Model Checking Adaptive Multilevel Service Compositions*

Sabina Rossi

Dipartimento di Informatica, Università Ca' Foscari, Venezia (Italy)
srossi@dsi.unive.it

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

1 Introduction

A Service Oriented Architecture (SOA) provides a software architectural style to integrate loosely specified and coupled services that communicate with each other. Simple Object Access Protocol (SOAP)-based Web services are becoming the most common implementation of SOA. They are designed to support interoperable service-to-service interactions over a network. In such a context of heterogeneous systems where each application is leveraging the services of other applications, a service-oriented analysis and design process plays a significant role.

In this paper we develop a method for verifying both security and correctness properties of multilevel service compositions based on the use of model-checking techniques [9]. We specify service compositions in terms of behavioral contracts [5, 6, 8, 16] which provide abstract descriptions of system behaviors by means of terms of a process algebra. Multi-party service compositions are modeled as the parallel composition of contracts. We define modal μ-calculus [15] formulae, interpreted over service configurations, characterizing those compositions which satisfy a non-interference property and are compliant, i.e., are both deadlock and livelock free. A model checker (like, e.g., NCSU Concurrency Workbench) can then be used to simultaneously check non-interference and compliance.

The notion of *non-interference* [12, 14, 22] we consider here is based on an information flow security model with dynamic policies. It demands that public synchronizations are unchanged as confidential communications are varied. Security policies are used to specify the security requirements of service components. In order to capture the dynamic nature, heterogeneity and lack of knowledge which are intrinsic features

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

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

Table 1. Syntax

of modern web services, we allow policies to be dynamically specified by the service participants. In our model, for example, customers may formulate their security requirements by dynamically assign types (that are security annotations) to individual service components. We also consider the property of *compliance* which is widely used in the context of SOA as a formal device to identify well-formed service compositions, those whose interactions are free of synchronization errors.

Finally, we develop an algorithm for verifying adaptive service compositions. This is based on the use of filters, introduced in [7], as prescriptions of behaviour (coercions to prevent service misbehaviour). Security and correctness properties for adaptive multilevel service compositions are ensured by the automatic synthesis of an adapting filter.

Plan of the paper. Section 2 introduces the calculus for multilevel service compositions. Section 3 formalizes the notion of non-interference. Section 4 presents characteristic modal μ-calculus formulae for non-interference. Section 5 introduces the notion of compliance and defines a modal μ-calculus formula characterizing it. Section 6 presents an algorithm for adaptive service compositions. Finally, Section 7 concludes the paper.

2 The Calculus

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

In order to specify multilevel service compositions, we assign security levels to principal identities and express both contracts and compositions as typed terms of our calculus. Formally, we assume a complete lattice $\langle \Sigma, \preceq \rangle$ of security annotations, ranged over by ς, ϱ, where $\top$ and $\bot$ represent the top and the bottom elements of the lattice. We denote by $\sqcup$ and $\sqcap$, respectively, the join and meet operators over Σ. Type environments are used to assign security levels to principals. A type environments Γ is a finite mapping from principals and variables to security annotations. We define $\Gamma_1 \sqcup \Gamma_2$ (resp., $\Gamma_1 \sqcap \Gamma_2$) the type environment Γ such that $\Gamma(p) = \Gamma_i(p)$ if $p \notin dom(\Gamma_1) \cap dom(\Gamma_2)$ and $p \in dom(\Gamma_i)$, and $\Gamma(p) = \Gamma_1(p) \sqcup \Gamma_2(p)$ (resp., $\Gamma_1(p) \sqcap \Gamma_2(p)$) otherwise.

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

$$\sigma_C = \overline{\texttt{Req}}@T.\texttt{Lst}@T.(\,\overline{\texttt{Close}}@T.\mathbf{1}\lfloor_{\emptyset \oplus\ T:\mathtt{H}}\rfloor(\,\overline{\texttt{Buy1}}@T.\overline{\texttt{Pay}}@T.\ \texttt{Get}@A_1.\mathbf{1}\lfloor_{A_1:\mathtt{H} \oplus\ A_2:\mathtt{H}}\rfloor\overline{\texttt{Buy2}}@T.\overline{\texttt{Pay}}@T.\texttt{Get}@A_2.\mathbf{1}\,)\,)$$

$$\sigma_T = \texttt{Req}@x.\overline{\texttt{Inq}}@A_1.\overline{\texttt{Inq}}@A_2.\texttt{Price}@A_1.\texttt{Price}@A_2.\overline{\texttt{Lst}}@x.(\,\texttt{Close}@x.\mathbf{1} + \texttt{Buy1}@x.\overline{\texttt{Ord}}@A_1.\texttt{Pay}@x.\overline{\texttt{Conf}}@A_1.\mathbf{1} + \texttt{Buy2}@x.\overline{\texttt{Ord}}@A_2.\texttt{Pay}@x.\overline{\texttt{Conf}}@A_2.\mathbf{1})$$

$$\sigma_A = \texttt{Inq}@x.\overline{\texttt{Price}}@x.(\,\texttt{Ord}@x.\texttt{Conf}@x.\overline{\texttt{Get}}@y.\mathbf{1} + \mathbf{1}\,)$$

Table 2. Example of a travel agency

Syntax. The syntax of our calculus is presented in Table 1. Term $\mathbf{1}$ indicates a contract that has reached a successful state. The contract $\bar{a}@p.\sigma$ describes a service that sends a message on a to principal p and then behaves as σ; syntactically, the principal identity p may be a variable, but it must be a name when the prefix is ready to fire. Dually, the input prefix $a@u.\sigma$ waits for an input on a from a particular/any principal and then continues as σ. If u is a variable x, then the input form is a binder for x with scope σ: upon synchronization with a principal p, x gets uniformly substituted by p in σ. The contract $\sigma + \sigma'$ denotes an external choice, guided by the environment. The contract $\sigma\lfloor_{\Gamma \oplus\ \Gamma'}\rfloor\sigma'$ represents the internal choice between σ in the type environment Γ and σ' in the type environment Γ' made irrespective of the structure of the interacting components; the internal choice operator we adopt in this paper allows us to model the fact that a principal may dynamically change (upgrade) the security level of his interactions with other service components through the specific type environment associated to each choice. Finally, $\texttt{rec}(\mathbf{x})\,\sigma$ makes it possible to express iteration in the contract language. As usual, we assume a standard contractivity condition for recursion, requiring that recursive variables be guarded by a prefix. Given a principal $p \in \mathcal{P}$, we say that a contract σ is *p-compatible* if for all $\bar{a}@q$ and $a@q$ occurring in σ, q is different from p. A composition $p_1[\sigma_1] \parallel \cdots \parallel p_n[\sigma_n]$ of principals must be *well-formed* [4] to constitute a legal composition, namely: (i) the principal identities p_i's must all be pairwise different, and (ii) each contract σ_i, executed by principal p_i, is p_i-compatible. If $C = p_1[\sigma_1] \parallel \cdots \parallel p_n[\sigma_n]$ is a legal composition, we say that C is a $\{p_1, \ldots, p_n\}$-composition (dually, that $\{p_1, \ldots, p_n\}$ are the underlying principals for C). Throughout, we assume that contracts are closed (they have no free variables) and that compositions are well formed. Also, we often omit trailing $\mathbf{1}$'s.

A service component 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. However, it is reasonable to assume that a service component cannot downgrade the security level of other principals; moreover it cannot upgrade the level of its interactions with other components above its own level. These are the only typing constraints we assume. Such a typing discipline ensures that information does not explicitly flow from high to low, but it does not deal with implicit flows. Instead, we characterize non-interference in terms of the actions that typed service compositions may perform.

$$\coprod(\mathbf{1}) = \emptyset \qquad \coprod(\mathbf{x}) = \emptyset \qquad \coprod(\varphi.\sigma) = \coprod(\sigma) \qquad \coprod(\sigma_1 + \sigma_2) = \coprod(\sigma_1) \sqcup \coprod(\sigma_2)$$

$$\coprod(\sigma_1 \lfloor_{\Gamma_1} \oplus_{\Gamma_2} \rfloor \sigma_2) = \Gamma_1 \sqcup \Gamma_2 \sqcup \coprod(\sigma_1) \sqcup \coprod(\sigma_2) \qquad \coprod(\mathtt{rec}(\mathbf{x})\,\sigma) = \coprod(\sigma)$$

$$\frac{}{\Gamma, p : \varsigma \vdash p[\sigma]}\ \varsigma \in \Sigma,\ \bigsqcup_{q \in dom(\coprod(\sigma))} \coprod(\sigma)(q) \preceq \varsigma \qquad \frac{\Gamma \vdash C_1 \quad \Gamma \vdash C_2}{\Gamma \vdash C_1 \parallel C_2}$$

Table 3. Type system

Example 1. Table 2 shows an example of a service contract composition. Four services are involved: $C[\sigma_C]$, $T[\sigma_T]$ and $A_i[\sigma_A]$ representing a *customer*, a *travel agency*, and two *airline companies*, respectively. The elementary actions represent business activities that result in messages being sent or received. For example, the action $\overline{\mathtt{Req}}@T$ undertaken by the customer results in a message being sent to the travel agency. The customer sends a request to the travel agency which then inquires the airlines to get the prices for the selected route. Each airline responds and the travel agency sends to the customer the list of the best prices. The customer decides whether to close the communication with the travel agency or to buy from one of the airlines. In the latter case the customer decides to assign a high security level (H) to both the travel agency and the chosen airline company in order to safeguard the confidentiality of the purchasing data. The travel agency orders the ticket from the selected airline and takes a deposit (or a full payment) from the customer. As soon as the airline receives the confirmation of the payment, the ticket is issued to the customer. □

Type System. The typing rules reported in Table 3 ensure that, given a service composition with an underling set π of principals, every $p \in \pi$ cannot upgrade the security level of the principals in π (including p) above the level of p itself. The judgments take the form $\Gamma \vdash C$, where Γ is a type environment and C is a service composition. We denote by $\coprod$ the function that associates to each contract σ the join of all the Γ_i occurring as a parameter of an internal choice in σ. We say that a service component $p[\sigma]$ is well-typed in Γ if p will never upgrade the security level of other principals over its own level; this is obtained by requiring that for all $q \in dom(\coprod(\sigma))$, it holds that $\coprod(\sigma)(q) \preceq \varsigma$ where ς is the security level of p.

Semantics. We define the dynamics of typed service compositions in terms of labelled transition systems (and a success predicate), with rules reported in Table 4. In the table, and in the whole paper, λ ranges over visible contract typed actions $\bar{a}@p$, $a@p$ and silent actions τ; δ ranges over service composition actions $a_{p \to q}$, $\bar{a}_{p \to q}$ and τ. We say that $\Gamma \rhd C$ is a *configuration* if Γ is a type environment and C is a $\{p_1, \ldots, p_n\}$-service composition such that $\{p_1, \ldots, p_n\} \subseteq dom(\Gamma)$.

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

Contract and composition satisfaction: $\sigma \checkmark$

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

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

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

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

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

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

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

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

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

Table 4. Typed contract and composition transitions

component cannot downgrade the security level of other principals. Each typed contract transition yields a corresponding transition for the principal hosting the contract. Transitions for configurations are relative to the underlying set $dom(\Gamma)$ of principals and are entirely standard.

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

Lemma 1 (Subject reduction). *Let Γ be a type environment and C be a service composition such that $\Gamma \rhd C$ is a configuration. If $\Gamma \rhd C$ is well-formed and $\Gamma \rhd C \xrightarrow{\tau} \Gamma' \rhd C'$, then $\Gamma' \rhd C'$ is well formed.* □

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

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

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

Table 5. Typed internal composition transitions

Example 2. Consider again the service composition S of Example 1. Let Σ contain two security annotations, $\mathtt{L}$ (*public*) and H (*confidential*), with $\mathtt{L} \preceq \mathtt{H}$. Let Γ be the type environment $C : \mathtt{H}$, $T : \mathtt{L}$, $A_1 : \mathtt{L}$, $A_2 : \mathtt{L}$. The service composition S is well-typed in Γ, i.e., $\Gamma \vdash S$. □

3 Non-Interference

The concept of *noninterference* [14] has been introduced to formalize the absence of information flow in multilevel systems. In the context of service compositions it demands that public interactions between service components are unchanged as secret communications are varied or, more generally, that the low level behaviour of the service composition is independent from the behaviour of its high components. In this way clients are assured that the data transmitted over the air to a web server remains confidential; in other words, sensitive data cannot be intercepted and understood by eavesdroppers.

The notion of non-interference we are going to introduce is relative to the internal behaviour of service compositions, i.e., we are interested in observing the synchronizations between service components. We thus refine the semantics of compositions in order to help (i) to distinguish a local contract move from a synchronization, and (ii) to identify the principals involved in every synchronization. This is captured by the rules collected in Table 5, where we use the relation $\hookrightarrow$ to represent typed synchronizations between service components. The τ label now indicates an internal action to a service component, while synchronizations between different peers in a composition are represented through a label of the form $\{a\}_{p\to q}$ meaning that principals p and q synchronize on action a. We let α range over the labels $\{a\}_{p\to q}$ and τ. We denote by $\overset{\tau}{\hookrightarrow\!\!\!\!\!\rightarrow}$ a possible empty sequence of $\overset{\tau}{\hookrightarrow}$ and define $\overset{\{a\}_{p\to q}}{\hookrightarrow\!\!\!\!\!\rightarrow} \overset{def}{=} \overset{\tau}{\hookrightarrow\!\!\!\!\!\rightarrow} \overset{\{a\}_{p\to q}}{\hookrightarrow} \overset{\tau}{\hookrightarrow\!\!\!\!\!\rightarrow}$.

The two semantics for service compositions, one expressed in terms of $\longrightarrow$ and the other one expressed in terms of $\hookrightarrow$, are related as follows.

Lemma 2. *Let $\Gamma \rhd C$ be a service configuration.*

- *$\Gamma \rhd C \overset{\tau}{\hookrightarrow} \Gamma' \rhd C'$ if and only if $C = C_1 \parallel p[\sigma] \parallel C_2$, $C' = C_1 \parallel p[\sigma'] \parallel C_2$ and $\sigma \overset{\tau}{\longrightarrow} \sigma'$;*

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

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

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

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

Behavioural Observations. We define behavioural observations in terms of equivalences that are parametric with respect to the security level $\varsigma \in \Sigma$ of the behaviour we want to observe. Such equivalences are relations over configurations that equate service compositions exhibiting the same ς-level component interactions. They are defined as a variant of the notion of *weak bisimulation* [18], an observation equivalence which allows one to observe the nondeterministic structure of the LTSs and focuses only on the observable actions.

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

Definition 1 (Weak bisimulation on ς-low actions). *Let $\varsigma \in \Sigma$. A* weak bisimulation on ς-low actions *is the largest symmetric relation $\approx_\varsigma$ over configurations such that whenever $\Gamma_1 \rhd C_1 \approx_\varsigma \Gamma_2 \rhd C_2$ with $\Gamma_1 =_\varsigma \Gamma_2$*

- *if $\Gamma_1 \rhd C_1 \overset{\alpha}{\hookrightarrow} \Gamma_1' \rhd C_1'$ with $\alpha = \tau$ or $\Gamma_1(\alpha) \preceq \varsigma$, then there exist Γ_2' and C_2' such that $\Gamma_2 \rhd C_2 \overset{\alpha}{\hookrightarrow\!\!\!\!\rightarrow} \Gamma_2' \rhd C_2'$ with $\Gamma_1' \rhd C_1' \approx_\varsigma \Gamma_2' \rhd C_2'$ and $\Gamma_1' =_\varsigma \Gamma_2'$;*
- *if $\Gamma_1 \rhd C_1 \overset{\alpha}{\hookrightarrow} \Gamma_1' \rhd C_1'$ with $\alpha \neq \tau$ and $\Gamma(\alpha) \not\preceq \varsigma$, then there exist Γ_2' and C_2' such that either $\Gamma_2 \rhd C_2 \overset{\alpha}{\hookrightarrow\!\!\!\!\rightarrow} \Gamma_2' \rhd C_2'$ or $\Gamma_2 \rhd C_2 \overset{\tau}{\hookrightarrow\!\!\!\!\rightarrow} \Gamma_2' \rhd C_2'$ with $\Gamma_1' \rhd C_1' \approx_\varsigma \Gamma_2' \rhd C_2'$ and $\Gamma_1' =_\varsigma \Gamma_2'$.*

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

The notion of non-interference is inspired by [13] and is expressed in terms of a restriction operator $\cdot|_\varsigma$ which allows one to represent a service composition prevented from performing internal synchronizations of a level higher than ς. The semantics of $C|_\varsigma$ is described by the following rule:

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

Definition 2 (Non-interference). *Let $\varsigma \in \Sigma$ and $\Gamma \rhd C$ 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*

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

□

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

$$\sigma_C = \overline{\texttt{Inq}}@F_1.\overline{\texttt{Inq}}@F_2.\texttt{Plan}@F_1.\texttt{Plan}@F_2.$$
$$(\,\overline{\texttt{Agree}}@F_1.\overline{\texttt{Close}}@F_2.\mathbf{1}\lfloor_{F_1:\mathtt{H}} \oplus_{F_2:\mathtt{H}}\rfloor\overline{\texttt{Agree}}@F_2.\overline{\texttt{Close}}@F_1.\mathbf{1}\,)\,)$$

$$\sigma_F = \texttt{Inq}@x.\overline{\texttt{LookUp}}@S.\texttt{Quote}@x.\overline{\texttt{Plan}}@C.(\,\texttt{Agree}@x.\mathbf{1} + \texttt{Close}@x.\mathbf{1}\,)$$

$$\sigma_S = \texttt{LookUp}@x.\overline{\texttt{Quote}}@x.\mathbf{1}$$

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

$$\sigma'_C = \overline{\texttt{Inq}}@F_1.\overline{\texttt{Inq}}@F_2.\texttt{Plan}@F_1.\texttt{Plan}@F_2.$$
$$(\,\overline{\texttt{Close}}@F_2.\overline{\texttt{Agree}}@F_1.\mathbf{1}\lfloor_{F_1:\mathtt{H}} \oplus_{F_2:\mathtt{H}}\rfloor\overline{\texttt{Close}}@F_1.\overline{\texttt{Agree}}@F_2.\mathbf{1}\,)\,)$$

Table 6. Example of a financial consulting platform

Example 3. Consider again the service composition S of Example 1 in the type environment Γ of Example 2. The property $S \in \mathcal{NI}_{\Gamma,\mathtt{L}}$ holds. □

Example 4. Consider the service composition depicted in Table 6: it consists of a *client* C, two *financial consulting services* F_1 and F_2 and a *stock quote service provider* S. The client inquires the financial services to get investment advices. The financial services consult the stock quote service provider in order to look up information on the financial quotes. Then the financial services send their investment recommendations to the client which may decide whether or not adhere to the investment plan proposed by one of the financial services and close the connection with the other one.

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

4 Modal Formulae for Non-Interference

In this section we present a method for verifying whether $\Gamma \models C \approx_\varsigma C|_\varsigma$ which consists in defining a modal μ-calculus formula $\phi^{\approx_\varsigma}(\Gamma \rhd C)$ such that $\Gamma' \rhd C'$ satisfies $\phi^{\approx_\varsigma}(\Gamma \rhd C)$ iff $\Gamma \rhd C \approx_\varsigma \Gamma' \rhd C'$. The proofs are in the spirit of [19].

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

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

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

Table 7. Semantics of modal mu-calculus

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

Modal μ-calculus formulae are interpreted over service configurations modelled by LTS's. Let $\Gamma \rhd C$ be a configuration and $LTS(\Gamma \rhd C) = (S_{\Gamma \rhd C}, Act, \longmapsto)$ where $S_{\Gamma \rhd C}$ is the set of states reachable from $\Gamma \rhd C$ through $\stackrel{\alpha}{\longmapsto}$ and $\alpha \in Act$ denotes the action $\{a\}_{p \to q}$ or τ. The subset of configurations in $S_{\Gamma \rhd C}$ that satisfy a formula ϕ, noted by $M_{\Gamma \rhd C}(\phi)(\rho)$, is inductively defined in Table 7. ρ is an *environment*, i.e., it is a partial mapping $\rho : Var \not\to 2^{S_{\Gamma \rhd C}}$ that interprets at least the free variables of ϕ by subsets of $S_{\Gamma \rhd C}$. For a set $x \subseteq S_{\Gamma \rhd C}$ and a variable X, we write $\rho[X \mapsto x]$ for the environment that maps X to x and $Y \neq X$ to $\rho(Y)$ if ρ is defined on Y.

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

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

$$Eqn : X_1 = \phi_1 \ \ldots \ X_n = \phi_n$$

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

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

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

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

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

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

$$Eqn_{\approx_\varsigma} : X_{\Gamma_1 \rhd C_1} = \phi^{\approx_\varsigma}_{\Gamma_1 \rhd C_1}, \ldots, X_{\Gamma_n \rhd C_n} = \phi^{\approx_\varsigma}_{\Gamma_n \rhd C_n}$$

consisting of one equation for each service configuration $\Gamma_1 \rhd C_1, \ldots, \Gamma_n \rhd C_n \in S_{\Gamma \rhd C}$. We define the formulae $\phi^{\approx_\varsigma}_{\Gamma_i \rhd C_i}$ such that the largest solution $M_{\Gamma \rhd C}(Eqn_{\approx_\varsigma})$ 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 bisimilar on ς-low actions 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 1 shows the exact form of such formulae. First we define:

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

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

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

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

Theorem 1. *Let $\phi^{\approx_\varsigma}_{\Gamma_i \rhd C_i}$ be the formula*

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

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

Characteristic formulae, i.e., *single* formulae characterizing configurations can be constructed by applying simple semantics-preserving transformation rules on equation systems as described in [19]. These rules are similar to the ones used by A. Mader in [17] as a mean of solving Boolean equation systems (with alternation) by Gauss elimination. Hence, since for any equation system Eqn there is a formula ϕ such that $Conf(Eqn) = Conf(\phi)$, we obtain that:

Theorem 2. *For any finite-state configuration $\Gamma \rhd C$ there is a modal μ-calculus formula $\phi^{\approx_\varsigma}(\Gamma \rhd C)$ such that $Conf(\phi^{\approx_\varsigma}(\Gamma \rhd C)) = \{\Gamma' \rhd C' \in S_{\Gamma \rhd C} \mid \Gamma' \rhd C' \approx_\varsigma (\Gamma' \rhd C')|_\varsigma\}$, that is the set of all the states reachable from $\Gamma \rhd C$ and satisfying $\mathcal{NI}_{\Gamma,\varsigma}$* □

5 A Modal Formula for Compliance

In this paper we refer to the notion of compliance for contract service compositions studied in [3]. 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 from the security levels of the principals involved in the component synchronizations. Therefore we omit trailing type environments in the definitions below, and write, e.g., $C \Longrightarrow C'$ to denote a transition of the form $\Gamma \rhd C \Longrightarrow \Gamma' \rhd C'$ for some type environments Γ and Γ'.

Definition 3 (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 synchronizations). This is enough to avoid both deadlocks and livelocks.

Example 5. Consider the service composition S of Example 1. It holds that $\Gamma \rhd S$ is both compliant, i.e., $S \downarrow$, and non interfering, i.e., $S \in \mathcal{NI}_{\Gamma,\mathbb{L}}$. □

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

Proposition 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 $C''\checkmark$.* □

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

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

where

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

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. The proof is given in [2].

Theorem 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$\}$.* □

Corollary 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 2 and 3 we have:

Corollary 2. *Let $\varsigma \in \Sigma$, $\Gamma \rhd C$ be a configuration and*

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

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

Using this method we can for instance exploit the model checker NCSU Concurrency Workbench (see [10]) to check whether both $C \in \mathcal{NI}_{\Gamma,\varsigma}$ and $C\downarrow$.

6 An Adaptive Algorithm

The model checking technique is based on the idea that the state transition graph of a finite-state system defines a Kripke structure, and efficient algorithms can be given for checking if the state graph defines a model of a given specification expressed in an appropriate temporal logic. In the explicit state approach the Kripke structure is

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

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

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

Table 8. Dynamics of Filtered contract service compositions

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 [3], 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 8.

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

Definition 4 (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 $\bullet \in \{\times, \otimes\}$:

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

We generalize the syntax of service compositions by allowing the term $\Gamma \rhd F(C)$ to account for the application of filters on the components of C. The dynamics of filtered service compositions derives directly by combining the transitions in Tables 4 and 8.

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

Definition 5 (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 $\delta \in \{a_{\_\to p}, \bar{a}_{p\to\_}\}$ and there exists $\Gamma \rhd C \in \mathcal{C}$ such that $\Gamma \rhd C \stackrel{\alpha}{\hookrightarrow\!\!\!\!\to}$ with $\alpha \in \{\{a\}_{\_\to p}, \{a\}_{p\to\_}\}$ and $\widehat{f} \propto_p \{\Gamma' \rhd C' \mid \Gamma \rhd C \stackrel{\alpha}{\hookrightarrow\!\!\!\!\to} \Gamma' \rhd C'\}$.*
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 $F \propto \{\Gamma \rhd C\}$. □

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

As discussed above, a global model checking algorithm applied to a configuration $\Gamma \rhd C$ and the modal formula $\Phi^{\varsigma}_{\Gamma \rhd C}$ calculates the set of states in the reduction graph (tracing the states reached by means of synchronizations or internal moves) of $\Gamma \rhd C$ satisfying $\Phi^{\varsigma}_{\Gamma \rhd C}$. This is the input of our algorithm. The reduction graph can be represented as a directed graph $G = (V, E)$ with labelled edges and vertices. The vertices in V represent the reachable states of $\Gamma \rhd C$. With each $\mathbf{v} \in V$ we associate two fields: $state[\mathbf{v}]$ gives the computation state (i.e., the derivative $\Gamma' \rhd C'$ of the initial state $\Gamma \rhd C$) associated with $\mathbf{v}$; $result[\mathbf{v}]$ is a tag SUCC or FAIL depending on whether the corresponding configuration satisfies $\Phi^{\varsigma}_{\Gamma \rhd C}$ or not as calculated by the model checker. An edge in E is a triple $(\mathbf{u}, \mathbf{v})_\alpha$ representing the transition $state[\mathbf{u}] \stackrel{\alpha}{\hookrightarrow\!\!\!\!\to} state[\mathbf{v}]$. Reduction graphs may be stored in a adjacency list representation, so that the set of

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
```

outgoing edges for each $\mathbf{u} \in V$ can be retrieved as $Adj[\mathbf{u}]$: thus $(\mathbf{u}, \mathbf{v})_\alpha \in E$ iff $(\alpha, \mathbf{v}) \in Adj[\mathbf{u}]$. We also write $Adj[\mathbf{u}, \alpha]$ for the set $\{\mathbf{v} \in V \mid (\mathbf{u}, \mathbf{v})_\alpha \in E\}$. Vertices with no outgoing edges are called leaves. We denote by $root[G]$ the vertex representing the initial state $\Gamma \rhd C$.

The first step consists in *re-labelling* the graph G calculated by the model-checker in such a way that the result label at each vertex $\mathbf{u}$ is set to FAIL if there exists at least one silent transition from $\mathbf{u}$ to a FAIL vertex; it is set to SUCC if either there are no silent transitions from $\mathbf{u}$ to a FAIL vertex and there exists a silent transition from $\mathbf{u}$ to a SUCC vertex or there exists one non-silent and non-conflicting transition from $\mathbf{u}$ to a SUCC vertex. The procedure iteratively examines all the vertices in the graph until it reaches a fixed point. This computation is accomplished by the `PushLabels` procedure and uses the following auxiliary definitions. Let $locs(\alpha)$ be $\{p, q\}$ in case $\alpha = \{a_{p \to q}\}$, and $\emptyset$ in case $\alpha = \tau$. Let $G = (V, E)$ be a reduction graph, and $\alpha = \{a_{p \to q}\}$.
- A path $\varpi = (\mathbf{u}, \mathbf{u}_1)_{\alpha_1}, \ldots, (\mathbf{u}_{n-1}, \mathbf{v})_{\alpha_n}$ from $\mathbf{u}$ to $\mathbf{v}$ in G is α-*free* if $locs(\alpha) \cap locs(\alpha_i) = \emptyset$ for all i's.
- A vertex $\mathbf{v}$ is a α-*free descendant* of $\mathbf{u}$ in G (dually, $\mathbf{u}$ is a α-*free ancestor* of $\mathbf{v}$) if there is a α-free path from $\mathbf{u}$ to $\mathbf{v}$.
- A vertex $\mathbf{u}$ *yields a conflict on* α if $\mathbf{u}$ has two distinct α-free descendants $\mathbf{v}_1$ and $\mathbf{v}_2$ such that $(\mathbf{v}_1, \mathbf{w}_1)_\alpha$ and $(\mathbf{v}_2, \mathbf{w}_2)_\alpha \in E$ and $result[\mathbf{w}_1] \neq result[\mathbf{w}_2]$.
- A vertex $\mathbf{v}$ has a conflict on α in G, noted $Conflict_G(\alpha, \mathbf{v})$ if $\mathbf{v}$ has a α-free ancestor yielding a conflict on α.

Intuitively, our algorithm will prune G by banning all the 'bad' synchronizations, and by preserving all the 'good' synchronizations that lead to nodes satisfying both non-interference and compliance. Due to the presence of internal choices, the same

Function `SuccessGraph(`G`)`

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

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

synchronization can look good at one point, but actually be bad. The definition of conflict formally captures this notion of ambiguous synchronizations.

Lemma 3. *After the call to* `PushLabels(`G`)`*, the following conditions hold for every node* $\mathbf{u}$ *in* G*:* (i) $\mathit{result}[\mathbf{u}] = \text{FAIL}$ *iff either there exists no* $(\mathbf{u}, \mathbf{v})_\alpha \in E$ *such that* $\mathit{result}[\mathbf{v}] = \text{SUCC}$ *and* $\neg \mathit{Conflict}_G(\alpha, \mathbf{u})$ *or there exists* $(\mathbf{u}, \mathbf{v})_\tau \in E$ *such that* $\mathit{result}[\mathbf{v}] = \text{FAIL}$*;* (ii) $\mathit{result}[\mathbf{u}] = \text{SUCC}$ *iff there exists no* $(\mathbf{u}, \mathbf{v})_\tau \in E$ *such that* $\mathit{result}[\mathbf{v}] = \text{FAIL}$ *and there exists either* $(\mathbf{u}, \mathbf{v})_\tau \in E$ *such that* $\mathit{result}[\mathbf{v}] = \text{SUCC}$ *or* $(\mathbf{u}, \mathbf{v})_\alpha \in E$ *with* $\alpha \neq \tau$, $\neg \mathit{Conflict}_G(\alpha, \mathbf{u})$ *and* $\mathit{result}[\mathbf{v}] = \text{SUCC}$. □

We say that a path ϖ in G is *successful* if $\mathit{result}[\mathbf{u}] = \text{SUCC}$ for every node $\mathbf{u}$ in ϖ, otherwise ϖ is *unsuccessful*. A node $\mathbf{u}$ is *root-successful* if it is reachable from $\mathit{root}[G]$ 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.

Lemma 4. *Let* $G' = (E', V')$ *be the graph generated by* `SuccessGraph(`G`)`*. Then* $\mathbf{u} \in V'$ *if and only if* $\mathbf{u}$ *is root-successful in* G. □

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

Theorem 4 (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 both* $F^{Alg}[\Phi^{\varsigma}_{\Gamma \rhd C}](C) \in \mathcal{NI}_{\Gamma,\varsigma}$ *and* $F^{Alg}[\Phi^{\varsigma}_{\Gamma \rhd C}](C)\downarrow$. *Also, if a filter* F *fixes* $\Gamma \rhd C$ *and is relevant for* $\Gamma \rhd C$*, then* $F \leq F^{Alg}[\Phi^{\varsigma}_{\Gamma \rhd C}]$. □

7 Conclusion

Some research efforts on model checking web services have already been proposed [1, 11, 21, 23]. The most related paper that we are aware of is by Nakajima [20] who introduces a lattice-based security labelling into BPEL in order to detect potential insecure

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

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 $state[\mathbf{u}] \checkmark$ **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 $\alpha = \{a_{p\rightarrow\_}\}$ **then**
      $F[p] := F[p] \times \bar{a}_{p\rightarrow\_}.F_\mathbf{v}[p]$;
    else if $\alpha = \{a_{\_\rightarrow p}\}$ **then**
      $F[p] := F[p] \times a_{\_\rightarrow p}.F_\mathbf{v}[p]$;
    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**
    $F[p] := \texttt{rec}(\mathbf{X}_\mathbf{u})\, F[p]$;
return F;

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 [20] is built upon a simple lattice-based model for security labels. Instead, our approach deals with more flexible security policies which can be dynamically specified by the service participants. As far as correctness is concerned, [20] considers safety properties such as deadlock freedom and specific progress properties. Our model instead deals also with the property of livelock freedom.

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

References

1. F. Abouzaid and J. Mullins. Model-checking Web Services Orchestrations using BP-calculus. *Electronic Notes in Theoretical Computer Science*, 255:3–21, 2009.
2. T. Basciutti. Model-Checking Web Services. Master's thesis, Department of Computer Science, University Ca' Foscari of Venice, 2010.

3. G. Bernardi, M. Bugliesi, D. Macedonio, and 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)*, pages 327–334. IEEE Computer Society, 2008.
4. M. Bravetti and 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)*, volume 5387 of *LNCS*, pages 37–54. Springer, 2008.
5. M. Bravetti and G. Zavattaro. A Foundational Theory of Contracts for Multi-party Service Composition. *Fundamenta Informaticae*, 89(4):451–478, 2009.
6. S. Carpineti, G. Castagna, C. Laneve, and 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)*, volume 4184 of *LNCS*, page 148162. Springer, 2006.
7. G. Castagna, N. Gesbert, and L. Padovani. A Theory of Contracts for Web Services. In *Proc. of the annual Symposium on Principles of Programming Languages (POPL'08)*, pages 261–272. ACM press, 2008.
8. G. Castagna, N. Gesbert, and L. Padovani. A Theory of Contracts for Web Services. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 31:53–61, 2009.
9. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. The MIT Press, 1999.
10. R. Cleaveland and S. Sims. The NCSU Concurrency Workbench. In *Proc. of International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *LNCS*, pages 394–397. Springer, 1996.
11. G. Dai, X. Bai, and C. Zhao. A Framework for Model Checking Web Service Compositions Based on BPEL4WS. In *Proc. of the IEEE International Conference on e-Business Engineering (ICEBE'07)*, pages 165–172. IEEE Computer Society, 2007.
12. R. Focardi and R. Gorrieri. A Classification of Security Properties for Process Algebras. *Journal of Computer Security*, 3(1):5–33, 1994/1995.
13. R. Focardi and S. Rossi. Information Flow Security in Dynamic Contexts. *Journal of Computer Security*, 14(1):65–110, 2006.
14. J. A. Goguen and J. Meseguer. Security Policies and Security Models. In *Proc. of the IEEE Symposium on Security and Privacy (SSP'82)*, pages 11–20. IEEE Computer Society, 1982.
15. D. Kozen. Results on the Propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
16. C. Laneve and L. Padovani. The *must* Preorder Revisited. In *Proc. of the International Conference on Concurrency Theory (CONCUR'07)*, volume 4703 of *LNCS*, pages 212–225. Springer, 2007.
17. 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)*, volume 1019 of *LNCS*, pages 72–88. Springer, 1995.
18. R. Milner. *Communication and Concurrency*, volume 92 of *Prentice Hall International Series in Computer Science*. Prentice Hall, 1989.
19. M. Müller-Olm. Derivation of Characteristic Formulae. *Electronic Notes in Theoretical Computer Science*, 18, 1998.
20. S. Nakajima. Model-Checking of Safety and Security Aspects in Web Service Flows. In *Proc. of the International Conference of Web Engineering (ICWE'04)*, volume 3140 of *LNCS*, pages 488–501. Springer, 2004.
21. S. Nakajima. Model-Checking Behavioral Specification of BPEL Applications. *Electronic Notes in Theoretical Computer Science*, 151:89–105, 2006.
22. P. Ryan and S. Schneider. Process Algebra and Non-Interference. *Journal of Computer Security*, 9(1/2):75–103, 2001.
23. H. Schlingloff, A. Martens, and K. Schmidt. Modeling and Model Checking Web Services. *Electronic Notes in Theoretical Computer Science*, 126:3–26, 2005.