Model Checking Adaptive Service Compositions[☆]

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

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

Abstract

We present a logic-based verification framework for multilevel security and transactional correctness of service oriented architectures. The framework is targeted at the analysis of data confidentiality, enforced by non-interference, and of service responsiveness, captured by a notion of *compliance* that implies deadlock and livelock freedom. We isolate a class of modal μ-calculus formulae, interpreted over service configurations, that characterise configurations satisfying the properties of interest. We then investigate an adaptation technique based on the use of *coercion filters* to block any action that might potentially break security or transactional correctness. Based on the above, we devise a model checking algorithm for adaptive service compositions which automatically synthesises the maximal (most expressive / permissive) filter enforcing the desired security and correctness properties.

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

1. Introduction

Service Oriented Architectures (SOAs) and Service Oriented Computing (SOC) have emerged as leading paradigms to design interoperable, open-ended applications on the Internet. They rely on a series of XML-based standards to publish service interfaces as *contracts* (WSDL [1]), to structure the interaction among services so as to deliver the desired behaviour (WS-CDL [2] and BPEL [3]), and to secure point-to-point data exchanges (WS-Security) and more complex interaction

[☆] Work partially supported by the MIUR Projects IPODS "Interacting Processes in Open-ended Distributed Systems" and CINA "Compositionality, Interaction, Negotiation and Autonomicity".

[*] Corresponding author.

Email addresses: bugliesi@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

sessions (WS-Trust [4] and WS-SecureConversation [5]). Effective as they are as engineering tools, such standards are instead inadequate for analysis and verification, as they lack the formal semantics required for certifying the security and the correctness properties they are intended to enforce.

In response to this weakness, a substantial body of research has targeted new models of communication-centred computing, to serve as the formal counterpart of current XML-based standards (see Section 7 for related work). In the present paper we make a step further along the same direction, by developing a uniform model-checking verification framework for multilevel security and transactional correctness. We target information flow security as a fundamental property in multilevel security, and focus on a core notion of *compliance* to formalise transactional correctness. These both represent critical properties in SOAs, given the growing deployment of SOAs as the backbone of IT infrastructures for business and citizen-centric services, where information at mixed security levels (e.g., classified vs public) is processes and exchanged across domains with different security policies and varying privacy constraints.

Drawing on earlier work on process algebraic characterisation of SOAs, [6, 7, 8, 9, 10, 11], we formalise service compositions within a process calculus whose terms represent service interfaces, i.e., behavioural contracts providing abstract descriptions of service behaviour. Multilevel security specifications are expressed within this calculus by means security annotations, dynamically associated with individual service components. We then formalise information flow security in terms of a notion of *non-interference* [12] which we adapt to service compositions to capture the information-flow properties of interest [13, 14, 15, 16]. Our notion of compliance, in turn, draws on the work of [10, 11] and corresponds to a strong condition that ensures the absence of deadlocks *and* livelocks: compliant compositions are those whose computations never get stuck or trapped into infinite loops without chances to exit.

Given such characterisation of the properties of interest, we develop a verification method based on model-checking [17]. For that purpose, we isolate a set of modal μ-calculus [18] formulae, interpreted over service configurations, to characterise compositions which satisfy our security and correctness criteria. As a further step, we investigate an adaptation algorithm that draws on the *filters* introduced in [7] to coerce the behaviour of a service composition as needed to enforce the security and correctness of interest, by blocking any action that might potentially break them: specifically, we show that our adaptation algorithm is able to automatically synthesize the maximal (most permissive) filter enforcing those properties.

We demonstrate our framework at work on the analysis of a federated login system based on *OpenId* [19]. This is an open standard for authentication, which allows users to create accounts with their preferred OpenID providers (e.g., Google,

Type envs	Γ	::=	$\emptyset \mid \Gamma, u : \varsigma \mid \Gamma, (u, a, v) : \varsigma$	$u, v \in \mathcal{P} \cup \mathcal{V},\ a \in \mathcal{A},\ \varsigma \in \Sigma$
Actions	φ	::=	$\bar{a}(q)@u \mid a(x)@u$	$a \in \mathcal{A},\ u \in \mathcal{P} \cup \mathcal{V}, x \in \mathcal{V}$
Contracts	σ	::=	$\mathbf{1} \mid \mathbf{x} \mid \varphi.\sigma \mid \sigma + \sigma \mid \sigma \lfloor_\Lambda \oplus {}_\Lambda \rfloor \sigma \mid \mathtt{rec}(\mathbf{x})\,\sigma$	
Compositions	C	::=	$p[\sigma] \mid C \parallel C$	

Table 1: Syntax

Facebook, Wordpress, ... etc.) and then use those accounts to sign-in and access any service from any of the federated websites. As we show, the analysis is effective in detecting potential weaknesses of the system, and in supporting a principled developing practice.

Plan of the paper. Section 2 introduces the calculus of services. Section 3 formalises the notions of non-interference and compliance. Section 4 provides the corresponding μ-calculus characterisations. Section 5 presents the adaptation algorithm. Section 6 reports on the case study. Section 7 reviews related work and Section 8 concludes the the paper.

New content. This paper revises the work in [20] and extends it to include a new, more expressive version of the calculus, proofs of the main results, an updated analysis of related work, as well an analysis of the effectiveness of our verification techniques on a realistic case study.

2. A Calculus of services

We represent service contracts as terms of a value passing CCS-like [21] 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 [11, 22, 23] as the parallel (and concurrent) composition of a set of entities, hereafter called *principals*, executing contracts. We presuppose denumerable sets of action names $\mathcal{A}$, ranged over by a, b, c, principal identities $\mathcal{P}$, ranged over by p, q, r, and variables $\mathcal{V}$, ranged over by x, y. Actions represent the basic unit of observable behaviour of the underlying services, while the principal names specify the peers providing the services.

Multilevel security policies are specified by means of type annotations determining the security levels of principal identities. 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. Accordingly, type environments include two classes of bindings: principal bindings $u : \varsigma$, and *link* bindings $(u, a, v) : \varsigma$, with

u, v and a as in Table 1. Links are ranged over by ℓ, and are bidirectional, hence (u, a, v), is the same as (v, a, u). Closed type environments, assign security levels to *ground* links and principals, containing no variables. We note with Λ type environments that only contain link bindings. The join and meet operators are lifted from security levels to type environments as expected. We tacitly assume that type environments are *consistent*, a condition that requires that all link bindings draw on principal names and variables bound in the environment: formally, whenever $(u, a, v) : \varsigma \in \Gamma$ then $u, v \in dom(\Gamma)$.

2.1. Syntax

The syntax of the calculus is reported in Table 1. $\mathbf{1}$ denotes a contract that has reached a successful state (trailing $\mathbf{1}$'s are often omitted). $\bar{a}(q)@u.\sigma$ sends message q (a principal name) on a to principal u and then behaves as σ: syntactically, u may be a variable, but it must be a name when the prefix is ready to fire. Dually, the input prefix $a(x)@u.\sigma$ waits for an input on a from an arbitrary or specific principal and then continues as σ. When u is a variable, it is bound upon input and has scope σ (just as x); when it is a principal name, it is to be matched by the principal name received on input. We remark that principal names are the only values admitted in our contract specifications: in fact, communicating principal names is useful to capture the dynamic structure of complex service compositions, and at the same time is all that is needed, as other data exchanged among the actual services may safely be disregarded and abstracted away in the services' specifications. $\mathtt{rec}(\mathbf{x})\,\sigma$ 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. $\sigma + \sigma'$ denotes an external choice, guided by the context, while $\sigma \lfloor_\Lambda \oplus \ _{\Lambda'} \rfloor \sigma'$ is an internal choice, made irrespective of the structure of the interacting components. Λ and Λ' are link environments (which must be closed when the internal choice is ready to fire): depending on the branch chosen, the current type environment is updated with Λ or Λ'. We let $\Lambda(\sigma)$ note the set of all link bindings occurring in the type environments (associated with the internal choices) of σ.

Contracts define the behaviour of the principals executing them: $p[\sigma]$ denotes principal $p \in \mathcal{P}$ executing contract σ. We say that a contract σ is *p-compatible* if (i) for all $\bar{a}(r)@q$ and $a(r)@q$ occurring in σ, $q \neq p$, and (ii) for all $(u, a, v) \in \Lambda(\sigma)$, $u \neq v$ and either u or v is p. A configuration $p_1[\sigma_1] \parallel \cdots \parallel p_n[\sigma_n]$ of principals must be *well-formed* [22] to make a legal service composition: (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 legal, 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. We say that $\Gamma \triangleright C$ 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. Labelled Transition Semantics

We define the dynamics of the calculus in terms of a labelled transition system (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 $\bar{a}(r)@p$, $a(r)@p$ and silent actions τ; δ ranges over service composition actions $a(r)_{p \to q}$, $\bar{a}(r)_{p \to q}$ and τ.

The first block of rules defines the successful states for contracts and compositions: these are the compositions which 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.

The second block of rules defines the transitions for contracts. They are mostly self-explanatory, though a few remarks are in order for the internal choice transitions. Internal choices act not only as non-deterministic branching operator, but also provide services with the ability to dynamically update the security level of their interactions with other services by appropriately choosing the specific type environment associated with each branch. Though syntactically localized within choices, dynamic security-policy updates do not necessarily require a real choice. Indeed, the following derived form ${}_\Lambda|\sigma \stackrel{def}{=} \sigma\lfloor_\Lambda \oplus {}_\Lambda\rfloor\sigma$ makes it possible to perform a dynamic update at any step along a contract execution. We assume the binding environments Λ_1 and Λ_2 of an internal choice to be closed when choice is ready to fire.

Each contract transition yields a corresponding transition for the principal hosting the contract. τ transitions for configurations arise from the execution of a local internal choice. In that 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 the τ transitions for compositions. This is the only constraint on the principal security levels we assume, to rule out explicit flow of information from high (trusted) to low (untrusted) principals. In section 3 we will will characterise implicit flows, in terms of non-interference.

Example 2.1. *Table 3 shows an example of a service contract composition. It consists of a* client C*, two* financial consulting services F_1 *and* F_2 *and a* stock quote service provider S*. Let* $\Sigma = \{L, H\}$ *with* $L \preceq H$ *and* Γ *be the type environment* $C : H$, $F_1 : L$, $F_2 : L$, $S : L$*. The composition* $\Gamma \triangleright M$ *is well-formed and*

Contract and composition satisfaction: $\sigma \checkmark$

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

Contract transitions: $\Gamma \rhd \sigma \xrightarrow{\lambda} \Gamma' \rhd \sigma'$

$$\Gamma \rhd a(x)@p.\sigma \xrightarrow{a(q)@p} \Gamma \rhd \sigma\{x := q\} \quad \Gamma \rhd a(x)@y.\sigma \xrightarrow{a(q)@p} \Gamma \rhd \sigma\{x, y := q, p\}$$

$$\Gamma \rhd \bar{a}(q)@p.\sigma \xrightarrow{\bar{a}(q)@p} \Gamma \rhd \sigma \qquad \Gamma \rhd \sigma_1 \lfloor_{\Lambda_1} \oplus {}_{\Lambda_2} \rfloor \sigma_2 \xrightarrow{\tau} \Gamma \sqcup \Lambda_i \rhd \sigma_i \quad (i = 1, 2)$$

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

Composition Transitions: $\Gamma \rhd C \xrightarrow{\delta} \Gamma' \rhd C'$

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

$$\frac{\Gamma \rhd C_1 \xrightarrow{a(r)_{p \to q}} \Gamma \rhd C_1' \quad \Gamma \rhd C_2 \xrightarrow{\bar{a}(r)_{p \to q}} \Gamma \rhd C_2'}{\Gamma \rhd C_1 \parallel C_2 \xrightarrow{\tau} \Gamma \rhd C_1' \parallel C_2'}$$

$$\frac{\Gamma \rhd \sigma \xrightarrow{\tau} \Gamma' \rhd \sigma'}{\Gamma \rhd p[\sigma] \xrightarrow{\tau} \Gamma' \rhd p[\sigma']} \quad \Gamma'(\ell) \preceq \Gamma(p),\ \forall \ell \in dom(\Gamma')$$

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

Table 2: Labelled transitions for contracts and compositions

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

$$\begin{aligned}
\sigma_C = \;& \overline{\texttt{inq}}@F_1.\overline{\texttt{inq}}@F_2.\texttt{plan}@F_1.\texttt{plan}@F_2. \\
& (\,(\,\overline{\texttt{agree}}@F_1.\overline{\texttt{close}}@F_2.\mathbf{1} \lfloor_{(C,F_1):H} \oplus {}_{(C,F_2):H} \rfloor \overline{\texttt{agree}}@F_2.\overline{\texttt{close}}@F_1.\mathbf{1}\,) \\
& \quad \lfloor_{\emptyset} \oplus {}_{\emptyset} \rfloor \overline{\texttt{close}}@F_1.\overline{\texttt{close}}@F_2.\mathbf{1}\,)
\end{aligned}$$

$$\sigma_F = \;\texttt{inq}@x.\overline{\texttt{lookup}}@S.\texttt{quote}@x.\overline{\texttt{plan}}@C.(\,\texttt{agree}@x.\mathbf{1} + \texttt{close}@x.\mathbf{1}\,)$$

$$\sigma_S = \;\texttt{lookup}@x.\overline{\texttt{quote}}@x.\mathbf{1}$$

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

Table 3: Example: a financial consulting platform

consists of four services: $C[\sigma_C]$, $F_1[\sigma_F]$ and $F_2[\sigma_F]$ and $S[\sigma_S]$. The elementary actions represent business activities that result in messages being sent or received. For example, the action $\overline{\texttt{inq}}@F_1$ undertaken by the customer corresponds to the request being sent to the first financial consulting service. Throughout, we use the simplified syntax $a@p$ and $\bar{a}@p$ whenever the values exchanged by two synchronizing actions may be disregarded, as it happens in the current (we see value passing fully at work in Section 6, on the case study). Also, we use a shorthand for link bindings writing $(p, q) : \varsigma$ to note the set of link bindings $(p, a, q) : \varsigma$ for all $a \in \mathcal{A}$. In the example, the client inquires with 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 either to adhere to the investment plan proposed by one of the financial services and close the connection with the other one or not to adhere to any of the two and hence close both the connections.

The following, simple proposition proves a sanity condition for the transition system, namely that τ transitions map configurations into configurations: as a consequence, τ transitions may safely be chained.

Proposition 2.1. *Let Γ be a type environment and C be a service composition such that $\Gamma \rhd C$ is a configuration. If $\Gamma \rhd C \xrightarrow{\tau} \Gamma' \rhd C'$, then $\Gamma' \rhd C'$ is a configuration.*

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 enclos-

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

$$\frac{\Gamma \rhd C_1 \xrightarrow{a(r)_{p \to q}} \Gamma \rhd C_1' \quad \Gamma \rhd C_2 \xrightarrow{\bar{a}(r)_{p \to q}} \Gamma \rhd C_2'}{\Gamma \rhd C_1 \parallel C_2 \xhookrightarrow{p\langle a \rangle q} \Gamma \rhd C_1' \parallel C_2'}$$

Table 4: Labelled synchronisation transitions: $\Gamma \rhd C \xhookrightarrow{\alpha} \Gamma' \rhd C'$

ing input prefix. Now, given that $\Gamma \rhd C$ is well-formed, that input may never synchronise with an output from p itself: hence x will never get bound to p. $\square$

A computation for a configuration $\Gamma \rhd C$, may now be defined as a sequence $\Gamma \rhd C = \Gamma_0 \rhd C_0 \xrightarrow{\tau} \Gamma_1 \rhd C_1 \xrightarrow{\tau} \ldots$ of internal actions. We write $\Longrightarrow$ to denote the reflexive and transitive closure of $\xrightarrow{\tau}$, and $\overset{\delta}{\Longrightarrow}$ for $\Longrightarrow\xrightarrow{\delta}\Longrightarrow$. The notation is extended to sequences of actions: with $w = \delta_1 \ldots \delta_n$, we write $\overset{w}{\Longrightarrow}$ to note $\overset{\delta_1}{\Longrightarrow} \cdots \overset{\delta_n}{\Longrightarrow}$. When the configuration typing environments may safely be disregarded, we write $C \Longrightarrow C'$ to note a transition of $\Gamma \rhd C \Longrightarrow \Gamma' \rhd C'$ for some appropriate Γ and Γ'.

Based on the labelled transition relation given in Table 2, we introduce a further transition relation that will be instrumental to define the behavioural invariants which constitute the targets of our verification framework. The new relation applies to compositions only, and allows us (i) to distinguish a local τ move determined by a contract internal choice from a τ move resulting from a distributed synchronisation, and (ii) to make the components involved in every synchronisation explicit. The new relation is defined in Table 4. Notice that the τ label now indicates an internal action, local to a single service component, while synchronisations between different components composition are represented through a label of the form $p\langle a \rangle q$ making it explicit that principals p and q synchronise on action a (exchanging some, unspecified, data).

We let α range over the labels $p\langle a \rangle q$ and τ and write $\xhookrightarrow{\tau}\!\!\!\!\twoheadrightarrow$ for a possible empty sequence of $\xhookrightarrow{\tau}$. Also, we define $\xhookrightarrow{p\langle a \rangle q}\!\!\!\!\twoheadrightarrow \overset{def}{=} \xhookrightarrow{\tau}\!\!\!\!\twoheadrightarrow \xhookrightarrow{p\langle a \rangle q} \xhookrightarrow{\tau}\!\!\!\!\twoheadrightarrow$, and note by $\xhookrightarrow{\gamma}\!\!\!\!\twoheadrightarrow$ the sequence of transitions $\xhookrightarrow{\alpha_1}\!\!\!\!\twoheadrightarrow \xhookrightarrow{\alpha_2}\!\!\!\!\twoheadrightarrow \ldots \xhookrightarrow{\alpha_n}\!\!\!\!\twoheadrightarrow$ for $\gamma = \alpha_1, \ldots, \alpha_n$.. As for $\Longrightarrow$, we omit the typing environment when irrelevant and write $C \xhookrightarrow{\gamma}\!\!\!\!\twoheadrightarrow C'$ to denote a derivation $\Gamma \rhd C \xhookrightarrow{\gamma}\!\!\!\!\twoheadrightarrow \Gamma' \rhd C'$ for given Γ and Γ'.

The following lemma relates the two semantics for service compositions.

Lemma 2.1. *Let $\Gamma \rhd C$ be a configuration.*

- $\Gamma \rhd C \overset{\tau}{\longmapsto} \Gamma' \rhd C'$ *if and only if* $C = C_1 \parallel p[\sigma] \parallel C_2$, $C' = C_1 \parallel p[\sigma'] \parallel C_2$ *and* $\sigma \overset{\tau}{\longrightarrow} \sigma'$;
- $\Gamma \rhd C \overset{p\langle a\rangle q}{\longmapsto} \Gamma \rhd C'$ *if and only if* $C = C_1 \parallel p[\sigma] \parallel C_2 \parallel q[\rho] \parallel C_3$, $C' = C_1 \parallel p[\sigma'] \parallel C_2 \parallel q[\rho'] \parallel C_3$, $\sigma \overset{\bar{a}(r)@q}{\longrightarrow} \sigma'$ *and* $\rho \overset{a(r)@p}{\longrightarrow} \rho'$ *for some* r.

Proof. For each item, by induction on the derivation in the hypothesis. □

2.3. Behavioural Observations and Observational Equivalence

Given the security-dependent definition of the semantics, the notions of behavioural observation and observational equivalence that arise for the calculus are naturally parametric in the security level at which the observations are made (equivalently, in the security level of the behaviour under observation). To make the definition formal, we first associate a security level with each synchronisation transition. Let $\Gamma(\alpha)$ denote the security level of (the label associated with) a $\longmapsto$ transition with respect to the environment Γ, defined as follows:

$$\Gamma(\tau) \;=\; \bot, \quad \Gamma(p\langle a\rangle q) \;=\; \Gamma(p, a, q)$$

Our notion of equivalence is then formalised as a relation over configurations that equate service compositions exhibiting the same ς-level component interactions. The formal definition yields variant of the notion of *weak bisimulation* [21], 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 Γ_1 and Γ_2 have the same set of (principal and link) bindings at security levels $\preceq \varsigma$.

Definition 2.1 (Weak ς-bisimulation). Let $\varsigma \in \Sigma$. A *weak ς-bisimulation* 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 $\Gamma_1 =_\varsigma \Gamma_2$

(1) if $\Gamma_1 \rhd C_1 \overset{\alpha}{\longmapsto} \Gamma_1' \rhd C_1'$ with $\Gamma_1(\alpha) \preceq \varsigma$, then there exist Γ_2' and C_2' such that $\Gamma_2 \rhd C_2 \overset{\alpha}{\longmapsto\!\!\!\!\!\rightarrow} \Gamma_2' \rhd C_2'$ with $\Gamma_1' \rhd C_1' \approx_\varsigma \Gamma_2' \rhd C_2'$ and $\Gamma_1' =_\varsigma \Gamma_2'$;

(2) if $\Gamma_1 \rhd C_1 \overset{\alpha}{\longmapsto} \Gamma_1' \rhd C_1'$ with $\Gamma(\alpha) \not\preceq \varsigma$, then there exist Γ_2' and C_2' such that

- either $\Gamma_2 \rhd C_2 \overset{\alpha}{\longmapsto\!\!\!\!\!\rightarrow} \Gamma_2' \rhd C_2'$
- or $\Gamma_2 \rhd C_2 \overset{\tau}{\longmapsto\!\!\!\!\!\rightarrow} \Gamma_2' \rhd C_2'$

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 $\Gamma \rhd C_1 \approx_\varsigma \Gamma \rhd C_2$. □

3. Multilevel security and correctness of service compositions

As anticipated, our analysis and verification methods are targeted at two fundamental properties in web service architectures, namely: information flow security as a fundamental property to protect against unintended leaks of sensitive data and information, and *compliance* as a formal certification of transactional correctness.

3.1. Non-interference for information-flow security

The characterisation of information-flow security in terms of non-interference in multi-level security systems has been studied extensively in the literature since the seminal work by Goguen and Meseguer [12]. Non-interference may effectively be employed for our present purposes to ensure that public interactions between services are independent of any secret communications or, more generally, that the low-level behaviour observed of a service composition is independent of the behaviour of (and the sensitive data available at) its high-level components. A non-interferent service composition will therefore guarantee that the sensitive data (e.g., authentication credentials, secret cookies, or even search queries) that a client transmits to a server remain confidential and are not leaked, directly, or indirectly, to any unintended component or eavesdropper.

Our notion of non-interference is inspired by [15] and is expressed in terms of a restriction operator $\cdot|_\varsigma$ which allows one to represent a service composition prevented from performing internal synchronisations at a level higher than ς. The semantics of $C|_\varsigma$ is described by the following rule:

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

Definition 3.1 (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 in Table 3. One readily sees that $M \notin \mathcal{NI}_{\Gamma,L}$, as there is a direct causality between the high-level synchronization $C\langle\texttt{agree}\rangle F_i$ and the low-level synchronization $C\langle\texttt{close}\rangle F_j$ 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 (F_2) then an external observer knows that the*

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

$$\begin{aligned}\sigma'_C = \quad & \overline{\texttt{inq}}@F_1.\overline{\texttt{inq}}@F_2.\texttt{plan}@F_1.\texttt{plan}@F_2. \\ & (\,(\,\overline{\texttt{agree}}@F_1.\overline{\texttt{close}}.@F_2.\mathbf{1}\lfloor_{(C,F_1):H} \oplus {}_{(C,F_2):H}\rfloor\overline{\texttt{agree}}@F_2.\overline{\texttt{close}}@F_1.\mathbf{1}\,) \\ & \quad \lfloor\emptyset \oplus \emptyset\rfloor(\,\overline{\texttt{close}}@F_1.\overline{\texttt{close}}@F_2.\mathbf{1}\lfloor_{(C,F_2):H} \oplus {}_{(C,F_1):H}\rfloor\overline{\texttt{close}}@F_2.\overline{\texttt{close}}@F_1.\mathbf{1}\,)\,)\end{aligned}$$

Table 5: Example: non-interference for the financial consulting platform ($F_1[\sigma_F]$, $F_2[\sigma_F]$ and $S[\sigma_S]$ as in Table 3).

customer has agreed to proceed with investment recommendation of F_1 (F_2) by just observing that the action $C\langle\texttt{close}\rangle F_2$ ($C\langle\texttt{close}\rangle F_1$) has been performed. Indeed, if the customer chose not to adhere to any of the two proposed plans, the external observer would notice two `close` *messages. One obvious fix is to swap the the order between the two actions. A more interesting and practically robust solution, is to inject noise into the client contract, so as to fool a low-clearance observer as shown in σ'_C in Table 5.*

3.2. Compliance for transactional correctness

Compliance is a core property that characterises the correct behaviour of concurrent distributed systems. It is used widely in the context of SOAs as a formal device to identify service compositions that provide guarantees of service responsiveness, as they are free of synchronisation errors. For our present endeavour, we refer to the notion of compliance for contract service compositions studied in [24]. 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 synchronisations.

Definition 3.2 (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 $C''\checkmark$. □

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 synchronisations). This is enough to prevent both deadlocks and livelocks.

The notion of compliance can be equivalently expressed in terms of the labelled synchronisation transitions $\hookrightarrow$.

Proposition 3.1. *Let C be a contract service composition. It holds that C is* compliant*, $C\downarrow$, if and only if every C' such that $C \stackrel{\gamma'}{\longmapsto\!\!\!\!\!\rightarrow} C'$ for some $\gamma' \in Act^*$ there exist C'' and $\gamma'' \in Act^*$ such that $C' \stackrel{\gamma''}{\longmapsto\!\!\!\!\!\rightarrow} C''$ and $C''\checkmark$.*

Proof. By induction on the derivations and Lemma 2.1. □

4. Modal μ-calculus characterisation

In this section we set the foundations for our model-checking verification framework, by showing that the properties of non-interference and compliance we just introduced can be characterized exactly by means of *characteristic formulae* expressed in μ-calculus. Based on such characterization, the properties of interest in a given composition may be verified by model checking the composition's characteristic formula with, e.g. the NCSU Concurrency Workbench model checker [25].

4.1. Background

The modal μ-calculus [18] is a propositional temporal logic that allows one to express liveness, safety, fairness and cyclic properties of processes. The formulae of the logic, given in positive normal form, are defined by the following productions:

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

Here X ranges over an infinite set of variables and, for our present purposes, we take α to range over the labels $p\langle a\rangle q$ and τ. The least and greatest fixpoint operators (respectively μX and νX) are binders for the recursion variable X. We adopt the usual notions of free and bound variables in a formula and of closed formula. Also, for a finite set $\mathcal{M}$ of formulae, we write $\bigwedge \mathcal{M}$ and $\bigvee \mathcal{M}$ for the conjunction and the disjunction of the formulae in $\mathcal{M}$, where $\bigwedge \emptyset = \mathbf{true}$ and $\bigvee \emptyset = \mathbf{false}$.

We interpret the μ-calculus formulae over configurations whose semantics is expressed in terms of labelled transition systems as defined in Table 4. Given a configuration $\Gamma \rhd C$, we denote by $S_{\Gamma\rhd C}$ the set of all states reachable from $\Gamma \rhd C$ via $\stackrel{\alpha}{\longmapsto\!\!\!\!\!\rightarrow}$. We denote by $M_{\Gamma\rhd C}(\phi)(\rho)$ the subset of $S_{\Gamma\rhd C}$ that satisfy a formula ϕ, where ρ provides the interpretation *environment*, i.e., a partial mapping from Var to $2^{S_{\Gamma\rhd C}}$ that interprets at least the free variables of ϕ by subsets of $S_{\Gamma\rhd C}$. As usual, $\rho[X \mapsto x]$ is the environment that maps the variable X to the set $x \subseteq S_{\Gamma\rhd C}$ and $Y \neq X$ to $\rho(Y)$ if ρ is defined on Y.

The formal definition of $M_{\Gamma\rhd C}(\phi)(\rho)$ is given in Table 6. Intuitively, $\mathbf{true}$ and $\mathbf{false}$ hold for all, resp. no, states and $\wedge$ and $\vee$ are interpreted by conjunction

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

Table 6: Semantics of modal mu-calculus

and disjunction. The formula $\langle \alpha \rangle \phi$ 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 interpretation of the formula $\mu X.\phi$ (resp. $\nu X.\phi$) is the smallest (resp. largest) subset x of $S_{\Gamma \rhd C}$ that recurs when ϕ is interpreted with the substitution of x for X. The existence of such sets follows from the well-known Knaster-Tarski fixpoint theorem. Since the interpretation of a closed formula 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

$$Conf(\phi) = \{\Gamma \rhd C \mid \Gamma \rhd C \in M_{\Gamma \rhd C}(\phi)\}.$$

As usual, in order to derive a formula characterizing a process property we also refer to (closed) *equation systems* of the form

$$Eqn_{\Gamma \rhd C} :\quad X_1 = \phi_1 \quad \ldots \quad X_n = \phi_n$$

where $X_1, \ldots, X_n$ are mutually distinct variables and $\phi_1, \ldots, \phi_n$ are formulae having at most $X_1, \ldots, X_n$ as free variables. We say that an environment $\rho : \{X_1, \ldots, X_n\} \to 2^{S_{\Gamma \rhd C}}$ is a *solution* of the equation system $Eqn_{\Gamma \rhd C}$, if $\rho(X_i) = M_{\Gamma \rhd C}(\phi_i)(\rho)$. The fact that a solution always exist, is again a consequence of the Knaster-Tarski fixpoint theorem.

The solutions of $Eqn_{\Gamma \rhd C}$ are the fixpoints of the *equation functional* $Func^{Eqn}_{\Gamma \rhd C} : Env_{\Gamma \rhd C} \to Env_{\Gamma \rhd C}$ defined by $Func^{Eqn}_{\Gamma \rhd C}(\rho)(X_i) = M_{\Gamma \rhd C}(\phi_i)(\rho)$ for $i \in [1..n]$. Since $Func^{Eqn}_{\Gamma \rhd C}$ is monotonic, the largest solution (with respect to $\sqsubseteq$) $\nu Func^{Eqn}_{\Gamma \rhd C}$ of $Env_{\Gamma \rhd C}$ exists and we denote it by $M_{\Gamma \rhd C}(Eqn_{\Gamma \rhd C})$. 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

$$Conf(Eqn) = \{\Gamma \rhd C \mid \Gamma \rhd C \in M_{\Gamma \rhd C}(Eqn)(X_1)\}.$$

4.2. Characterizing weak ς-bisimulation and non-interference

The relation $\approx_\varsigma$ can be characterized as the greatest fixpoint $\nu Func_{\approx_\varsigma}$ of the monotonic functional $Func_{\approx_\varsigma}$ on the complete lattice of relations $\mathcal{R}$ over configurations ordered by set inclusion such that $(\Gamma_1 \rhd C_1, \Gamma_2 \rhd C_2) \in Func_{\approx_\varsigma}(\mathcal{R})$ if and only if points (1) and (2) of Definition 2.1 hold. The largest weak ς-bisimulation is the greatest fixpoint $\nu Func_{\approx_\varsigma}$ of $Func_{\approx_\varsigma}$ (see, e.g., [26]).

Following [27], we show how a formula characterizing a finite-state configuration $\Gamma \rhd C$ up to $\approx_\varsigma$ can be derived from the fixpoint characterization of $\approx_\varsigma$ described above.

Let $\Gamma \rhd C$ be a configuration with $S_{\Gamma \rhd C} = \{\Gamma_1 \rhd C_1, \ldots, \Gamma_n \rhd C_n\}$, 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* consisting of one equation for each service configuration $\Gamma_1 \rhd C_1, \ldots, \Gamma_n \rhd C_n \in S_{\Gamma \rhd C}$ as follows:

$$\begin{array}{rrcl} Eqn_{\approx_\varsigma}: & X_{\Gamma_1 \rhd C_1} & = & \phi^{\approx_\varsigma}_{\Gamma_1 \rhd C_1} \\ & & \vdots & \\ & X_{\Gamma_n \rhd C_n} & = & \phi^{\approx_\varsigma}_{\Gamma_n \rhd C_n} \end{array}$$

Here the formulae $\phi^{\approx_\varsigma}_{\Gamma_i \rhd C_i}$ are defined so that the largest solution $M_{\Gamma \rhd C}(Eqn_{\approx_\varsigma})$ of $Eqn_{\approx_\varsigma}$ 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 ς-bisimilar to $\Gamma_i \rhd C_i$, i.e., such that $M_{\Gamma \rhd C}(Eqn_{\approx_\varsigma})(X_{\Gamma_i \rhd C_i}) = \{\Gamma'_i \rhd C'_i \in S_{\Gamma \rhd C} \mid \Gamma_i \rhd C_i \approx_\varsigma \Gamma'_i \rhd C'_i\}$. Theorem 4.1 shows the exact form of such formulae. First we define:

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

where $\langle\!\langle \tau \rangle\!\rangle \phi \stackrel{def}{=} \mu X.\phi \vee \langle \tau \rangle X$ and $\langle\!\langle \alpha \rangle\!\rangle \phi \stackrel{def}{=} \langle\!\langle \tau \rangle\!\rangle \langle \alpha \rangle \langle\!\langle \tau \rangle\!\rangle \phi$. Let $\stackrel{\alpha}{\hookrightarrow\!\!\!\!\!\rightarrow}_{\Gamma,\varsigma}$ note either $\stackrel{\alpha}{\hookrightarrow\!\!\!\!\!\rightarrow}$ or $\stackrel{\tau}{\hookrightarrow\!\!\!\!\!\rightarrow}$. Then $\langle\!\langle \alpha \rangle\!\rangle_{\Gamma,\varsigma}$, $\langle\!\langle \tau \rangle\!\rangle$ and $\langle\!\langle \alpha \rangle\!\rangle$ correspond to $\stackrel{\alpha}{\hookrightarrow\!\!\!\!\!\rightarrow}_{\Gamma,\varsigma}$, $\stackrel{\tau}{\hookrightarrow\!\!\!\!\!\rightarrow}$ and $\stackrel{\alpha}{\hookrightarrow\!\!\!\!\!\rightarrow}$, since

- $M_{\Gamma \rhd C}(\langle\!\langle \alpha \rangle\!\rangle_{\Gamma,\varsigma} \phi)(\rho) = \{\Gamma' \rhd C' \mid \exists\, \Gamma'' \rhd C'' : \Gamma' \rhd C' \stackrel{\alpha}{\longmapsto\!\!\!\!\!\rightarrow}_{\Gamma,\varsigma} \Gamma'' \rhd C'' \wedge \Gamma'' \rhd C'' \in M_{\Gamma \rhd C}(\phi)(\rho)\}$

- $M_{\Gamma \rhd C}(\langle\!\langle \tau \rangle\!\rangle \phi)(\rho) = \{\Gamma' \rhd C' \mid \exists\, \Gamma'' \rhd C'' \;:\; \Gamma' \rhd C' \stackrel{\tau}{\longmapsto\!\!\!\!\!\rightarrow} \Gamma'' \rhd C'' \wedge \Gamma'' \rhd C'' \in M_{\Gamma \rhd C}(\phi)(\rho)\}$

- $M_{\Gamma \rhd C}(\langle\!\langle \alpha \rangle\!\rangle \phi)(\rho) = \{\Gamma' \rhd C' \mid \exists \Gamma'' \rhd C'' \;:\; \Gamma' \rhd C' \stackrel{\alpha}{\longmapsto\!\!\!\!\!\rightarrow} \Gamma'' \rhd C'' \wedge \Gamma'' \rhd C'' \in M_{\Gamma \rhd C}(\phi)(\rho)\}$

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

$$\bigwedge\{\bigwedge\{\langle\!\langle \alpha \rangle\!\rangle_{\Gamma,\varsigma} X_{\Gamma'_i \rhd C'_i} \mid \Gamma_i \rhd C_i \stackrel{\alpha}{\longmapsto\!\!\rightarrow} \Gamma'_i \rhd C'_i\}\}$$
$$\wedge \bigwedge\{[\alpha] \bigvee\{X_{\Gamma'_i \rhd C'_i} \mid \Gamma_i \rhd C_i \stackrel{\alpha}{\longmapsto\!\!\!\!\!\rightarrow}_{\Gamma,\varsigma} \Gamma'_i \rhd C'_i\}\}.$$

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

Proof. Observe that $Env_{\Gamma \rhd C}$, the set of candidates for solutions of $Eqn_{\approx_\varsigma}$, is order-isomorphic to $2^{S_{\Gamma \rhd C} \times S_{\Gamma \rhd C}}$, the set of relations that are candidates to be weak ς-bisimulations between $S_{\Gamma \rhd C} \times S_{\Gamma \rhd C}$.

Let us consider the mapping $\zeta : Env_{\Gamma \rhd C} \to 2^{S_{\Gamma \rhd C} \times S_{\Gamma \rhd C}}$ defined by:

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

ζ is an order isomorphism between $Env_{\Gamma \rhd C}$ and $2^{S_{\Gamma \rhd C} \times S_{\Gamma \rhd C}}$. The inverse of ζ is the mapping $\eta : 2^{S_{\Gamma \rhd C} \times S_{\Gamma \rhd C}} \to Env_{\Gamma \rhd C}$ defined by

$$\eta(\mathcal{R})(X_{\Gamma_i \rhd C_i}) = \{\Gamma'_i \rhd C'_i \in S_{\Gamma \rhd C} \mid (\Gamma_i \rhd C_i, \Gamma'_i \rhd C'_i) \in \mathcal{R}\}.$$

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

$$Func^{Eqn_{\approx_\varsigma}}_{\Gamma \rhd C} = \eta \circ Func_{\approx_\varsigma} \circ \zeta.$$

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

For any equation system Eqn there is a characteristic formula ϕ such that $Conf(Eqn) = Conf(\phi)$. In particular, characteristic formulae characterizing configurations can be simply derived from $Eqn_{\approx_\varsigma}$ by applying semantics-preserving transformation rules in the style of [28].

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*

$$Conf(\phi^{\approx_\varsigma}(\Gamma \rhd C)) = \{\Gamma' \rhd C' \in S_{\Gamma \rhd C} \mid \Gamma' \rhd C' \approx_\varsigma \Gamma \rhd C|_\varsigma\}.$$

Proof. Consider the characteristic equation system $Eqn_{\approx_\varsigma}$ consisting of one equation for each service configuration $\Gamma_1 \rhd C_1, \ldots, \Gamma_n \rhd C_n \in S_{\Gamma \rhd C|_\varsigma}$ where $\Gamma_1 \rhd C_1 = \Gamma \rhd C|_\varsigma$. Following [27], the formula $\phi^{\approx_\varsigma}(\Gamma \rhd C)$ is obtained from $Eqn_{\approx_\varsigma}$ by applying three simple semantics-preserving transformation rules: the first rule removes the recursive dependencies from the right hand side and the left hand side of any equation; the second rule substitutes variables on the left hand side of an equation by the corresponding formula on the right hand side in the other equations; the third rule removes unnecessary equations. The formula $\phi^{\approx_\varsigma}(\Gamma \rhd C)$ satisfies the property that

$$\begin{array}{rcl} Conf(\phi^{\approx_\varsigma}(\Gamma \rhd C)) & = & Conf(Eqn_{\approx_\varsigma}) \\ & = & \{\Gamma' \rhd C' \mid \Gamma' \rhd C' \in M_{\Gamma \rhd C|_\varsigma}(Eqn_{\approx_\varsigma})(X_{\Gamma \rhd C|_\varsigma})\} \\ & = & \{\Gamma' \rhd C' \in S_{\Gamma \rhd C} \mid \Gamma' \rhd C' \approx_\varsigma \Gamma \rhd C|_\varsigma\}. \end{array}$$

□

4.3. Characterizing Compliance

The construction of the characteristic formula for compliance is more direct, and may be given as follows:

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

where

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

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 [29].

Theorem 4.3. *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$\}$.*

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

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

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

Table 7: Dynamics of Filtered contract service compositions

Proof. The fact that $Conf(\phi^c) \supseteq \{\Gamma \rhd C \mid C\downarrow \text{ and } \Gamma \text{ is a type environment}\}$ 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 $\mu X.\big((\checkmark) \vee \bigvee_{\alpha\in Act}(\langle\alpha\rangle X)\big)$. □

Corollary 4.1. *A composition C is compliant if and only if $\Gamma \rhd C \in Conf(\phi^c)$ for some type environment Γ.* □

As a consequence of Theorems 4.2 and 4.3 we have:

Corollary 4.2. *Let $\varsigma \in \Sigma$, $\Gamma \rhd C$ be a configuration and*

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

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 $C\downarrow$. □

5. An adaptation 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 [7] and revised in [24], 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.

$$\begin{array}{rcl} f \in \mathcal{F} & ::= & \mathbf{0} \mid \delta.f \mid f \times f \mid f \otimes f \mid X \mid \mathtt{rec}(\mathbf{X})\, f \\ \delta & ::= & a_{p\to q} \mid \bar{a}_{p\to q} \end{array}$$

With a slight abuse of notation, we overload the symbol δ employed to range over contract labels, to note filter actions. Note however, that the former include the values exchanged by the contract actions, while the latter disregard such values. When there is a risk of confusion, we write $\delta(r)$ for the contract label associated with the corresponding filter action δ (e.g. $a(r)_{p\to q}$ vs $a_{p\to q}$).

Definition 5.1 (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 $\mathbf{0}$ as bottom and the identity filter $I_A \overset{def}{=} \mathtt{rec}(\mathbf{X}) \prod_{\delta\in A} \delta.X$ 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 $\bullet \in \{\times, \otimes\}$:

$$F \leq_\pi G \quad \text{iff} \quad F[p] \leq G[p] \quad \text{for all } p \in \pi$$
$$(F \bullet_\pi G)[p] \quad \stackrel{def}{=} \quad F[p] \bullet G[p] \quad \text{for all } p \in \pi$$

We generalise 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 synchronisations), 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 5.2 (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 $f \stackrel{\delta}{\longmapsto} \widehat{f}$ one has $\delta \in \{a_{\_\to p}, \overline{a}_{p\to\_}\}$ and there exists $\Gamma \rhd C \in \mathcal{C}$ such that $\Gamma \rhd C \stackrel{\alpha}{\hookrightarrow\!\!\!\!\rightarrow}$ with $\alpha \in \{\_\langle a\rangle p, p\langle a\rangle\_\}$ and $\widehat{f} \propto_p \{\Gamma' \rhd C' \mid \Gamma \rhd C \stackrel{\alpha}{\hookrightarrow\!\!\!\!\rightarrow} \Gamma' \rhd C'\}$.
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 $F \propto \{\Gamma \rhd C\}$. $\square$

The Algorithm. We describe an algorithm that synthesises 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 synchronisations 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

Algorithm 1: Procedure PushLabels(G)

Input: A reduction graph $G = (V, E)$
Output: The graph G updated

```
done := false;
while ¬ done do
    done := true;
    foreach u ∈ V do
        succ := false;  fail := false;
        if Adj[u, τ] ≠ ∅ then
            if ∃v ∈ Adj[u, τ] : result[v] = FAIL then
                fail := true;
            else if ∃v∈Adj[u, τ] : result[v] = SUCC then
                succ := true;
        else if ∃(α, v) ∈ Adj[u] ∧ result[v] = SUCC ∧ ¬Conflict(α, u)
        then
            succ := true;
        if succ ∧ result[u] ≠ SUCC then
            result[u] := SUCC;  done := false;
        else if fail ∧ result[u] ≠ FAIL then
            result[u] := FAIL;  done := false
```

$\Phi^{\varsigma}_{\Gamma \rhd C}$ or not as calculated by the model checker. An edge in E is a triple $(\mathbf{u}, \mathbf{v})_\alpha$ representing the transition $state[\mathbf{u}] \xhookrightarrow{\alpha} state[\mathbf{v}]$. 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 $\alpha = p\langle a \rangle q$, and $\emptyset$ in case $\alpha = \tau$. Then,

let $G = (V, E)$ be a reduction graph, and $\alpha = p\langle a \rangle q$.

- 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 $result[\mathbf{w}_1] \neq result[\mathbf{w}_2]$.
- 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' synchronisations, and by preserving all the 'good' synchronisations that lead to nodes satisfying both non-interference and compliance. Due to the presence of internal choices, the same synchronisation can look good at one point, but actually be bad. The definition of conflict formally captures this notion of ambiguous synchronisations.

Proposition 5.1. *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* $result[\mathbf{v}] = \text{SUCC}$*.*

Proof. The proof easily follows by construction. □

We say that a path ϖ in G is *successful* if $result[\mathbf{u}] = \text{SUCC}$ 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 5.2. *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 synthesises 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 $F^{Alg}[\Phi^{\varsigma}_{\Gamma \triangleright C}] =$ `ExtractFilter`$_\pi(root[G], \emptyset, G')$. A complete proof of the next theorem is given in [30].

Algorithm 2: Function SuccessGraph(G)

Input: A reduction graph $G = (V, E)$
Output: $G' = (V', E')$ the success sub-graph of G

$V' := (\mathit{result}[\mathit{root}[G]] = \textsc{succ})\,?\,\{\mathit{root}[G]\} : \emptyset$; $E' := \emptyset$;
$\mathit{done} := \texttt{false}$;
while $\neg\, \mathit{done}$ **do**
  $\mathit{done} := \texttt{true}$;
  foreach $(\mathbf{u}, \mathbf{v})_\alpha \in E \setminus E'$ **do**
    if $\mathbf{u} \in V' \wedge \mathit{result}[\mathbf{v}] = \textsc{succ} \wedge \neg \mathit{Conflict}(\alpha, \mathbf{u})$ **then**
      $V' := V' \cup \{\mathbf{v}\}$; $E' := E' \cup \{(\mathbf{u}, \mathbf{v})_\alpha\}$;
      $\mathit{done} := \texttt{false}$
return $G' = (V', E')$;

Theorem 5.1 (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 fact that $F^{Alg}[\Phi^{\varsigma}_{\Gamma \rhd C}](C) \in \mathcal{NI}_{\Gamma,\varsigma}$ follows from the assumption that the input of our algorithm is the reduction graph obtained by applying a global model checking algorithm to $\Gamma \rhd C$ and the modal formula $\Phi^{\varsigma}_{\Gamma \rhd C}$, i.e., consisting of all the states reachable from $\Gamma \rhd C$ and satisfying $\Phi^{\varsigma}_{\Gamma \rhd C}$. The rest of the proof follows by construction and by Propositions 5.1 and 5.2. □

6. A case study: authentication services

We demonstrate our framework at work on a real-world case study of a web-service architecture implementing a federated authentication system. First we introduce the system description and discuss how our calculus copes with it. Then, we show that our verification framework may effectively be employed to prevent and protect against subtle architectural design flaws.

Algorithm 3: Function ExtractFilter$_\pi(\mathbf{u}, U, G)$

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

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

6.1. System definition

Authentication services are web services that allow web applications to partially or totally avoid managing users' accounts. They simplify user access to web services, by making it possible for users to hold just one account to access several independent services, and ease the design of web applications by factoring the storage and management of user accounts out into a specialised component. A popular implementation of such mechanisms is provided by the *OpenID* standard [19]. A user wishing to obtain an OpenID just needs to register with one of the providers that support the standard (Google, Yahoo! and Wordpress are probably the most common). Web applications and services that rely on OpenID for authentication redirect their users to one of the OpenID providers upon login: upon receiving the user credentials, the authentication server recognises the user and reports back the results of the successful authentication both to the user and to the web application. This process is illustrated in Figure 1 and consists of nine steps:

1. The Web Application WA offers the user a set of login options;
2. The user selects to log-in with a third party account, choosing among OpenID-compatible providers;
3. WA sends a *discovery* request to the selected provider in order to obtain a login web service endpoint;
4. The provider answers with an eXtensible Resource Descriptor Sequence (XRDS) document containing the address(es) of the login web service endpoint;
5. 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 Authentication Server AS sends to WA a token which confirms the user's identity;
9. Finally, WA is able to exchange confidential information to the user.

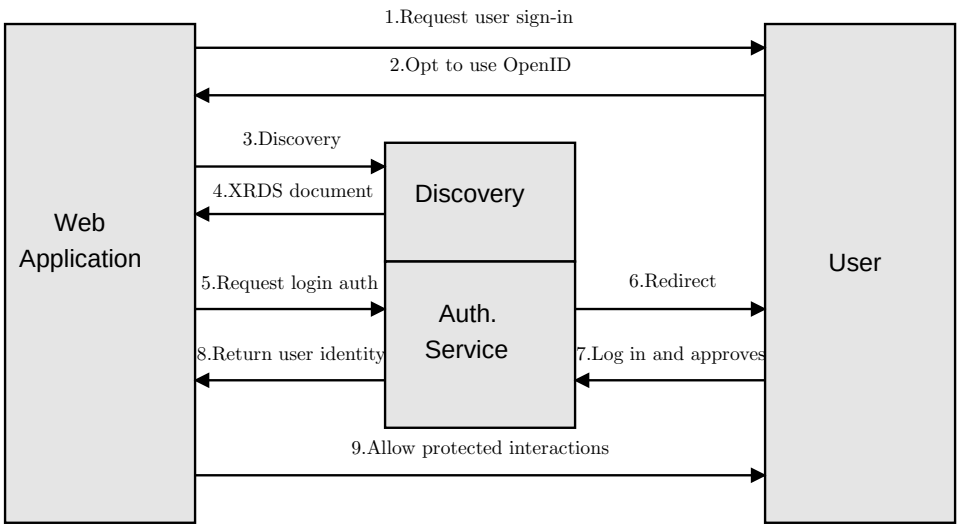

Figure 1: OpenID login authentication for web applications.

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

$$\sigma_{\text{DS}} = \texttt{disc\_req}@x.\overline{\texttt{disc}_{\text{AS}}}@x.\mathbf{1}$$

$$\sigma_{\text{AS}} = \texttt{req}(x)@\text{WA}.\overline{\texttt{redirect}}@x.\texttt{get\_account}@x.\\ (\overline{\texttt{ok}}@x.\overline{\texttt{token}}@\text{WA}.\mathbf{1} \lfloor_{\emptyset} \oplus {}_{\emptyset}\rfloor \overline{\texttt{fail}}@x.\overline{\texttt{no\_token}}@\text{WA}.\mathbf{1})$$

Table 8: Contracts executed by the Authentication Service AS and the Discovery Service DS

6.2. Formal model

We define a formal model of the federated authentication system based on OpenID, in which we disregard error management and other low-level implementation details. We presuppose a security lattice with four security levels: H, L, h_1, h_2

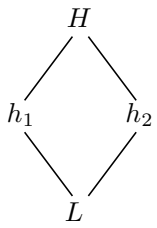

Figure 2: Lattice $\langle \Sigma, \preceq \rangle$ used in the Example of Section 6.

(cf. Figure 2), assigned according to the WA confidentiality requirements. Before authenticating, users are anonymous, and associated with the lowest security level L; once authenticated, their security level is upgraded to h_1 and h_2, depending on autonomous decision on the web application side. The web application itself is at the highest security level: H.

We first give the definition of the contracts executed by the discovery and authentication services, i.e., σ_{DS} and σ_{AS}, respectively (cf. Table 8). The discovery service σ_{DS} simply returns a WA endpoint ($\texttt{disc}_{\text{AS}}$) in response to the `disc_req` request. The AS contract, in turn, receives a request to authenticate a user, $\texttt{req}(x)$, redirects user x to the login service and waits for the account information (`account`). Based on the credentials received, AS accepts or rejects x, and notifies x and WA accordingly.

Table 9 formalises the definition of contract σ_{WA} performed by principal WA, as described in Figure 1. The first line in the definition of σ_{WA} corresponds to steps 1 and 2 of the authentication procedure: the client principal C decides between an internal authentication procedure or one based on OpenID. In the first case, WA accepts the credentials for the account directly, (`account`) and based on the

$$
\begin{aligned}
\sigma_{\text{WA}} = \;& \texttt{conf\_req}@x.\overline{\texttt{choices}}@x.(\texttt{internal}@x.\sigma_{aut} + \texttt{open\_id}@x.\sigma_{oid}) \\
\sigma_{aut} = \;& \texttt{account}@x.(\overline{\texttt{no\_auth}}@x.\sigma_L \lfloor \emptyset \oplus \emptyset \rfloor \overline{\texttt{auth}}@x.(\sigma_{h_1} \lfloor_{(\text{WA},x):h_1 \,\oplus\, (\text{WA},x):h_2} \rfloor \sigma_{h_2})) \\
\sigma_{oid} = \;& \overline{\texttt{disc\_req}}@\text{DS}.\texttt{disc}_{\text{AS}}@\text{DS}.\overline{\texttt{req}}(x)@\text{AS}. \\
& \qquad (\texttt{no\_token}@\text{AS}.\sigma_L + \texttt{token}@\text{AS}.(\sigma_{h_1} \lfloor_{(\text{WA},x):h_1 \,\oplus\, (\text{WA},x):h_2} \rfloor \sigma_{h_2})) \\
\sigma_{h_1} = \;& \texttt{op}_{h_1}@x.\overline{\texttt{reply}_{h_1}}@x.\mathbf{1} + \texttt{op}_{h_2}@x.\overline{\texttt{deny}_{h_2}}@x.\mathbf{1} + \texttt{op}_L@x.\overline{\texttt{reply}_L}@x.\mathbf{1} \\
\sigma_{h_2} = \;& \texttt{op}_{h_1}@x.\overline{\texttt{deny}_{h_1}}@x.\mathbf{1} + \texttt{op}_{h_2}@x.\overline{\texttt{reply}_{h_2}}@x.\mathbf{1} + \texttt{op}_L@x.\overline{\texttt{reply}_L}@x.\mathbf{1} \\
\sigma_L = \;& \texttt{op}_{h_1}@x.\overline{\texttt{deny}_{h_1}}@x.\mathbf{1} + \texttt{op}_{h_2}@x.\overline{\texttt{deny}_{h_2}}@x.\mathbf{1} + \texttt{op}_L@x.\overline{\texttt{reply}_L}@x.\mathbf{1}
\end{aligned}
$$

Table 9: Contract executed by the Web Application WA.

$$
\begin{aligned}
\sigma_{\text{C}} = \;& \overline{\texttt{conf\_req}}@\text{WA}.\texttt{choices}@\text{WA}.(\overline{\texttt{internal}}@\text{WA}.\sigma_{ia} \lfloor \emptyset \oplus \emptyset \rfloor \overline{\texttt{open\_id}}@x.\sigma_{oid}) \\
\sigma_{ia} = \;& \overline{\texttt{account}}@\text{WA}.(\texttt{auth}@\text{WA}.\sigma_{Act} + \texttt{no\_auth}@x.\overline{\texttt{op}}_L@x.\texttt{reply}_L@x.\mathbf{1}) \\
\sigma_{oid} = \;& \texttt{redirect}@z.\overline{\texttt{get\_account}}@z.(\texttt{ok}@z.\sigma_{act} + \texttt{fail}@z.\overline{\texttt{op}}_L@x.\texttt{reply}_L@x.\mathbf{1}) \\
\sigma_{act} = \;& \overline{\texttt{op}}_L@\text{WA}.\texttt{reply}_L@\text{WA}.\mathbf{1} \lfloor \emptyset \oplus \emptyset \rfloor (\overline{\texttt{op}}_{h_2}@\text{WA}.(\texttt{reply}_{h_2}@\text{WA}.\mathbf{1} + \texttt{deny}_{h_2}@\text{WA}.\mathbf{1}) \\
& \qquad \lfloor \emptyset \oplus \emptyset \rfloor \overline{\texttt{op}}_{h_1}@\text{WA}.(\texttt{reply}_{h_1}@\text{WA}.\mathbf{1} + \texttt{deny}_{h_1}@\text{WA}.\mathbf{1})))
\end{aligned}
$$

Table 10: Contract executed by the User principal C.

credential received, it decides whether to reject the request or to accept. If accepted, the user is granted access at level h_1 or h_2, and provided a service according to a corresponding contract, σ_{h_1} or σ_{h_2} respectively. If rejected, instead, the user is only guaranteed a basic service σ_L corresponding to its low security level. Observe that upon a successful authentication, the user security level is upgraded by the internal choice operator.

Table 10 completes the description of the system with the definition of the client contract σ_C. Initially, 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 $\overline{\texttt{get\_account}}$) 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 $\texttt{op}_i$, with $i = L, h_1, h_2$.

Compliance and Security Analysis. An analysis of the complete system

$$SYS \stackrel{def}{=} \text{WA}[\sigma_{\text{WA}}] \parallel \text{C}[\sigma_{\text{C}}] \parallel \text{DS}[\sigma_{\text{DS}}] \parallel \text{AS}[\sigma_{\text{AS}}]$$

shows that SYS makes a compliant composition. In particular, we observe that the WA component that explicitly denies access is crucial for compliance. In fact, omitting the `deny` responses from the definition of WA (in that case WA would simply disregard requests from unauthorised users), makes SYS non-compliant. In fact, failing to respond to those requests, would cause a deadlock for users trying to access a service for which they have not been granted access. This corresponds to what the well-known software engineering practice stating that all user requests should be handled explicitly (even when handling a request is implemented simply as rejecting it).

Letting $\Gamma = \{\text{WA} : H, \text{C} : L, \text{DS} : L, \text{AS} : L\}$, we can show that $SYS \in \mathcal{NI}_{\Gamma,L}$ 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.

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

Web services are vulnerable to attacks that can defeat the security measures designed by the developer. We continue with our analysis by considering the possibility that a malicious use sends to WA an ill-formed request that causes a failure in the activated service. In practice, this may correspond to an ill-formed XML document or to a SQL injection attack. A principled implementation would require that, as a consequence of a failure, WA leaks 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., [31]). We show how our framework is able to formally detect whether a WA is designed according to this security principle.

The revised version of the model is reported in Table 11. We omit the authentication and discovery services (σ_{AS} and σ_{DS}) as identical to those of Table 8.

In the revised 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. Contract σ_{X} executed by the attacker principal X tries to send to WA the malicious request by synchronising on `ill_msg`, and if this happens then it catches one of the possible error messages returned. Observe that, before an operation request $\texttt{op}_i$, $i = L, h_1, h_2$, 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.

$$SYS_{\mathrm{X}} \stackrel{def}{=} \mathrm{AS}[\sigma_{\mathrm{AS}}] \parallel \mathrm{C}[\sigma_{\mathrm{C}}] \parallel \mathrm{X}[\sigma_{\mathrm{X}}] \parallel \mathrm{DS}[\sigma_{\mathrm{DS}}] \parallel \mathrm{WA}[\sigma_{\mathrm{WA}}]$$

$$\sigma_{\mathrm{WA}} = \mathtt{conf\_req}@x.\overline{\mathtt{choices}}@x.(\mathtt{internal}@x.\sigma_{aut} + \mathtt{open\_id}@x.\sigma_{oid})$$

$$\sigma_{aut} = \mathtt{account}@x.(\overline{\mathtt{no\_auth}}@x.\sigma_L \lfloor \emptyset \oplus \emptyset \rfloor \overline{\mathtt{auth}}@x.(\sigma_{h_1} \lfloor {}_{(\mathrm{WA},x):h_1} \oplus {}_{(\mathrm{WA},x):h_2} \rfloor \sigma_{h_2}))$$

$$\sigma_{oid} = \overline{\mathtt{disc\_req}}@\mathrm{DS}.\mathtt{disc}_{\mathrm{AS}}@\mathrm{DS}.\overline{\mathtt{req}}(x)@\mathrm{AS}.\ \mathtt{no\_token}@\mathrm{AS}.\sigma_L + \mathtt{token}@\mathrm{AS}.(\sigma_{h_1} \lfloor {}_{(\mathrm{WA},x):h_1} \oplus {}_{(\mathrm{WA},x):h_2} \rfloor \sigma_{h_2})$$

$$\sigma_{h_1} = \mathtt{op}_{h_1}@x.(\overline{\mathtt{reply}_{h_1}}@x.\mathbf{1} + \sigma^{\dagger}_{h_1}) + \mathtt{op}_{h_2}@x.\overline{\mathtt{deny}_{h_2}}@x.\mathbf{1} + \mathtt{op}_L@x.(\overline{\mathtt{reply}_L}@x.\mathbf{1} + \sigma^{\dagger}_L) + \sigma^{\dagger}_0$$

$$\sigma_{h_2} = \mathtt{op}_{h_1}@x.\overline{\mathtt{deny}}_{h_1}@x.\mathbf{1} + \mathtt{op}_{h_2}@x.\overline{\mathtt{reply}}_{h_2}@x.\mathbf{1} + \sigma^{\dagger}_{h_2}) + \mathtt{op}_L@x.(\overline{\mathtt{reply}}_L@x.\mathbf{1} + \sigma^{\dagger}_L) + \sigma^{\dagger}_0$$

$$\sigma_L = \mathtt{op}_L@x.\overline{\mathtt{reply}}_L@x.\mathbf{1} + \sigma^{\dagger}_0$$

$$\sigma^{\dagger}_i = \mathtt{ill\_msg}@y.\overline{\mathtt{msg}}_i@y.\overline{\mathtt{close}}@x.\mathbf{1}$$

$$\sigma_{\mathrm{C}} = \overline{\mathtt{conf\_req}}@\mathrm{WA}.\mathtt{choices}@\mathrm{WA}.(\overline{\mathtt{internal}}@\mathrm{WA}.\sigma_{ia} \lfloor \emptyset \oplus \emptyset \rfloor \overline{\mathtt{open\_id}}@x.\sigma_{oid})$$

$$\sigma_{ia} = \overline{\mathtt{account}}@\mathrm{WA}.(\mathtt{auth}@\mathrm{WA}.\sigma_{act} + \mathtt{no\_auth}@x.\overline{\mathtt{op}}_L@x.\mathtt{reply}_L@x.\mathbf{1})$$

$$\sigma_{oid} = \mathtt{redirect}@z.\overline{\mathtt{get\_account}}@z.\mathtt{ok}@z.\sigma_{act} + \mathtt{fail}@z.\overline{\mathtt{op}}_L@x.\mathtt{reply}_L@x.\mathbf{1})$$

$$\sigma_{act} = (\overline{\mathtt{op}}_L@\mathrm{WA}.(\mathtt{reply}_L@\mathrm{WA}.\mathbf{1} + \mathtt{close}@\mathrm{WA}.\mathbf{1}) \lfloor \emptyset \oplus \emptyset \rfloor (\overline{\mathtt{op}}_{h_2}@\mathrm{WA}.\ (\mathtt{reply}_{h_2}@\mathrm{WA}.\mathbf{1} + \mathtt{deny}_{h_2}@\mathrm{WA}.\mathbf{1} + \mathtt{close}@\mathrm{WA}.\mathbf{1}) \lfloor \emptyset \oplus \emptyset \rfloor \overline{\mathtt{op}}_{h_1}@\mathrm{WA}.\ (\mathtt{reply}_{h_1}.@\mathrm{WA}.\mathbf{1} + \mathtt{deny}_{h_1}@\mathrm{WA}.\mathbf{1} + \mathtt{close}@\mathrm{WA}.\mathbf{1}))) + \mathtt{close}@\mathrm{WA}.\mathbf{1}$$

$$\sigma_{\mathrm{X}} = \overline{\mathtt{ill\_msg}}@\mathrm{WA}.(\mathtt{msg}_0@\mathrm{WA}.\mathbf{1} + \mathtt{msg}_{h_1}@\mathrm{WA}.\mathbf{1} + \mathtt{msg}_{h_2}@\mathrm{WA}.\mathbf{1} + \mathtt{msg}_L@\mathrm{WA}.\mathbf{1}) + \mathbf{1}$$

Table 11: Insecure contracts for WA and C.

Analysis. We show that modified system SYS_{X}, as reported in Table 11 is compliant but does not satisfy non-interference at security level L. Compliance follows directly from the observation that upon an `ill_msg` 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, the user contract always considers the possibility of a service abort by synchronising on `close`. To prove that $SYS_{\mathrm{X}} \notin \mathcal{NI}_{\Gamma,L}$ it suffices to show that when X synchronises on $\mathtt{msg}_1$ this is surely preceded in SYS_{X} by a synchronisation on $\mathtt{op}_{h_1}$ of level h_1" hence the attacker can infer the profile of the user being logged and that it required an operation $\mathtt{op}_{h_1}$ (the same holds for $\mathtt{op}_{h_2}$). This vulnerability is well-known to practitioners, and is usually caused by un-handled exceptions. Conversely, as we argued, exceptions should be always handled, and any principal causing them should be notified with an error message independent of the server status [31]. Indeed, if we assume $\mathtt{msg}_i = \mathtt{msg}$, it is possible to automatically prove that $SYS_{\mathrm{X}} \in \mathcal{NI}_{\Gamma,L}$.

It is instructive to notice that the fix is not always as direct as in the case we just examined. Consider for instance a further version of SYS_{X} with $\mathtt{msg}_i = \mathtt{msg}$ for $i = 0, L, h_1, h_2$ and in which we assume that ill-formed messages may only be received after any synchronisation on $\mathtt{op}_i$. At first, the new assumption would seem to corresponds to a stronger security protection over SYS_{X}. On the other hand, it turns out that the new system does not satisfy non-interference at L, because an observer can infer that a high security level synchronisation occurred by observing how the error is handled. In fact, if a `close` synchronisation occurs at L, then we know that neither of the steps $\mathtt{op}_{h_1}$ and $\mathtt{op}_{h_2}$ occurred, for these correspond to a synchronisation at level h_1 or h_2. This is enough to break non-interference.

7. Related work

The problem of modelling web service orchestrations and choreographies with formal languages has received considerable attention in the literature, and a substantial body of work has been directed towards the development of algorithmic analysis of SOAs and their properties (see [32] for an updated taxonomy). Verification of web service orchestrations has indeed become a central aspect of the software architecture design (see, e.g., [33, 34]).

In [35, 36, 37] the authors propose a methodology that translates a choreography described with the Business Process Execution Language for Web Services (BPEL) into Petri nets. Then static analysis on Petri nets joint with events log mining are applied to verify that the system behavior respects the specifications. Petri nets are used in [38] to give a method for web service discovery, modelling and composition. A different formalism is proposed in [39] where the authors build their analysis of web service choreographies on symbolic transition systems

(STS). The main advantage of using STSs is that while computations and message exchanges are modelled, the symbolic executions contain the state space explosion. In [40] automata are used to infer a model of the behavior of a web service given its WSDL description.

Fu et al. [41, 42] address the problem of verifying the correctness of web service interactions by means of a detailed model which is then translated into a language that can be analysed by the SPIN model checker. Similarly, in [43] the authors propose an encoding of collaboration diagrams into the LOTOS process algebra. As a consequence, automate verifications and service peer generations are allowed.

More closely related to our present approach is the work of Brogi et al. in [44] where CCS is used to formalize Web Service Choreography Interface (WSCI). In [45, 46] a graph based analysis is proposed to check if the cooperation of a set of software components is deadlock or failure free. In [47] the authors use the language Communicating Sequential Processes (CSP) to give a formal semantics of components' interactions. Then they develop a theory to analyse if a system admits deadlock or livelock. The approach we take to test the compliance of a choreographies is inspired by [22]. With respect to all these works, we introduce the use of filters as prescriptions of behaviors and the language we propose is enriched with a type system that allows us to reason about the security properties of a web service choreography. Therefore, the strength of out approach is providing a unique framework to carry out liveness and security analysis.

As to security, [48] introduces the a value-passing process calculus to model and reason about the security of web service transactions. Though related for the context in which security is approached (web services), the technical development is very different: their logic is specifically targeted at predicating on the values exchanged and their relationships, and consequently, the properties of integrity and confidentiality are characterized in terms of direct relationships (and explicit flows) among program variables. Similarly, coordination properties of service compositions are expressed in terms of constraints over shared references, and the verification problem reduces to the dynamic solution of the associated constraint problem.

In [49] the authors introduce a set of process algebraic operators to mimic the behavior of security automata, acting *program controllers* to enforce security policies expressed as formulas in the modal μ-calculus, and devise an automatic algorithm to synthesize controllers implementing the security policy of interest. Technically, the approach shares several ideas with our framework, though it has largely complementary targets. Instead of characterizing specific security operators, we provide a characteristic formula for non-interference, and enforce a rather general information-flow security policy. Furthermore, though our filters may be understood as (truncation) controllers, we employ them as a tool to enforce security

and compliance in a uniform setting.

A further paper strongly related to ours is by Nakajima [50] 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 [50] is built upon a simple lattice-based model for security labels. Instead, our approach admits more flexible security policies which can be dynamically specified by the service participants. Similarly, [50] considers safety properties such as deadlock freedom and specific progress properties, while model instead deals also with the property of livelock freedom.

Our use of filters as prescription of compliant and secure behaviours is inspired by [7, 24]; with respect to these work we include in the filter definition the non-interference property that is developed on a dynamic typing system. In [51] the authors introduce an algorithm based on model checking that derive the less strict conditions under which a property is satisfied by labelled transition system. In [52] model checking is exploited for ensuring the safety of some component, i.e., that its internal invariant are never violated by other components' method calls. The authors introduce an algorithm to compute an interface (represented by a marked graph) that allows only the correct interactions among the system's components.

8. Conclusion

We have developed a formal model for the analysis of multilevel security and transactional correctness in service contract compositions. The model is based on a calculus to represent service contracts and their compositions, and on a characterization of the properties of interest in terms of modal formulae of the μ-calculus. We also devise an algorithm for adaptable service compositions based on the automatic synthesis of a filtering mechanism to prune the execution graph of such compositions so as to enforce the desired security and correctness invariants. we prove the the filtering mechanism synthesized is the best (most permissive) possible one. Collectively, our results represent the first attempt to provide a uniform, and effective, model-checking framework for two central properties of security and functional correctness in web service compositions and more generally in distributed systems.

References

[1] D. Booth, C. K. Liu, Web services description language (WSDL), `http://www.w3.org/TR/wsdl20-primer/`.

[2] Web Services Choreography Working Group, Web Services Choreography Description Language, http://www.w3.org/2002/ws/chor/.

[3] A. A., A. Arkin, S. Askary, C. Barreto, et al., Web Services Business Process Execution Language version 2.0., OASIS Standard, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[4] C. Kaler, e. a. A. Nadalin, Web Services Trust Language (WS-Trust), http://msdn.microsoft.com/ws/2004/04/ws-trust (2004).

[5] C. Kaler, e. a. A. Nadalin, Web services secure conversation language (ws-secureconversation), http://msdn.microsoft.com/ws/2004/04/ws-secure-conversation/ (2004).

[6] 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 Lect. Notes on Comp. Sci., Springer, 2006, p. 148162.

[7] 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.

[8] 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.

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

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

[11] 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 Lect. Notes on Comp. Sci., Springer, 2007, pp. 34–50.

[12] 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.

[13] 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.

[14] 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.

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

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

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

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

[19] OpenId Foundation, OpenId authentication, http://openid.net/specs/openid-authentication-2_0.html.

[20] S. Rossi, Model checking adaptive multilevel service compositions, in: Formal Aspects of Component Software - 7th International Workshop, FACS 2010, Guimarães, Portugal, October 14-16, 2010, Revised Selected Papers, Vol. 6921 of Lecture Notes in Computer Science, Springer, 2010, pp. 106–124.

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

[22] 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 Lect. Notes on Comp. Sci., Springer, 2008, pp. 37–54.

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

[24] 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.

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

[26] D. Sangiorgi, On the origins of bisimulation and coinduction, ACM Transactions on Programming Languages and Systems (TOPLAS) 31.

[27] M. Müller-Olm, Derivation of Characteristic Formulae, Elect. Notes in Theor. Comp. Sci. 18 (1998) 159–170.

[28] 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 Lect. Notes on Comp. Sci., Springer, 1995, pp. 72–88.

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

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

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

[32] K. S. Chan, J. Bishop, J. Steyn, L. Baresi, S. Guinea, A fault taxonomy for web service composition, in: Service-Oriented Computing - ICSOC 2007 Workshops, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 363–375.

[33] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, P. Spoletini, Validation of web service compositions, Software, IET 1 (6) (2007) 219 –232.

[34] F. Yu, C. Wang, A. Gupta, T. Bultan, Modular verification of web services using efficient symbolic encoding and summarization, in: Proc. of the 16th ACM SIGSOFT Int. Symp. on Foundations of software engineering, ACM, New York, NY, USA, 2008, pp. 192–202.

[35] C. Ouyang, E. Verbeek, W. van der Aalst, S. Breutel, M. Dumas, A. ter Hofstede, Formal Semantics and Analysis of Control Flows in WS-BPEL, Science of Computer Programming 67 (2-3) (2005) 162–198.

[36] N. Lohmann, P. Massuthe, C. Stahl, D. Weinberg, Analysing interacting BPEL processes, in: Business Process Management, Vol. 4102, 2006, pp. 17–32.

[37] W. van der Aalst, M. Dumas, C. Ouyang, Conformance Checking of Service Behavior, ACM Trans. on Internet Technology 8 (3) (2008) 1–30.

[38] A. Brogi, S. Corfini, Behaviour-aware discovery of web service compositions, Int. J. Web Service Res. 4 (3) (2007) 1–25.

[39] L. Bentakouk, P. Poizat, F. Zaïdi, A Formal Framework for Service Orchestration Testing based on Symbolic Transition Systems, in: Proc. of Int. Conf. on Testing Communicating Systems (TESTCOM), 2009, pp. 16–32.

[40] A. Bertolino, P. Inverardi, P. Pelliccione, M. Tivoli, Automatic Synthesis of Behavior Protocols for Composable Web-Services, in: Proc. of the 7th joint meeting of ESEC and FSE, 2009, pp. 141–150.

[41] X. Fu, T. Bultan, J. Su, Analysis of interacting BPEL web services, in: ProC. of the 13th Int. Conf. on World Wide Web, ACM, New York, NY, USA, 2004, pp. 621–630.

[42] Z. Xiangpeng, Y. Hongli, Q. Zongyan, Towards the formal model and verification of web service choreography description language, in: Web Services and Formal Methods, Vol. 4184 of Lect. Notes in Comp. Sci., Springer Berlin Heidelberg, 2006, pp. 273–287.

[43] G. Salaün, T. Bultan, N. Roohi, Realizability of choreographies using process algebra encodings, IEEE Trans. on Services Computing 5 (3) (2012) 290–304.

[44] A. Brogi, C. Canal, E. Pimentel, A. Vallecillo, Formalizing web service choreographies, Elect. Notes in Theor. Comp. Sci. 105 (2004) 73–94.

[45] P. Inverardi, S. Uchitel, Proving deadlock freedom in component-based programming, in: Fundamental Approaches to Software Engineering, Vol. 2029 of Lect. Notes in Comp. Sci., Springer Berlin Heidelberg, 2001, pp. 60–75.

[46] M. Tivoli, P. Inverardi, Failure-free coordinators synthesis for component-based architectures, Science of Computer Programming 71 (2008) 181–212.

[47] R. Allen, D. Garlan, A formal basis for architectural connection, ACM Trans. Softw. Eng. Methodol. 6 (3) (1997) 213–249.

[48] A. Bracciali, A. Brogi, G. Ferrari, E. Tuosto, Security and Dynamic Compositions of Open Systems, in: Proc. of the Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA), CSREA Press, 2002, pp. 1372–1377.

[49] F. Martinelli, I. Matteucci, Through modeling to synthesis of security automata, Elect. Notes Theor. Comput. Sci. 179 (2007) 31–46.

[50] 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 Lect. Notes on Comp. Sci., Springer, 2004, pp. 488–501.

[51] D. Giannakopoulou, C. S. Pasareanu, H. Barringer, Assumption generation for software component verification, in: Proc. of 17th IEEE Int. Conf. on Automated Soft. Eng., 2002 (ASE 2002), 2002, pp. 3–12.

[52] T. A. Henzinger, R. Jhala, R. Majumdar, Permissive interfaces, SIGSOFT Softw. Eng. Notes 30 (5) (2005) 31–40.