alpha) \preceq \varsigma$, then there exist Γ'_2 and C'_2 such that $\Gamma_2 \rhd C_2 \overset{\alpha}{\hookrightarrow\!\!\!\!\rightarrow} \Gamma'_2 \rhd C'_2$ with $\Gamma'_1 \rhd C'_1 \approx_\varsigma \Gamma'_2 \rhd C'_2$ and $\Gamma'_1 =_\varsigma \Gamma'_2$;
- if $\Gamma_1 \rhd C_1 \xhookrightarrow{\alpha} \Gamma'_1 \rhd C'_1$ with $\alpha \neq \tau$ and $\Gamma(\alpha) \not\preceq \varsigma$, then there exist Γ'_2 and C'_2 such that
  - either $\Gamma_2 \rhd C_2 \overset{\alpha}{\hookrightarrow\!\!\!\!\rightarrow} \Gamma'_2 \rhd C'_2$
  - or $\Gamma_2 \rhd C_2 \overset{\tau}{\hookrightarrow\!\!\!\!\rightarrow} \Gamma'_2 \rhd C'_2$

  with $\Gamma'_1 \rhd C'_1 \approx_\varsigma \Gamma'_2 \rhd C'_2$ and $\Gamma'_1 =_\varsigma \Gamma'_2$.

We write $\Gamma \models C_1 \approx_\varsigma C_2$ when $\Gamma \rhd C_1 \approx_\varsigma \Gamma \rhd C_2$. □

$$M = C[\sigma_C] \parallel F_1[\sigma_F] \parallel F_2[\sigma_F] \parallel S[\sigma_S]$$

$$\begin{aligned}
\sigma_C = \;& \overline{\texttt{Inq}}@F_1.\overline{\texttt{Inq}}@F_2.\texttt{Plan}@F_1.\texttt{Plan}@F_2. \\
& (\,\overline{\texttt{Agree}}@F_1.\overline{\texttt{Close}}@F_2.\mathbf{1}\lfloor_{F_1:\texttt{H}} \oplus\; _{F_2:\texttt{H}}\rfloor\overline{\texttt{Agree}}@F_2.\overline{\texttt{Close}}@F_1.\mathbf{1}\,)\,) \\
\sigma_F = \;& \texttt{Inq}@x.\overline{\texttt{LookUp}}@S.\texttt{Quote}@x.\overline{\texttt{Plan}}@C.(\,\texttt{Agree}@x.\mathbf{1} + \texttt{Close}@x.\mathbf{1}\,) \\
\sigma_S = \;& \texttt{LookUp}@x.\overline{\texttt{Quote}}@x.\mathbf{1}
\end{aligned}$$

$$M' = C[\sigma'_C] \parallel F_1[\sigma_F] \parallel F_2[\sigma_F] \parallel S[\sigma_S]$$

$$\begin{aligned}
\sigma'_C = \;& \overline{\texttt{Inq}}@F_1.\overline{\texttt{Inq}}@F_2.\texttt{Plan}@F_1.\texttt{Plan}@F_2. \\
& (\,\overline{\texttt{Close}}@F_2.\overline{\texttt{Agree}}@F_1.\mathbf{1}\lfloor_{F_1:\texttt{H}} \oplus\; _{F_2:\texttt{H}}\rfloor\overline{\texttt{Close}}@F_1.\overline{\texttt{Agree}}@F_2.\mathbf{1}\,)\,)
\end{aligned}$$

Table 5: Example of a financial consulting platform

3.2. Non-interference

The notion of non-interference is inspired by [12] and is expressed in terms of a restriction operator $\cdot|_\varsigma$ which allows one to represent a service composition prevented from performing internal synchronizations of a level higher than ς. The semantics of $C|_\varsigma$ is described by the following rule:

$$\frac{\Gamma \rhd C \xhookrightarrow{\alpha} \Gamma' \rhd C'}{\Gamma \rhd C|_\varsigma \xhookrightarrow{\alpha} \Gamma' \rhd C'|_\varsigma} \; \Gamma(\alpha) \preceq \varsigma$$

Definition 3.2 (Non-interference). Let $\varsigma \in \Sigma$, Γ be a type environment and C be a service composition such that $\Gamma \rhd C$ be a configuration. We say that the service composition C satisfies the non-interference property with respect to the level ς in the type environment Γ, denoted $C \in \mathcal{NI}_{\Gamma,\varsigma}$, if

$$\Gamma \models C \approx_\varsigma C|_\varsigma.$$

□

Example 3.1. *Consider again the service composition S in the type environment Γ of Example 2.1. The property $S \in \mathcal{NI}_{\Gamma,L}$ holds.* □

Example 3.2. *Consider the service composition depicted in Table 5: it consists of a* client *C, two* financial consulting services *F_1 and F_2 and a* stock quote service provider *S. The client inquires the financial services to get investment advices. The financial services consult the stock quote service provider in order to look up information on the financial quotes. Then, the financial services send their investment recommendations to the client which may decide whether or not adhere to the investment plan proposed by one of the financial services and close the connection with the other one.*

Let $\Sigma = \{\mathtt{L}, \mathtt{H}\}$ with $\mathtt{L} \preceq \mathtt{H}$ and Γ be the type environment $C : \mathtt{H}$, $F_1 : \mathtt{L}$, $F_2 : \mathtt{L}$, $S : \mathtt{L}$. In this case we have that $M \notin \mathcal{NI}_{\Gamma,\mathtt{L}}$. Indeed, there is a direct causality between the high level actions $\{\mathtt{Agree}\}_{C\to F_i}$ and the low level action $\{\mathtt{Close}\}_{C\to F_j}$ with $i \neq j$, performed after the clients makes the choice. As a consequence, if the client decides to accept the proposal of F_1 then F_2 knows that the customer has agreed to proceed with investment recommendation of F_1 by just observing that the action $\{\mathtt{Close}\}_{C\to F_2}$ has been performed. The service composition can be made secure by letting $\{\mathtt{Close}\}_{C\to F_j}$ be executed independently of $\{\mathtt{Agree}\}_{C\to F_i}$ as in the composition M' which is obtained from M by replacing the contract σ_C with σ'_C. □

4. Modal Formulae for Non-Interference

In this section we present a method for verifying whether $\Gamma \models C \approx_\varsigma C|_\varsigma$ which consists in defining a modal μ-calculus formula $\phi^{\approx_\varsigma}(\Gamma \rhd C)$ such that $\Gamma' \rhd C'$ satisfies $\phi^{\approx_\varsigma}(\Gamma \rhd C)$ iff $\Gamma \rhd C \approx_\varsigma \Gamma' \rhd C'$.

The modal μ-calculus [15] is a small, yet expressive process logic. We consider modal μ-calculus formulae constructed according to the following grammar:

$$\phi \ ::= \ \mathbf{true} \mid \mathbf{false} \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \langle\alpha\rangle\phi \mid [\alpha]\phi \mid X \mid \mu X.\phi \mid \nu X.\phi$$

where X ranges over an infinite set of variables and α over the labels $\{a\}_{p\to q}$ and τ. The *fixpoint operators* μX and νX bind the variable X and we adopt the usual notion of closed formula. For a finite set M of formulae, we write $\bigwedge M$ and $\bigvee M$ for the conjunction and disjunction of the formulae in M, where $\bigwedge \emptyset = \mathbf{true}$ and $\bigvee \emptyset = \mathbf{false}$.

Modal μ-calculus formulae are interpreted over configurations modelled by LTS's. Let $\Gamma \rhd C$ be a configuration and $LTS(\Gamma \rhd C) = (S_{\Gamma\rhd C}, Act, \hookrightarrow)$ where $S_{\Gamma\rhd C}$ is the set of states reachable from $\Gamma \rhd C$ through $\overset{\alpha}{\hookrightarrow}$ and $\alpha \in Act$ denotes the action $\{a\}_{p\to q}$ or τ. The subset of configurations in $S_{\Gamma\rhd C}$ that satisfy a formula ϕ, noted by $M_{\Gamma\rhd C}(\phi)(\rho)$, is inductively defined in Table 6. ρ is an *environment*, i.e., it is a partial mapping $\rho : Var \not\to 2^{S_{\Gamma\rhd C}}$ that interprets at least

$$
\begin{array}{lcl}
M_{\Gamma\rhd C}(\mathbf{true})(\rho) & = & S_{\Gamma\rhd C} \\
M_{\Gamma\rhd C}(\mathbf{false})(\rho) & = & \emptyset \\
M_{\Gamma\rhd C}(\phi_1 \wedge \phi_2)(\rho) & = & M_{\Gamma\rhd C}(\phi_1)(\rho) \cap M_{\Gamma\rhd C}(\phi_2)(\rho) \\
M_{\Gamma\rhd C}(\phi_1 \vee \phi_2)(\rho) & = & M_{\Gamma\rhd C}(\phi_1)(\rho) \cup M_{\Gamma\rhd C}(\phi_2)(\rho) \\
M_{\Gamma\rhd C}(\langle\alpha\rangle\phi)(\rho) & = & \{\Gamma' \rhd C' \mid \exists\, \Gamma'' \rhd C'' :\ \Gamma' \rhd C' \xhookrightarrow{\alpha} \Gamma'' \rhd C'' \\
 & & \qquad\qquad \wedge\ \Gamma'' \rhd C'' \in M_{\Gamma\rhd C}(\phi)(\rho)\} \\
M_{\Gamma\rhd C}([\alpha]\phi)(\rho) & = & \{\Gamma' \rhd C' \mid \forall\, \Gamma'' \rhd C'' :\ \Gamma' \rhd C' \xhookrightarrow{\alpha} \Gamma'' \rhd C'' \\
 & & \qquad\qquad \Rightarrow\ \Gamma'' \rhd C'' \in M_{\Gamma\rhd C}(\phi)(\rho)\} \\
M_{\Gamma\rhd C}(X)(\rho) & = & \rho(X) \\
M_{\Gamma\rhd C}(\mu X.\phi)(\rho) & = & \bigcap\{x \subseteq S_{\Gamma\rhd C} \mid M_{\Gamma\rhd C}(\phi)(\rho[X \mapsto x]) \subseteq x\} \\
M_{\Gamma\rhd C}(\nu X.\phi)(\rho) & = & \bigcup\{x \subseteq S_{\Gamma\rhd C} \mid M_{\Gamma\rhd C}(\phi)(\rho[X \mapsto x]) \supseteq x\}
\end{array}
$$

Table 6: Semantics of modal mu-calculus

the free variables of ϕ by subsets of $S_{\Gamma\rhd C}$. For a set $x \subseteq S_{\Gamma\rhd C}$ and a variable X, we write $\rho[X \mapsto x]$ for the environment that maps X to x and $Y \neq X$ to $\rho(Y)$ if ρ is defined on Y.

Intuitively, $\mathbf{true}$ and $\mathbf{false}$ hold for all, resp. no, states and $\wedge$ and $\vee$ are interpreted by conjunction and disjunction, $\langle\alpha\rangle\phi$ holds for a configuration $\Gamma' \rhd C' \in S_{\Gamma\rhd C}$ if there exists $\Gamma'' \rhd C''$ reachable from $\Gamma' \rhd C'$ with action α and satisfying ϕ, and $[\alpha]\phi$ holds for $\Gamma' \rhd C'$ if all configurations $\Gamma'' \rhd C''$ reachable from $\Gamma' \rhd C'$ with action α satisfy ϕ. The interpretation of a variable X is as prescribed by the environment. The formula $\mu X.\phi$, called *least fixpoint formula*, is interpreted by the smallest subset x of $S_{\Gamma\rhd C}$ that recurs when ϕ is interpreted with the substitution of x for X. Similarly, $\nu X.\phi$, called *greatest fixpoint formula*, is interpreted by the largest such set. Existence of such sets follows from the well-known Knaster-Tarski fixpoint theorem. As the meaning of a closed formula ϕ does not depend on the environment, we sometimes write $M_{\Gamma\rhd C}(\phi)$ for $M_{\Gamma\rhd C}(\phi)(\rho)$ where ρ is an arbitrary environment.

The set of configurations *satisfying* a closed formula ϕ is defined as $\mathit{Conf}(\phi) = \{\Gamma \rhd C \mid \Gamma \rhd C \in M_{\Gamma\rhd C}(\phi)\}$. We also refer to (closed) *equation systems*:

$$\mathit{Eqn}:\quad X_1 = \phi_1 \quad \ldots \quad X_n = \phi_n$$

where $X_1, \ldots, X_n$ are distinct variables and $\phi_1, \ldots, \phi_n$ are formulae having at most $X_1, \ldots, X_n$ as free variables.

An environment $\rho : \{X_1, \ldots, X_n\} \to 2^{S_{\Gamma\rhd C}}$ is a *solution* of an equation system *Eqn*, if $\rho(X_i) = M_{\Gamma\rhd C}(\phi_i)(\rho)$. The fact that solutions always exist, is again a consequence of the Knaster-Tarski fixpoint theorem. In fact the set of environments that are candidates for solutions, $\mathit{Env}_{\Gamma \rhd C} = \{\rho \mid \rho : \{X_1, \dots, X_n\} \to 2^{S_{\Gamma \rhd C}}\}$, together with the lifting $\sqsubseteq$ of the inclusion order on $2^{S_{\Gamma \rhd C}}$, defined by

$$\rho \sqsubseteq \rho' \text{ iff } \rho(X_i) \subseteq \rho'(X_i) \text{ for } i \in [1..n]$$

forms a complete lattice. Now, we can define the *equation functional* $\mathit{Func}^{\mathit{Eqn}}_{\Gamma \rhd C} : \mathit{Env}_{\Gamma \rhd C} \to \mathit{Env}_{\Gamma \rhd C}$ by $\mathit{Func}^{\mathit{Eqn}}_{\Gamma \rhd C}(\rho)(X_i) = M_{\Gamma \rhd C}(\phi_i)(\rho)$ for $i \in [1..n]$, the fixpoints of which are just the solutions of Eqn. $\mathit{Func}^{\mathit{Eqn}}_{\Gamma \rhd C}$ is monotonic and there is the largest solution $\nu \mathit{Func}^{\mathit{Eqn}}_{\Gamma \rhd C}$ of Eqn (with respect to $\sqsubseteq$), which we denote by $M_{\Gamma \rhd C}(\mathit{Eqn})$. This definition interprets equation systems on the configurations reachable by a given initial configuration $\Gamma \rhd C$. We lift this to configurations by agreeing that a configuration satisfies an equation system Eqn, if its initial state is in the largest solution of the first equation. Thus the set of configurations satisfying the equation system Eqn is $\mathit{Conf}(\mathit{Eqn}) = \{\Gamma \rhd C \mid \Gamma \rhd C \in M_{\Gamma \rhd C}(\mathit{Eqn})(X_1)\}$.

The relation $\approx_\varsigma$ can be characterized as the greatest fixpoint $\nu \mathit{Func}_{\approx_\varsigma}$ of the monotonic functional $\mathit{Func}_{\approx_\varsigma}$ on the complete lattice of relations $\mathcal{R}$ over configurations ordered by set inclusion, where $(\Gamma_1 \rhd C_1, \Gamma_2 \rhd C_2) \in \mathit{Func}_{\approx_\varsigma}(\mathcal{R})$ if and only if points (1) and (2) of Definition 3.1 hold. Thus a relation $\mathcal{R}$ is a weak bisimulation on ς-low actions if and only if $\mathcal{R} \subseteq \mathit{Func}_{\approx_\varsigma}(\mathcal{R})$, i.e., $\mathcal{R}$ is a *post-fixpoint* of $\mathit{Func}_{\approx_\varsigma}$. By the Knaster-Tarski fixpoint theorem, $\nu \mathit{Func}_{\approx_\varsigma}$ is the union of all the post-fixpoints of $\mathit{Func}_{\approx_\varsigma}$, i.e., it is the largest weak bisimulation on ς-low actions. If we restrict to the complete lattice of relations $\mathcal{R} \subseteq S_{\Gamma_1 \rhd C_1} \times S_{\Gamma_2 \rhd C_2}$ we obtain a monotonic functional $\mathit{Func}^{(\Gamma_1 \rhd C_1, \Gamma_2 \rhd C_2)}_{\approx_\varsigma}$ whose greatest fixpoint is exactly $\nu \mathit{Func}_{\approx_\varsigma} \cap (S_{\Gamma_1 \rhd C_1} \times S_{\Gamma_2 \rhd C_2})$, and this is enough to determine if $\Gamma_1 \rhd C_1 \approx_\varsigma \Gamma_2 \rhd C_2$.

Let $\Gamma \rhd C$ be finite-state, $\Gamma_1 \rhd C_1, \dots, \Gamma_n \rhd C_n$ its $|S_{\Gamma \rhd C}| = n$ states, and $\Gamma_1 \rhd C_1 = \Gamma \rhd C$ its initial state. To derive a formula characterizing $\Gamma \rhd C$ up to $\approx_\varsigma$ we construct a *characteristic equation system* [19]:

$$\mathit{Eqn}_{\approx_\varsigma} : X_{\Gamma_1 \rhd C_1} = \phi^{\approx_\varsigma}_{\Gamma_1 \rhd C_1}, \dots, X_{\Gamma_n \rhd C_n} = \phi^{\approx_\varsigma}_{\Gamma_n \rhd C_n}$$

consisting of one equation for each service configuration $\Gamma_1 \rhd C_1, \dots, \Gamma_n \rhd C_n \in S_{\Gamma \rhd C}$. We define the formulae $\phi^{\approx_\varsigma}_{\Gamma_i \rhd C_i}$ such that the largest solution $M_{\Gamma \rhd C}(\mathit{Eqn}_{\approx_\varsigma})$ of $\mathit{Eqn}_{\approx_\varsigma}$ associates the variables $X_{\Gamma_i \rhd C_i}$ just with the states $\Gamma'_i \rhd C'_i$ of $S_{\Gamma \rhd C}$ which are weakly bisimilar on ς-low actions to $\Gamma_i \rhd C_i$, i.e., such that it holds $M_{\Gamma \rhd C}(\mathit{Eqn}_{\approx_\varsigma})(X_{\Gamma_i \rhd C_i}) = \{\Gamma'_i \rhd C'_i \in S_{\Gamma \rhd C} \mid \Gamma_i \rhd C_i {\approx_\varsigma} \Gamma'_i \rhd C'_i\}$. Theorem 4.1 shows the exact form of such formulae. First we define:

$$\langle\!\langle \alpha \rangle\!\rangle_{\Gamma,\varsigma} \phi \stackrel{\text{def}}{=} \begin{cases} \langle\!\langle \alpha \rangle\!\rangle \phi & \text{if } \Gamma(\alpha) \preceq \varsigma \text{ or } \alpha = \tau \\ \langle\!\langle \alpha \rangle\!\rangle \phi \vee \langle\!\langle \tau \rangle\!\rangle \phi & \text{if } \Gamma(\alpha) \not\preceq \varsigma \text{ and } \alpha \neq \tau \end{cases}$$

where $\langle\!\langle\tau\rangle\!\rangle\phi \overset{\text{def}}{=} \mu X.\phi \vee \langle\tau\rangle X$ and $\langle\!\langle\alpha\rangle\!\rangle\phi \overset{\text{def}}{=} \langle\!\langle\tau\rangle\!\rangle\langle\alpha\rangle\langle\!\langle\tau\rangle\!\rangle\phi$. Let $\overset{\alpha}{\lhook\joinrel\twoheadrightarrow}_{\Gamma,\varsigma}$ note either $\overset{\alpha}{\lhook\joinrel\twoheadrightarrow}$ or $\overset{\tau}{\lhook\joinrel\twoheadrightarrow}$. Then $\langle\!\langle\alpha\rangle\!\rangle_{\Gamma,\varsigma}$, $\langle\!\langle\tau\rangle\!\rangle$ and $\langle\!\langle\alpha\rangle\!\rangle$ correspond to $\overset{\alpha}{\lhook\joinrel\twoheadrightarrow}_{\Gamma,\varsigma}$, $\overset{\tau}{\lhook\joinrel\twoheadrightarrow}$ and $\overset{\alpha}{\lhook\joinrel\twoheadrightarrow}$, since

- $M_{\Gamma\rhd C}(\langle\!\langle\alpha\rangle\!\rangle_{\Gamma,\varsigma}\phi)(\rho) = \{\Gamma' \rhd C' \mid \exists\, \Gamma'' \rhd C'' : \Gamma' \rhd C' \overset{\alpha}{\lhook\joinrel\twoheadrightarrow}_{\Gamma,\varsigma} \Gamma'' \rhd C'' \wedge \Gamma'' \rhd C'' \in M_{\Gamma\rhd C}(\phi)(\rho)\}$
- $M_{\Gamma\rhd C}(\langle\!\langle\tau\rangle\!\rangle\phi)(\rho) = \{\Gamma' \rhd C' \mid \exists\, \Gamma'' \rhd C'' : \Gamma' \rhd C' \overset{\tau}{\lhook\joinrel\twoheadrightarrow} \Gamma'' \rhd C'' \wedge \Gamma'' \rhd C'' \in M_{\Gamma\rhd C}(\phi)(\rho)\}$
- $M_{\Gamma\rhd C}(\langle\!\langle\alpha\rangle\!\rangle\phi)(\rho) = \{\Gamma' \rhd C' \mid \exists \Gamma'' \rhd C'' : \Gamma' \rhd C' \overset{\alpha}{\lhook\joinrel\twoheadrightarrow} \Gamma'' \rhd C'' \wedge \Gamma'' \rhd C'' \in M_{\Gamma\rhd C}(\phi)(\rho)\}$.

Theorem 4.1. *Let $\phi^{\approx_\varsigma}_{\Gamma_i\rhd C_i}$ be the formula*

$$\begin{aligned}&\bigwedge\{\bigwedge\{\langle\!\langle\alpha\rangle\!\rangle_{\Gamma,\varsigma} X_{\Gamma'_i\rhd C'_i} \mid \Gamma_i \rhd C_i \overset{\alpha}{\hookrightarrow} \Gamma'_i \rhd C'_i\}\}\\ &\wedge \bigwedge\{[\alpha]\bigvee\{X_{\Gamma'_i\rhd C'_i} \mid \Gamma_i \rhd C_i \overset{\alpha}{\lhook\joinrel\twoheadrightarrow}_{\Gamma,\varsigma} \Gamma'_i \rhd C'_i\}\}.\end{aligned}$$

Then $M_{\Gamma\rhd C}(\mathit{Eqn}_{\approx_\varsigma})(X_{\Gamma_i\rhd C_i}) = \{\Gamma'_i \rhd C'_i \in S_{\Gamma\rhd C} \mid \Gamma_i \rhd C_i \approx_\varsigma \Gamma'_i \rhd C'_i\}$.

Proof. The proof is standard [19] and is based on the observation that $\mathit{Env}_{\Gamma\rhd C}$, the set of candidates for solutions of $\mathit{Eqn}_{\approx_\varsigma}$ is order-isomorphic to $2^{S_{\Gamma\rhd C}\times S_{\Gamma\rhd C}}$, the set of relations that are candidates to be weak bisimulation on ς-low actions between $S_{\Gamma\rhd C} \times S_{\Gamma\rhd C}$. Actually, the mapping $\alpha : \mathit{Env}_{\Gamma\rhd C} \to 2^{S_{\Gamma\rhd C}\times S_{\Gamma\rhd C}}$ defined by:

$$\alpha(\rho) = \{(\Gamma_i \rhd C_i, \Gamma'_i \rhd C'_i) \in S_{\Gamma\rhd C} \times S_{\Gamma\rhd C} \mid \Gamma'_i \rhd C'_i \in \rho(X_{\Gamma_i\rhd C_i})\}$$

is an order isomorphism between $\mathit{Env}_{\Gamma\rhd C}$ and $2^{S_{\Gamma\rhd C}\times S_{\Gamma\rhd C}}$, the inverse of which is the mapping $\beta : 2^{S_{\Gamma\rhd C}\times S_{\Gamma\rhd C}} \to \mathit{Env}_{\Gamma\rhd C}$ defined by $\beta(\mathcal{R})(X_{\Gamma_i\rhd C_i}) = \{\Gamma'_i \rhd C'_i \in S_{\Gamma\rhd C} \mid (\Gamma_i \rhd C_i, \Gamma'_i \rhd C'_i) \in \mathcal{R}\}$.

The proof follows by showing that $\mathit{Func}_{\approx_\varsigma}$ and $\mathit{Func}^{\mathit{Eqn}_{\approx_\varsigma}}_{\Gamma\rhd C}$ are equal up to the isomorphism induced by (α, β), i.e., such that

$$\mathit{Func}^{\mathit{Eqn}_{\approx_\varsigma}}_{\Gamma\rhd C} = \beta \circ \mathit{Func}_{\approx_\varsigma} \circ \alpha.$$

Then their largest fixpoints are also related by the isomorphism, which yields $M_{\Gamma\rhd C}(\mathit{Eqn}_{\approx_\varsigma})(X_{\Gamma_i\rhd C_i}) = \{\Gamma'_i \rhd C'_i \in S_{\Gamma\rhd C} \mid \Gamma_i \rhd C_i \approx_\varsigma \Gamma'_i \rhd C'_i\}$. □

Characteristic formulae, i.e., *single* formulae characterizing configurations can be constructed by applying simple semantics-preserving transformation rules on equation systems. These rules are similar to the ones used by A. Mader in [20] as a mean of solving Boolean equation systems (with alternation) by Gauss elimination. Hence, since for any equation system Eqn there is a formula ϕ such that $Conf(Eqn) = Conf(\phi)$, we obtain that:

Theorem 4.2. *For any finite-state configuration $\Gamma \rhd C$ there is a modal μ-calculus formula $\phi^{\approx_\varsigma}(\Gamma \rhd C)$ such that $Conf(\phi^{\approx_\varsigma}(\Gamma \rhd C)) = \{\Gamma' \rhd C' \in S_{\Gamma \rhd C} \mid \Gamma' \rhd C' \approx_\varsigma (\Gamma' \rhd C')|_\varsigma\}$, that is the set of all the states reachable from $\Gamma \rhd C$ and satisfying $\mathcal{NI}_{\Gamma,\varsigma}$* $\square$

5. A Modal Formula for Compliance

Compliance is a basic property that characterizes the correct behavior of concurrent distributed systems. It is used widely in the context of SOA as a formal device to identify well behaving service compositions, those whose interactions are free of synchronization errors.

In this paper we refer to the notion of compliance for contract service compositions studied in [21]. Intuitively, a composition of services is compliant if it is deadlock and livelock free, i.e., it does not get stuck nor does it get trapped into infinite loops with no exit states. This notion is independent of the security levels of the principals involved in the component synchronizations. Therefore, we omit trailing type environments in the definitions below, and write, e.g., $C \Longrightarrow C'$ to denote a transition of the form $\Gamma \rhd C \Longrightarrow \Gamma' \rhd C'$ for some type environments Γ and Γ'.

Definition 5.1 (Compliance). Let C be a contract service composition. We say that C is *compliant*, noted $C\!\downarrow$, if for every C' such that $C \Longrightarrow C'$ there exists C'' such that $C' \Longrightarrow C''$ and $C''\checkmark$. $\square$

In other words, the notion of compliance ensures that at each intermediate step of the computation in a service composition, each component has a way to reach a successful state (either autonomously, or via synchronizations). This is enough to avoid both deadlocks and livelocks.

Example 5.1. *Consider the service composition S of Example 2.1. It holds that $\Gamma \rhd S$ is both compliant, i.e., $S\!\downarrow$, and non interfering, i.e., $S \in \mathcal{NI}_{\Gamma,L}$.* $\square$

The notion of compliance can be equivalently expressed in terms of $\xhookrightarrow{\alpha}$ where α denotes a synchronization $\{a\}_{p \to q}$ or τ. More precisely, let $\gamma = \alpha_1, \ldots, \alpha_n$. We denote by $\overset{\gamma}{\hookrightarrow\!\!\!\!\rightarrow}$ the sequence of transitions $\overset{\alpha_1}{\hookrightarrow\!\!\!\!\rightarrow} \overset{\alpha_2}{\hookrightarrow\!\!\!\!\rightarrow} \ldots \overset{\alpha_n}{\hookrightarrow\!\!\!\!\rightarrow}$. Again we write $C \overset{\gamma}{\hookrightarrow\!\!\!\!\rightarrow} C'$ to denote a derivation $\Gamma \rhd C \overset{\gamma}{\hookrightarrow\!\!\!\!\rightarrow} \Gamma' \rhd C'$ for some type environments Γ and Γ'.

Proposition 5.1. *Let C be a contract service composition. It holds that C is* compliant, *$C\downarrow$, if and only if every C' such that $C \xhookrightarrow{\gamma'}\!\!\!\rightarrow C'$ for some $\gamma' \in Act^*$ there exist C'' and $\gamma'' \in Act^*$ such that $C' \xhookrightarrow{\gamma''}\!\!\!\rightarrow C''$ and $C''\checkmark$.*

Proof. The proof follows by induction on the derivations and Lemma 3.1. $\square$

The modal μ-calculus formula that characterizes compliance is defined as follows:

$$\phi^c \stackrel{\text{def}}{=} \mu X. \left(\bigwedge_{\alpha \in Act} ([\alpha]X) \wedge \phi \right)$$

where

$$\phi \stackrel{\text{def}}{=} \mu X. \left((\checkmark) \vee \bigvee_{\alpha \in Act} (\langle\alpha\rangle X) \right) \wedge \neg \mu X. \left(\bigvee_{\alpha \in Act} (\langle\alpha\rangle X) \right)$$

The sub-formula $\neg\mu X. \left(\bigvee_{\alpha\in Act}(\langle\alpha\rangle X)\right)$ will ensure that any configuration satisfying ϕ^c doesn't get trapped into infinite loops without chances to reach a successful state. The next theorem characterizes the set of service configurations satisfying ϕ^c. A complete proof is given in [22].

Theorem 5.1. *Consider the modal μ-calculus formula ϕ^c defined above. It holds that $Conf(\phi^c) = \{\Gamma \rhd C \mid C\downarrow$ and Γ is a type environment$\}$.*

Proof. The fact that $Conf(\phi^c) \supseteq \{\Gamma \rhd C \mid C\downarrow$ and Γ is a type environment$\}$ easily follows by contradiction. The other direction follows from the fact that, by definition of ϕ^c, for each configuration $\Gamma \rhd C \in Conf(\phi^c)$ it holds that $\Gamma \rhd C \in Conf(\phi')$ where ϕ' is the sub-formula $\mu X. \left((\checkmark) \vee \bigvee_{\alpha\in Act}(\langle\alpha\rangle X)\right)$. $\square$

Corollary 5.1. *A composition C is compliant if and only if $\Gamma \rhd C \in Conf(\phi^c)$ for some type environment Γ.* $\square$

As a consequence of Theorems 4.2 and 5.1 we have:

Corollary 5.2. *Let $\varsigma \in \Sigma$, $\Gamma \rhd C$ be a configuration and*

$$\Phi^{\varsigma}_{\Gamma\rhd C} \stackrel{\text{def}}{=} \phi^{\approx_\varsigma}(\Gamma \rhd C) \ \wedge\ \phi^c.$$

It holds that $\Gamma \rhd C \in Conf(\Phi^{\varsigma}_{\Gamma\rhd C})$ if and only if both $C \in \mathcal{NI}_{\Gamma,\varsigma}$ and $C\downarrow$. $\square$

Using this method we can for instance exploit the model checker NCSU Concurrency Workbench (see [23]) to check whether both $C \in \mathcal{NI}_{\Gamma,\varsigma}$ and $C\downarrow$.

Transitions for filters

$$\delta.f \overset{\delta}{\longmapsto} f \qquad \frac{f\{X := \mathtt{rec}(\mathbf{X})\, f\} \overset{\delta}{\longmapsto} f'}{\mathtt{rec}(\mathbf{X})\, f \overset{\delta}{\longmapsto} f'} \qquad \frac{f \overset{\delta}{\longmapsto} f_\delta \qquad g \overset{\delta}{\longmapsto} g_\delta}{f \otimes g \overset{\delta}{\longmapsto} f_\delta \otimes g_\delta}$$

$$\frac{f \overset{\delta}{\longmapsto} f_\delta \qquad g \overset{\delta}{\longmapsto} g_\delta}{f \times g \overset{\delta}{\longmapsto} f_\delta \times g_\delta} \qquad \frac{f \overset{\delta}{\longmapsto} f_\delta \qquad g \overset{\delta}{\not\longmapsto}}{f \times g \overset{\delta}{\longmapsto} f_\delta} \qquad \frac{f \overset{\delta}{\not\longmapsto} \qquad g \overset{\delta}{\longmapsto} g_\delta}{f \times g \overset{\delta}{\longmapsto} g_\delta}$$

Transitions for filtered peers

$$\frac{\Gamma \rhd p[\sigma] \xrightarrow{\delta} \Gamma \rhd p[\sigma'] \qquad f \overset{\delta}{\longmapsto} f'}{\Gamma \rhd f(p[\sigma]) \xrightarrow{\delta} \Gamma \rhd f'(p[\sigma'])}$$

$$\frac{\Gamma \rhd p[\sigma] \xrightarrow{\tau} \Gamma' \rhd p[\sigma']}{\Gamma \rhd f(p[\sigma])) \xrightarrow{\tau} \Gamma' \rhd f(p[\sigma'])} \qquad \frac{\Gamma \rhd p[\sigma] \checkmark}{\Gamma \rhd f(p[\sigma]) \checkmark}$$

Table 7: Dynamics of Filtered contract service compositions

6. An Adaptive Algorithm

The model checking technique is based on the idea that the state transition graph of a finite-state system defines a Kripke structure, and efficient algorithms can be given for checking if the state graph defines a model of a given specification expressed in an appropriate temporal logic. In the explicit state approach the Kripke structure is represented extensionally, using conventional data structures such as adjacency matrices and linked lists so that each state and transition is enumerated explicitly. Moreover, in the global calculation approach, given a structure M and formula ϕ, the model checking algorithms calculate $\phi^M = \{s : M, s \models \phi\}$ that is the set of all states in M satisfying ϕ. We show how such algorithms can be exploited to develop an adaptive model checking technique for service compositions which adapts, when it is possible, the composition under investigation in such a way that it satisfies both non-interference and compliance. We use the filters, introduced in [4] and revised in [21], as prescriptions of behaviour.

Filters. A filter is the specification of the legal flow of actions for an individual contract. The syntax is as follows, while the semantics is defined in Table 7.

$$f \in \mathcal{F} \ := \ \mathbf{0} \mid \delta.f \mid f \times f \mid f \otimes f \mid X \mid \mathtt{rec}(\mathbf{X})\, f$$

Definition 6.1 (Filter pre-order). The filter pre-order $f \leq g$ is the largest relation such that if $f \overset{\delta}{\longmapsto} f_\delta$ then $g \overset{\delta}{\longmapsto} g_\delta$ and $f_\delta \leq g_\delta$. □

We note $(\mathcal{F}, \sqsubseteq)$ the partial order induced by $\leq$: by abuse of notation, we identify a filter f with its equivalence class $[f]_\sim$, where $\sim$ is the symmetric closure of $\leq$. The union and intersection of filters represent the glb and lub operators for $(\mathcal{F}, \sqsubseteq)$. Furthermore, if we assume a finite alphabet A of actions, the set of filters $\mathcal{F}_A$ insisting on A forms a complete lattice with **0** as bottom and the identity filter $I_A \overset{def}{=} \texttt{rec}(\mathbf{X}) \prod_{\delta \in A} \delta.X$ as top element.

The application $\Gamma \rhd f(p[\sigma])$ blocks any action from $\Gamma \rhd p[\sigma]$ that is not explicitly enabled by f. Filters may be composed to help shape a service composition. Given a set π of principals, a composite π-filter F is a finite map from the principals in π to filters: $\{p \to f[p] \mid p \in \pi\}$. A π-filter may be applied to a π-composition:

$$\Gamma \rhd F(p_1[\sigma_1] \parallel \cdots \parallel p_n[\sigma_n]) ::= \Gamma \rhd F[p_1](p_1[\sigma_1]) \parallel \cdots \parallel \Gamma \rhd F[p_n](p_n[\sigma_n])$$

When we write $\Gamma \rhd F(C)$ we tacitly assume that the underlying set of principals for both F and C is π. The operators of union and intersection, as well as the ordering on filters extends directly to composite filters, as expected. Namely, for F and G π-filters and for $\bullet \in \{\times, \otimes\}$:

$$\begin{array}{ccll} F \leq_\pi G & \text{iff} & F[p] \leq G[p] & \text{for all } p \in \pi \\ (F \bullet_\pi G)[p] & \overset{def}{=} & F[p] \bullet G[p] & \text{for all } p \in \pi \end{array}$$

We generalize the syntax of service compositions by allowing the term $\Gamma \rhd F(C)$ to account for the application of filters on the components of C. The dynamics of filtered service compositions derives directly by combining the transitions in Tables 2 and 7.

Relevance. Below we present an algorithm that given a configuration $\Gamma \rhd C$ infers a composite filter F that fixes $\Gamma \rhd C$, whenever such F exists. The algorithm is so structured as to guarantee two important properties on the inferred filter. On the one hand, the filter is as permissive as possible, in that it is the greatest (with respect to the pre-order $\leq$) among the filters that fix $\Gamma \rhd C$. On the other side, the inferred filter is *relevant*, i.e., minimal in size: for any computation state reached by the service configuration via a series of τ transitions (local moves or synchronizations), the filter only enables actions that may be attempted at that state (either directly, or via a local choice), by one of the components of the service configuration.

Definition 6.2 (Relevance). Let π be a set of principals and $\mathcal{C}$ be a non-empty set of π-configurations. A filter f is *p-relevant* in $\mathcal{C}$, written $f \propto_p \mathcal{C}$, if whenever

$f \overset{\delta}{\longmapsto} \widehat{f}$ one has $\delta \in \{a_{\_\to p}, \bar{a}_{p\to\_}\}$ and there exists $\Gamma \rhd C \in \mathcal{C}$ such that $\Gamma \rhd C \overset{\alpha}{\hookrightarrow\!\!\!\twoheadrightarrow}$ with $\alpha \in \{\{a\}_{\_\to p}, \{a\}_{p\to\_}\}$ and $\widehat{f} \propto_p \{\Gamma' \rhd C' \mid \Gamma \rhd C \overset{\alpha}{\hookrightarrow\!\!\!\twoheadrightarrow} \Gamma' \rhd C'\}$. A composite π-filter F is *relevant* for $\mathcal{C}$, written $F \propto \mathcal{C}$, if $F(p) \propto_p \mathcal{C}$ for all $p \in \pi$. A composite π-filter is relevant for a π-configuration $\Gamma \rhd C$ if $F \propto \{\Gamma \rhd C\}$. $\Box$

The Algorithm. We describe an algorithm that synthesizes the $\sqsubseteq$-greatest relevant filter that fixes $\Gamma \rhd C$, if it exists, when $\Gamma \rhd C$ does not satisfy $\Phi^{\varsigma}_{\Gamma\rhd C}$.

As discussed above, a global model checking algorithm applied to a configuration $\Gamma \rhd C$ and the modal formula $\Phi^{\varsigma}_{\Gamma\rhd C}$ calculates the set of states in the reduction graph (tracing the states reached by means of synchronizations or internal moves) of $\Gamma \rhd C$ satisfying $\Phi^{\varsigma}_{\Gamma\rhd C}$. This is the input of our algorithm. The reduction graph can be represented as a directed graph $G = (V, E)$ with labelled edges and vertices. The vertices in V represent the reachable states of $\Gamma \rhd C$. With each $\mathbf{v} \in V$ we associate two fields: $state[\mathbf{v}]$ gives the computation state (i.e., the derivative $\Gamma' \rhd C'$ of the initial state $\Gamma \rhd C$) associated with $\mathbf{v}$; $result[\mathbf{v}]$ is a tag SUCC or FAIL depending on whether the corresponding configuration satisfies $\Phi^{\varsigma}_{\Gamma\rhd C}$ or not as calculated by the model checker. An edge in E is a triple $(\mathbf{u}, \mathbf{v})_\alpha$ representing the transition $state[\mathbf{u}] \overset{\alpha}{\hookrightarrow} state[\mathbf{v}]$. Reduction graphs may be stored in a adjacency list representation, so that the set of outgoing edges for each $\mathbf{u} \in V$ can be retrieved as $Adj[\mathbf{u}]$: thus $(\mathbf{u}, \mathbf{v})_\alpha \in E$ iff $(\alpha, \mathbf{v}) \in Adj[\mathbf{u}]$. We also write $Adj[\mathbf{u}, \alpha]$ for the set $\{\mathbf{v} \in V \mid (\mathbf{u}, \mathbf{v})_\alpha \in E\}$. Vertices with no outgoing edges are called leaves. We denote by $root[G]$ the vertex representing the initial state $\Gamma \rhd C$.

The first step consists in *re-labelling* the graph G calculated by the model-checker in such a way that the result label at each vertex $\mathbf{u}$ is set to FAIL if there exists at least one silent transition from $\mathbf{u}$ to a FAIL vertex; it is set to SUCC if either there are no silent transitions from $\mathbf{u}$ to a FAIL vertex and there exists a silent transition from $\mathbf{u}$ to a SUCC vertex or there exists one non-silent and non-conflicting transition from $\mathbf{u}$ to a SUCC vertex. The procedure iteratively examines all the vertices in the graph until it reaches a fixed point. This computation is accomplished by the `PushLabels` procedure and uses the following auxiliary definitions. Let $locs(\alpha)$ be $\{p, q\}$ in case $\alpha = \{a_{p\to q}\}$, and $\emptyset$ in case $\alpha = \tau$. Then, let $G = (V, E)$ be a reduction graph, and $\alpha = \{a_{p\to q}\}$.

- A path $\varpi = (\mathbf{u}, \mathbf{u}_1)_{\alpha_1}, \ldots, (\mathbf{u}_{n-1}, \mathbf{v})_{\alpha_n}$ from $\mathbf{u}$ to $\mathbf{v}$ in G is *α-free* if $locs(\alpha) \cap locs(\alpha_i) = \emptyset$ for all i's.
- A vertex $\mathbf{v}$ is a *α-free descendant* of $\mathbf{u}$ in G (dually, $\mathbf{u}$ is a *α-free ancestor* of $\mathbf{v}$) if there is a α-free path from $\mathbf{u}$ to $\mathbf{v}$.
- A vertex $\mathbf{u}$ *yields a conflict on* α if $\mathbf{u}$ has two distinct α-free descendants $\mathbf{v}_1$ and $\mathbf{v}_2$ such that $(\mathbf{v}_1, \mathbf{w}_1)_\alpha$ and $(\mathbf{v}_2, \mathbf{w}_2)_\alpha \in E$ and $result[\mathbf{w}_1] \neq result[\mathbf{w}_2]$.

Procedure `PushLabels(`G`)`

```
Input: A reduction graph $G = (V, E)$
Output: The graph $G$ updated
$done$ := false;
while $\neg\, done$ do
    $done$ := true;
    foreach $\mathbf{u} \in V$ do
        $succ$ := false;  $fail$ := false;
        if $Adj[\mathbf{u}, \tau] \neq \emptyset$ then
            if $\exists \mathbf{v} \in Adj[\mathbf{u}, \tau] : result[\mathbf{v}] = \text{FAIL}$ then
                $fail$ := true;
            else if $\exists \mathbf{v} \in Adj[\mathbf{u}, \tau] : result[\mathbf{v}] = \text{SUCC}$ then
                $succ$ := true;
        else if $\exists (\alpha, \mathbf{v}) \in Adj[\mathbf{u}] \wedge result[\mathbf{v}] = \text{SUCC} \wedge \neg Conflict(\alpha, \mathbf{u})$
        then
            $succ$ := true;
        if $succ \wedge result[\mathbf{u}] \neq \text{SUCC}$ then
            $result[\mathbf{u}]$ := SUCC;  $done$ := false;
        else if $fail \wedge result[\mathbf{u}] \neq \text{FAIL}$ then
            $result[\mathbf{u}]$ := FAIL;  $done$ := false
```

- A vertex $\mathbf{v}$ has a conflict on α in G, noted $Conflict_G(\alpha, \mathbf{v})$ if $\mathbf{v}$ has a α-free ancestor yielding a conflict on α.

Intuitively, our algorithm will prune G by banning all the 'bad' synchronizations, and by preserving all the 'good' synchronizations that lead to nodes satisfying both non-interference and compliance. Due to the presence of internal choices, the same synchronization can look good at one point, but actually be bad. The definition of conflict formally captures this notion of ambiguous synchronizations.

Proposition 6.1. *After the call to* `PushLabels(`G`)`*, the following conditions hold for every node* $\mathbf{u}$ *in* G*:*

1. $result[\mathbf{u}] = \text{FAIL}$ *iff either there exists no* $(\mathbf{u}, \mathbf{v})_\alpha \in E$ *such that* $result[\mathbf{v}] = \text{SUCC}$ *and* $\neg Conflict_G(\alpha, \mathbf{u})$ *or there exists* $(\mathbf{u}, \mathbf{v})_\tau \in E$ *such that* $result[\mathbf{v}] = \text{FAIL}$*;*
2. $result[\mathbf{u}] = \text{SUCC}$ *iff there exists no* $(\mathbf{u}, \mathbf{v})_\tau \in E$ *such that* $result[\mathbf{v}] = \text{FAIL}$ *and there exists either* $(\mathbf{u}, \mathbf{v})_\tau \in E$ *such that* $result[\mathbf{v}] = \text{SUCC}$ *or* $(\mathbf{u}, \mathbf{v})_\alpha \in E$ *with* $\alpha \neq \tau$*,* $\neg Conflict_G(\alpha, \mathbf{u})$ *and* $result[\mathbf{v}] = \text{SUCC}$.

Function `SuccessGraph(`G`)`

```
Input: A reduction graph $G = (V, E)$
Output: $G' = (V', E')$ the success sub-graph of $G$
$V' := (result[root[G]] = \textsc{succ})\,?\,\{root[G]\} : \emptyset;\quad E' := \emptyset;$
$done :=$ false;
while $\neg\, done$ do
    $done :=$ true;
    foreach $(\mathbf{u}, \mathbf{v})_\alpha \in E \setminus E'$ do
        if $\mathbf{u} \in V' \wedge result[\mathbf{v}] = \textsc{succ} \wedge \neg Conflict(\alpha, \mathbf{u})$ then
            $V' := V' \cup \{\mathbf{v}\};\ E' := E' \cup \{(\mathbf{u}, \mathbf{v})_\alpha\};$
            $done :=$ false
return $G' = (V', E')$;
```

Proof. The proof easily follows by construction. □

We say that a path ϖ in G is *successful* if $result[\mathbf{u}] = \textsc{succ}$ for every node $\mathbf{u}$ in ϖ, otherwise ϖ is *unsuccessful*. A node $\mathbf{u}$ is *root-successful* if it is reachable from *root*$[G]$ via a successful path, otherwise it is *root-unsuccessful*. The next step of the algorithm computes the sub-graph of G that only includes the root-successful vertices. This computation is accomplished by the `SuccessGraph` function.

Proposition 6.2. *Let $G' = (E', V')$ be generated by* `SuccessGraph(`G`)`. *Then $\mathbf{u} \in V'$ if and only if $\mathbf{u}$ is root-successful in G.* □

The final step of the algorithm synthesizes the filter out of the success graph, in case this is not empty. Let $G' =$ `SuccessGraph(`G`)`, π be the underlying set of principals, and $F^{Alg}[\Phi^{\varsigma}_{\Gamma \rhd C}] = \texttt{ExtractFilter}_\pi(root[G], \emptyset, G')$. A complete proof of the next theorem is given in [24].

Theorem 6.1 (Soundness and maximality)**.** *Let $\Gamma \rhd C$ be a π-composition. Then $\Gamma \rhd F^{Alg}[\Phi^{\varsigma}_{\Gamma \rhd C}](C)$ is such that*

- $F^{Alg}[\Phi^{\varsigma}_{\Gamma \rhd C}](C) \in \mathcal{NI}_{\Gamma,\varsigma}$
- $F^{Alg}[\Phi^{\varsigma}_{\Gamma \rhd C}](C)\downarrow$.

Also, if a filter F fixes $\Gamma \rhd C$ and is relevant for $\Gamma \rhd C$, then $F \leq F^{Alg}[\Phi^{\varsigma}_{\Gamma \rhd C}]$.

Proof. The proof follows by construction and by Propositions 6.1 and 6.2. □

Function `ExtractFilter`$_\pi$($\mathbf{u}, U, G$)

```
Input: $G = (V, E)$ a success graph. $\mathbf{u} \in V, U \subseteq V$
Output: $F$, an $\pi$-composite filter

$F[p] := \mathbf{0}$ for all $p \in \pi$;
if $state[\mathbf{u}]$ ✓ then
    return $F$;
if $\mathbf{u} \in U$ then
    $rec[\mathbf{u}] :=$ true; return $(X_\mathbf{u}, \ldots, X_\mathbf{u})$;
foreach $(\alpha, \mathbf{v}) \in Adj[\mathbf{u}]$ do
    $F_\mathbf{v} := \texttt{ExtractFilter}_\pi(\mathbf{v}, U \cup \{\mathbf{u}\}, G)$;
    foreach $p \in \pi$ do
        if $\alpha = \{a_{p \to \_}\}$ then
            $F[p] := F[p] \times \bar{a}_{p \to \_}.F_\mathbf{v}[p]$;
        else if $\alpha = \{a_{\_ \to p}\}$ then
            $F[p] := F[p] \times a_{\_ \to p}.F_\mathbf{v}[p]$;
        else
            $F[p] := F[p] \times F_\mathbf{v}[p]$;
if $rec[\mathbf{u}] =$ true then
    foreach $p \in \pi :\ X_\mathbf{u} \in fv(F[p])$ do
        $F[p] := \texttt{rec}(\mathbf{X_u})\, F[p]$;
return $F$;
```

7. A case study

In this section we address the problem of formalizing the interactions among an authentication service (AS), a web service (WS) and a user. First, we introduce the system description and discuss how our formalism copes with it. Then, we show that the non-interference framework presented in Section 3 can be used to prevent the developer to introduce architecture flaws while projecting the WS. The advantage of such a formal approach is shown to be twofold: first, it gives an unambiguous method to describe a possibly complex software architecture consisting of several cooperating services. Second, it allows one to decide if a model satisfies certain security properties in a completely algorithmic way.

7.1. System definition

Authentication services are web services that allow web applications to partially or totally avoid managing users' accounts. The advantages of using authentication services are: the possibility for the user to own just one account to access several independent web services and the simplifications of the design of the web service since the storage and managing of user accounts is left to a specialised application. Most of the implementations of such a mechanism relies on a project called *OpenID*. A user that wishes to obtain an OpenID, just needs to register with one of the providers that support the project: Google, Yahoo! and Wordpress are probably the most common. Many web applications and services that require authentication, at the login phase, redirect the user to one of the providers handling OpenID. Then, the authentication server recognises the user and sends back both to the user and to the web application that the authentication has been carried out successfully. This process is illustrated in Figure 1 and consists of nine steps:

1. The WA offers the end user a set of login options
2. The user selects to log-in with a third party account, choosing among OpenID-compatible providers (e.g., Google, facebook and Yahoo!)
3. The WA sends a *discovery* request to the selected provider in order to obtain a login web service endpoint
4. The provider answers with a eXtensible Resource Descriptor Sequence (XRDS) document that contains the address(es) of the login web service endpoint
5. The WA contacts the login endpoint to send a request for authentication service
6. The user is redirected to the chosen provider where the authentication step can be done
7. The user provides login information and, upon success, receives a confirmation and is again redirected to the web application
8. The AS sends to the WA a token which confirms user's identity
9. Finally, the WA is able to exchange confidential information to the user

Note that, with respect to the previous examples, in this case the confidentiality is a requirement of the web application instead of the user.

7.2. An ideal model

We consider an ideal model of the federated authentication system based on OpenID, i.e., in the first step we do not consider the possibility of errors in the WA development. The security level lattice is $\Sigma = \{L, h_1, h_2, H\}$ and Figure 2 shows relation $\preceq$. Let us assume that a WA is able to provide certain types of information that can be classified according to three security levels: L, h_1, h_2. Unknown users' principals are associated with the lowest security level L, while authenticated users are associated with one of the higher security levels h_1 and h_2. The web application security level is the highest: H. Observe that, the security levels are assigned according to the WA confidentiality requirements. We first give the definition of the

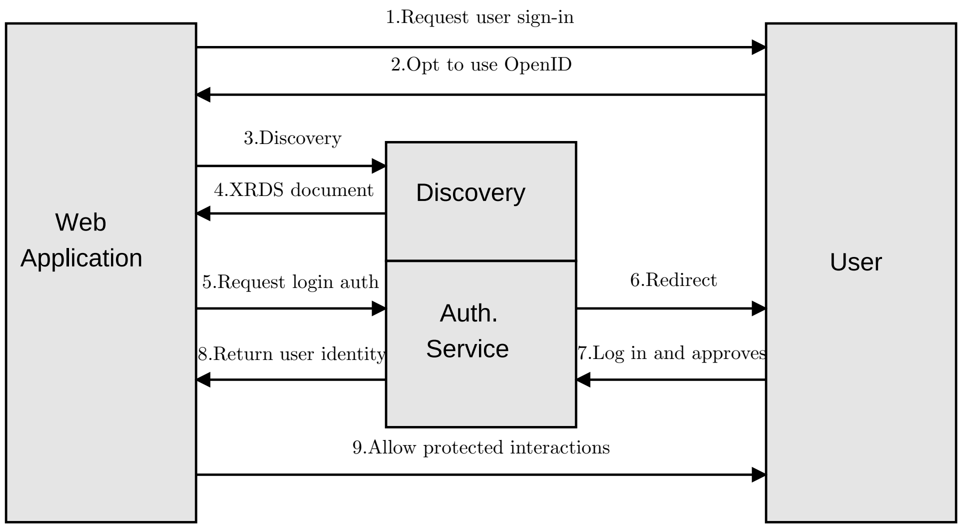

Figure 1: OpenID login authentication for web applications.

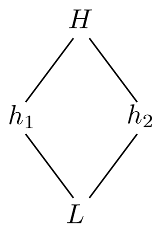

Figure 2: Lattice $\langle \Sigma, \preceq \rangle$ used in the Example of Section 7.

$$\begin{aligned}
\sigma_D = \quad & \texttt{DiscReq}@x.\overline{\texttt{Disc}_{\texttt{AS}}}@x.\mathbf{1} \\
\sigma_{AS} = \quad & \texttt{Req}_{\texttt{C}}@y.\overline{\texttt{Redirect}}@C.\texttt{GetAccount}@C. \\
& (\overline{\texttt{Ok}}@C.\overline{\texttt{Token}}@y.\mathbf{1}\lfloor_{\emptyset}\oplus{}_{\emptyset}\rfloor\overline{\texttt{Fail}}@C.\overline{\texttt{NoToken}}@y.\mathbf{1})
\end{aligned}$$

Table 8: Contracts executed by the Authentication Service and the Discovery Service in the model for federated login system.

contracts executed by the discovery (σ_D) and authentication (σ_{AS}) web services. Their definitions in terms of the calculus are given in Table 8. Process σ_D is rather simple, since it simply returns a WA endpoint ($\texttt{Disc}_{\texttt{AS}}$) upon a request `DiscReq`. Observe that, endpoint `AS` is going to be used as a principal name by the WA's contract. Since in our calculus we do not explicitly model the variable passing between synchronisations, the action synchronising with $\overline{\texttt{Disc}_{\texttt{AS}}}@x$ is part of an external choice with all the possible values of the output (see [16]). To clarify, consider the definition of σ_{AS} in Table 8: note that the first input $\texttt{Req}_{\texttt{C}}$ contains the label of principal C that is contacted by the following output action. Therefore, a complete definition of the AS process would be:

$$\sigma'_{AS} = \sigma_{ASC_1} + \sigma_{ASC_2} + \ldots + \sigma_{ASC_n},$$

where σ_{ASC_i} is the definition of σ_{AS} of Table 8 rewritten for $C = C_i$, and $C_1, \ldots, C_n$ are all the possible (finite) instances of user principal.

However, as illustrated by the definition of σ_{AS} in Table 8, for the sake of a compact notation, we omit to specify all the possible external choices. The AS process, after receiving a request to authenticate user C ($\texttt{Req}_{\texttt{C}}$) redirects C to the login service and waits for the account information (`Account`). If the authentication is successful, then C is informed by receiving an `Ok` message and the authentication token is sent to the WA. Otherwise, both are informed that the authentication process failed.

Table 9 formalises the definition of contract σ_{WA} performed by principal WA. Its behaviour strictly resembles the process depicted by Figure 1. The first line of the definition σ_{WA} corresponds to steps 1 and 2 of the authentication procedure. User's principal C can decide between an internal authentication procedure or one based on OpenID. In the first case, the WS directly accepts the account information (`Account`) and based on them, it decides if the user is not recognised (sending of message $\overline{\texttt{NoAuth}}$, if it can access information with security level h_1 (and hence the process continues as σ_{Prof1}) or with security level h_2 (and hence the process continues as σ_{Prof2}). The offered services depends on the user authentication and

$$
\begin{aligned}
\sigma_{WA} &= \mathtt{ConfReq}@x.\overline{\mathtt{Choices}}@x.(\mathtt{Internal}@x.\sigma_{AUT} + \mathtt{OpenID}@x.\sigma_{OID})\\
\sigma_{AUT} &= \mathtt{Account}@x.(\overline{\mathtt{Passed}}@x.(\sigma_{Prof_1} \lfloor_{x:h_1} \oplus \,_{x:h_2}\rfloor \sigma_{Prof_2})\\
&\quad \lfloor_{\emptyset} \oplus \,_{\emptyset}\rfloor \overline{\mathtt{NoAuth}}@x.\sigma_{Ser_3})\\
\sigma_{OID} &= \overline{\mathtt{DiscReq}}@D.\mathtt{Disc}_{\mathtt{AS}}@D.\overline{Req_x}@AS.(\mathtt{Token}@AS.\\
&\quad (\sigma_{Prof_1} \lfloor_{x:h_1} \oplus \,_{x:h_2}\rfloor \sigma_{Prof_2}) + \mathtt{NoToken}@AS.\sigma_{Ser_3})\\
\sigma_{Prof_1} &= \sigma_{Ser_1} + \sigma_{NoSer_2} + \sigma_{Ser_3}\\
\sigma_{Prof_2} &= \sigma_{NoSer_1} + \sigma_{Ser_2} + \sigma_{Ser_3}\\
\sigma_{Ser_i} &= \mathtt{Op}_i@x.\overline{\mathtt{Reply}_i}@x.\mathbf{1} \qquad i = 1, 2, 3\\
\sigma_{NoSer_i} &= \mathtt{Op}_i@x.\overline{\mathtt{Deny}_i}@x.\mathbf{1} \qquad i = 1, 2
\end{aligned}
$$

Table 9: Contract executed by the Web Application.

$$
\begin{aligned}
\sigma_C &= \overline{\mathtt{ConfReq}}@WA.\mathtt{Choices}@WA.\\
&\quad (\overline{\mathtt{Internal}}@WA.\sigma_{IA} \lfloor_{\emptyset} \oplus \,_{\emptyset}\rfloor \overline{\mathtt{OpenID}}@x.\sigma_{OID})\\
\sigma_{IA} &= \overline{\mathtt{Account}}@WA.(\mathtt{Passed}@WA.\sigma_{Act}\\
&\quad + \mathtt{NoAuth}@x.\overline{\mathtt{Op}_3}@x.\mathtt{Reply}_3@x.\mathbf{1})\\
\sigma_{OID} &= \mathtt{Redirect}@z.\overline{\mathtt{GetAccount}}@z.(\mathtt{Ok}@z.\sigma_{Act}\\
&\quad + \mathtt{Fail}@z.\overline{\mathtt{Op}_3}@x.\mathtt{Reply}_3@x.\mathbf{1})\\
\sigma_{Act} &= \overline{\mathtt{Op}_3}@WA.\mathtt{Reply}@WA_3.\mathbf{1} \lfloor_{\emptyset} \oplus \,_{\emptyset}\rfloor (\overline{\mathtt{Op}_2}@WA.(\mathtt{Reply}_2@WA.\mathbf{1} +\\
&\quad \mathtt{Deny}_2@WA.\mathbf{1}) \lfloor_{\emptyset} \oplus \,_{\emptyset}\rfloor \overline{\mathtt{Op}_1}@WA.(\mathtt{Reply}_1@WA.\mathbf{1} + \mathtt{Deny}_1@WA.\mathbf{1}))
\end{aligned}
$$

Table 10: Contract executed by the User.

are σ_{Ser_1} for users with level h_1, σ_{Ser_2} for users with level h_2, whereas σ_{Ser_3} is available for both (and also non-registered users). Observe that upon a successful authentication, user's security level is upgraded by the internal choice operator. Finally, Table 10 gives the definition of the user process σ_C.

Initially, user's contract σ_C synchronises with WA and chooses one of the two available authentication methods. In both cases, it sends the account information ($\overline{\texttt{Account}}$ and $\overline{\texttt{Getaccount}}$) and then proceeds with a service request (σ_{Act}). Note that, if the authentication method is OpenID then a `Redirect` input allows the principal to identify the authentication service endpoint. Finally, the user performs an internal choice among the three possible requests $\texttt{Op}_i$, $i = 1, 2, 3$. Observe that, in order to satisfy the compliance property, we consider the possibilities that in case operations 1 or 2 are requested, then WA may answer with a denial of access according to the user profile. It can be proved that the service composition $M = WA[\sigma_{WA}] \parallel C[\sigma_C] \parallel D[\sigma_D] \parallel AS[\sigma_{AS}]$ with the type system $\Gamma = \{WA : H, C : L, D : L, AS : L\}$ is compliant. However, if we consider the definitions of Table 9 omitting σ_{NoSer_i}, (intuitively, the WA simply does not accept requests from unauthorised users), then configuration M would be non-compliant. In fact, a user that cannot ask for service i ($i = 1, 2$) could anyway perform an internal choice for it (see the definition of σ_{Act}) and hence cause a deadlock in its execution. It is worthwhile pointing out that this corresponds to what is known to be a good software engineering practice, i.e., handling with correct procedures (maybe of denial) all the possible requests from the users. Finally, $M \in \mathcal{NI}_{\Gamma,L}$ since once the security level of the customer is upgraded to h_1 or h_2, all the synchronisations between WA and C have the same level of security.

7.3. Compliance and non-interference analysis in an insecure WA

Web services are vulnerable to many attacks that can overcome the security measures designed by the developer. This is mainly due to the fact that most of the web service endpoints are completely exposed in a untrusted environment, i.e., the Internet. In this part we consider the possibility that a user, for malicious purposes, is able to send to the WA an ill-formed request that causes a failure in the procedure that is being carried out. In practice, this corresponds to a badly formed XML document or to a SQL injection attack. It is well-known that correct security implementations require that the WA reveals, as a consequence of a failure, the lowest amount of information about its activity so that the attacker cannot infer any information about the confidential activities that are being processed (see, e.g., [25]). We show how our non-interference framework is able to formally detect if a WA is designed according to this security principle.

The revised version of the model is depicted in Table 11 where the definition of the AS and D contracts are omitted since they are identical to those of Table 8.

In this model, we assume that the WA may receive the malicious request at two epochs: just before the user chooses the required service (before synchronisations on $\texttt{Op}_i$) and just before it receives the answer for that request. In both cases, the WA procedure aborts. Although, a WA that suffers this problem clearly contains flaws in its design (probably missing checks on input), these are very common

$$M = AS[\sigma_{AS}] \parallel C[\sigma_C] \parallel X[\sigma_X] \parallel D[\sigma_D] \parallel WA[\sigma_{WA}]$$

$$\sigma_{WA} = \texttt{ConfReq@}x.\overline{\texttt{Choices}}\texttt{@}x.(\texttt{Internal@}x.\sigma_{AUT} + \texttt{OpenID@}x.\sigma_{OID})$$

$$\begin{aligned}\sigma_{AUT} = {} & \texttt{Account@}x.(\overline{\texttt{Passed}}\texttt{@}x.(\sigma_{Prof_1} \lfloor_{x:h_1} \oplus {}_{x:h_2} \rfloor \sigma_{Prof_2}) \\ & \lfloor_{\emptyset} \oplus {}_{\emptyset} \rfloor \overline{\texttt{NoAuth}}\texttt{@}x.\sigma_{Prof_3})\end{aligned}$$

$$\begin{aligned}\sigma_{OID} = {} & \overline{\texttt{DiscReq}}\texttt{@}D.\texttt{Disc}_{\texttt{AS}}\texttt{@}D.\overline{Req}_x\texttt{@}AS.(\texttt{Token@}AS. \\ & (\sigma_{Prof_1} \lfloor_{x:h_1} \oplus {}_{x:h_2} \rfloor \sigma_{Prof_2}) + \texttt{NoToken@}AS.\sigma_{Prof_3})\end{aligned}$$

$$\sigma_{Prof_1} = \sigma_{Ser_1} + \sigma_{NoSer_2} + \sigma_{Ser_3} + \texttt{IllMsg@}y.\overline{\texttt{Msg}_0}\texttt{@}y.\overline{\texttt{Close}}\texttt{@}x.\mathbf{1}$$

$$\sigma_{Prof_2} = \sigma_{NoSer_1} + \sigma_{Ser_2} + \sigma_{Ser_3} + \texttt{IllMsg@}y.\overline{\texttt{Msg}_0}\texttt{@}y.\overline{\texttt{Close}}\texttt{@}x.\mathbf{1}$$

$$\sigma_{Prof_3} = \sigma_{Ser_3} + \texttt{IllMsg@}y.\overline{\texttt{Msg}_0}\texttt{@}y.\overline{\texttt{Close}}\texttt{@}x.\mathbf{1}$$

$$\sigma_{Ser_i} = \texttt{Op}_i\texttt{@}x.(\overline{\texttt{Reply}_i}\texttt{@}x.\mathbf{1} + \texttt{IllMsg@}y.\overline{\texttt{Msg}_i}\texttt{@}y.\overline{\texttt{Close}}\texttt{@}x.\mathbf{1}) \qquad i = 1, 2, 3$$

$$\sigma_{NoSer_i} = \texttt{Op}_i\texttt{@}x.\overline{\texttt{Deny}_i}\texttt{@}x.\mathbf{1} \qquad i = 1, 2$$

$$\begin{aligned}\sigma_C = {} & \overline{\texttt{ConfReq}}\texttt{@}WA.\texttt{Choices@}WA. \\ & (\overline{\texttt{Internal}}\texttt{@}WA.\sigma_{IA} \lfloor_{\emptyset} \oplus {}_{\emptyset} \rfloor \overline{\texttt{OpenID}}\texttt{@}x.\sigma_{OID})\end{aligned}$$

$$\begin{aligned}\sigma_{IA} = {} & \overline{\texttt{Account}}\texttt{@}WA.(\texttt{Passed@}WA.\sigma_{Act} \\ & + \texttt{NoAuth@}x.\overline{\texttt{Op}_3}\texttt{@}x.\texttt{Reply}_3\texttt{@}x.\mathbf{1})\end{aligned}$$

$$\begin{aligned}\sigma_{OID} = {} & \texttt{Redirect@}z.\overline{\texttt{GetAccount}}\texttt{@}z.(\texttt{Ok@}z.\sigma_{Act} \\ & +\texttt{Fail@}z.\overline{\texttt{Op}_3}\texttt{@}x.\texttt{Reply}_3\texttt{@}x.\mathbf{1})\end{aligned}$$

$$\begin{aligned}\sigma_{Act} = {} & (\overline{\texttt{Op}_3}\texttt{@}WA.(\texttt{Reply}_3\texttt{@}WA.\mathbf{1} + \texttt{Close@}WA.\mathbf{1}) \lfloor_{\emptyset} \oplus {}_{\emptyset} \rfloor \\ & (\overline{\texttt{Op}_2}\texttt{@}WA.(\texttt{Reply}_2\texttt{@}WA.\mathbf{1} + \texttt{Deny}_2\texttt{@}WA.\mathbf{1} + \texttt{Close@}WA.\mathbf{1}) \\ & \lfloor_{\emptyset} \oplus {}_{\emptyset} \rfloor \overline{\texttt{Op}_1}\texttt{@}WA.(\texttt{Reply}_1.\texttt{@}WA.\mathbf{1} + \texttt{Deny}_1\texttt{@}WA.\mathbf{1} + \\ & \texttt{Close@}WA.\mathbf{1}))) + \texttt{Close@}WA.\mathbf{1}\end{aligned}$$

$$\begin{aligned}\sigma_X = {} & \overline{\texttt{IllMsg}}\texttt{@}WA.(\texttt{Msg}_0\texttt{@}WA.\mathbf{1} + \texttt{Msg}_1\texttt{@}WA.\mathbf{1} \\ & +\texttt{Msg}_2\texttt{@}WA.\mathbf{1} + \texttt{Msg}_3\texttt{@}WA.\mathbf{1}) + \mathbf{1}\end{aligned}$$

Table 11: Insecure contracts for WA and C.

and it is dangerous to assume that a service is immune from them. Therefore, the developer should take care that, when this type of attack happens, the system does not violate its confidentiality requirements and remains complaint. Contract σ_X executed by attacker's principal X tries to send to the WA a malicious request (e.g., SQL injection) by synchronising on `IllMsg` and if this happens, then it catches one of the possible error messages returned. Observe that, before an operation request $\texttt{Op}_i$, $i = 1, 2, 3$, the WA returns an error message $\texttt{Msg}_0$, whereas after that it returns an error message $\texttt{Msg}_i$ that depends on the service being processed.

We now show that the contract depicted by Table 11 are compliant but does not yield the non-interference property for security level L. The compliance can be straightforwardly derived from the observation that upon an `IllMsg` input the contract stops its activity just after sending an error message to the principal that caused it, and closing the connection with the customer. Symmetrically, user's contract always considers the possibility of an abortion of the service by synchronising on `Close`. To prove that $M \notin \mathcal{NI}_{\Gamma,L}$ it suffices to show that in M when X synchronises on $\texttt{Msg}_1$ this is surely preceded by a synchronisation on $\texttt{Op}_1$ of level h_1, therefore the attacker can infer the profile of the user being logged and that it required an operation $\texttt{Op}_1$ (the same holds for $\texttt{Op}_2$). This vulnerability is well-known by practitioners, and is usually caused by un-handled exceptions. Conversely, exceptions should be always handled and the principal that causes them should received a short error message that must not depend on the server status [25]. In fact, in the model of Table 11, if we assume $\texttt{Msg}_i = \texttt{Msg}$ it is possible to automatically prove that $M \in \mathcal{NI}_{\Gamma,L}$. Although in this case the non-interference analysis leads to an intuitive result, this is not always true. As an instance, consider the model depicted in Table 11 with $\texttt{Msg}_i = \texttt{Msg}$ for $i = 0, 1, 2, 3$ but without the possibility to send a malicious message just before $\texttt{Op}_i$ synchronisations. At a first glance, this modification corresponds to an improvement of the system security, however $M \notin \mathcal{NI}_{\Gamma,L}$ holds. The point is that an observer can infer that a high security level synchronisation occurred by observing how the error is handled. In fact, if `Close` synchronisation is done at level L (and hence is observable), then for sure neither $\texttt{Op}_1$ nor $\texttt{Op}_2$ synchronisations occurred, otherwise a synchronisation of security level $\{h_1, h_2\}$ has been done. Therefore, the system reveals more information about its confidential activities with respect to the system that can be attacked also just before user's choice of $\texttt{Op}_i$.

8. Conclusion

Some research efforts on model checking web services have already been proposed [26, 27, 28, 29]. The most related paper that we are aware of is by Nakajima [30] who introduces a lattice-based security labelling into BPEL in order to detect potential insecure information leakage. The paper discusses how both the safety and security aspects can be analyzed in a single framework using the model-checking verification techniques. The main difference with our approach is that the notion of security considered in [30] is built upon a simple lattice-based model for security labels. Instead, our approach deals with more flexible security policies which can be dynamically specified by the service participants. As far as correctness is concerned, [30] considers safety properties such as deadlock freedom and specific progress properties. Our model instead deals also with the property of livelock freedom.

In conclusion, we have developed a formal method for the analysis of both information flow security and compliance of contract service compositions. This is based on the characterization of such properties in terms of modal μ-calculus formulae. This allows us to use a model checker, like the NCSU Concurrency Workbench, in order to simultaneously check non-interference and compliance. An algorithm for adaptable service compositions is also proposed. It computes the greatest relevant filter fixing them.

References

[1] A. H. Anderson, An introduction to the web services policy language (wspl), in: 5th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2004), IEEE Computer Society, 2004, pp. 189–192.

[2] K. Bhargavan, C. Fournet, A. D. Gordon, Verifying policy-based web services security, ACM Trans. Program. Lang. Syst. 30 (6) (2008) 1–59.

[3] S. Carpineti, G. Castagna, C. Laneve, L. Padovani, A Formal Account of Contracts for Web Services, in: Proc. of the International Workshop on Web Services and Formal Methods (WS-FM'06), Vol. 4184 of LNCS, Springer, 2006, p. 148162.

[4] G. Castagna, N. Gesbert, L. Padovani, A Theory of Contracts for Web Services, in: Proc. of the annual Symposium on Principles of Programming Languages (POPL'08), ACM press, 2008, pp. 261–272.

[5] G. Castagna, N. Gesbert, L. Padovani, A Theory of Contracts for Web Services, ACM Transactions on Programming Languages and Systems (TOPLAS) 31 (2009) 53–61.

[6] C. Laneve, L. Padovani, The *must* Preorder Revisited, in: Proc. of the International Conference on Concurrency Theory (CONCUR'07), Vol. 4703 of LNCS, Springer, 2007, pp. 212–225.

[7] M. Bravetti, G. Zavattaro, A Foundational Theory of Contracts for Multi-party Service Composition, Fundamenta Informaticae 89 (4) (2009) 451–478.

[8] M. Bravetti, G. Zavattaro, Towards a unifying theory for choreography conformance and contract compliance, in: Proc. of 6th International Symposium on Software Composition (SC'07), Vol. 4829 of LNCS, Springer, 2007, pp. 34–50.

[9] J. A. Goguen, J. Meseguer, Security Policies and Security Models, in: Proc. of the IEEE Symposium on Security and Privacy (SSP'82), IEEE Computer Society, 1982, pp. 11–20.

[10] A. Zakinthinos, E. S. Lee, A General Theory of Security Properties, in: Proc. of the IEEE Symposium on Security and Privacy (SSP'97), IEEE Computer Society, 1997, pp. 74–102.

[11] J. McLean, Security Models and Information Flow, in: Proc. of the IEEE Symposium on Security and Privacy (SSP'90), IEEE Computer Society, 1990, pp. 180–187.

[12] R. Focardi, S. Rossi, Information Flow Security in Dynamic Contexts, Journal of Computer Security 14 (1) (2006) 65–110.

[13] A. Sabelfeld, A. C. Myers, Language-Based Information-Flow Security, IEEE Journal on Selected Areas in Communication 21 (1) (2003) 5–19.

[14] E. M. Clarke, O. Grumberg, D. A. Peled, Model checking, The MIT Press, 1999.

[15] D. Kozen, Results on the Propositional μ-calculus, Theoretical Computer Science 27 (1983) 333–354.

[16] R. Milner, Communication and Concurrency, Vol. 92 of Prentice Hall International Series in Computer Science, Prentice Hall, 1989.

[17] M. Bravetti, G. Zavattaro, Contract Compliance and Choreography Conformance in the Presence of Message Queues, in: Proc. of the International Workshop on Web Services and Formal Methods (WS-FM'08), Vol. 5387 of LNCS, Springer, 2008, pp. 37–54.

[18] M. Tarek, C. Boutrous-Saab, S. Rampacek, Verifying correctness of web services choreography, in: ECOWS'06, 2006, pp. 306–318.

[19] M. Müller-Olm, Derivation of Characteristic Formulae, Electronic Notes in Theoretical Computer Science 18.

[20] A. Mader, Modal μ-calculus, Model Checking, and Gauss Elimination, in: Proc. of International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'95), Vol. 1019 of LNCS, Springer, 1995, pp. 72–88.

[21] G. Bernardi, M. Bugliesi, D. Macedonio, S. Rossi, A Theory of Adaptable Contract-Based Service Composition, in: Proc. of International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Workshop on Global Computing Models and Technologies (GlobalComp'08), IEEE Computer Society, 2008, pp. 327–334.

[22] T. Basciutti, Model-Checking Web Services, Master's thesis, Department of Computer Science, University Ca' Foscari of Venice (2010).

[23] R. Cleaveland, S. Sims, The NCSU Concurrency Workbench, in: Proc. of International Conference on Computer Aided Verification (CAV'96), Vol. 1102 of LNCS, Springer, 1996, pp. 394–397.

[24] G. Bernardi, A Theory of Adaptable Contract-Based Service Compositions, Master's thesis, Department of Computer Science, University Ca' Foscari of Venice (2009).

[25] A. Singhal, T. Winograd, K. Scarfone, Guide to Secure Web Services, Tech. Rep. 800-95, National Institute of Standards and Technology (2007).

[26] F. Abouzaid, J. Mullins, Model-checking Web Services Orchestrations using BP-calculus, Electronic Notes in Theoretical Computer Science 255 (2009) 3–21.

[27] G. Dai, X. Bai, C. Zhao, A Framework for Model Checking Web Service Compositions Based on BPEL4WS, in: Proc. of the IEEE International Conference on e-Business Engineering (ICEBE'07), IEEE Computer Society, 2007, pp. 165–172.

[28] S. Nakajima, Model-Checking Behavioral Specification of BPEL Applications, Electronic Notes in Theoretical Computer Science 151 (2006) 89–105.

[29] H. Schlingloff, A. Martens, K. Schmidt, Modeling and Model Checking Web Services, Electronic Notes in Theoretical Computer Science 126 (2005) 3–26.

[30] S. Nakajima, Model-Checking of Safety and Security Aspects in Web Service Flows, in: Proc. of the International Conference of Web Engineering (ICWE'04), Vol. 3140 of LNCS, Springer, 2004, pp. 488–501.