Metric-Aware Secure Service Orchestration*

Gabriele Costa
Dipartimento di Informatica, Sistemistica e Telematica
Università di Genova
gabriele.costa@unige.it

Fabio Martinelli
Istituto di Informatica e Telematica
Consiglio Nazionale delle Ricerche
fabio.martinelli@iit.cnr.it

Artsiom Yautsiukhin
Istituto di Informatica e Telematica
Consiglio Nazionale delle Ricerche
artsiom.yautsiukhin@iit.cnr.it

Secure orchestration is an important concern in the internet of service. Next to providing the required functionality the composite services must also provide a reasonable level of security in order to protect sensitive data. Thus, the orchestrator has a need to check whether the complex service is able to satisfy certain properties. Some properties are expressed with metrics for precise definition of requirements. Thus, the problem is to analyse the values of metrics for a complex business process.

In this paper we extend our previous work on analysis of secure orchestration with quantifiable properties. We show how to define, verify and enforce quantitative security requirements in one framework with other security properties. The proposed approach should help to select the most suitable service architecture and guarantee fulfilment of the declared security requirements.

1 Introduction

Orchestration of complex web services is a multidimensional problem. Various criteria must be considered when different alternatives exist. Typically, one of such criteria is *security*. Recently, the security issues of service composition are receiving major attention [20, 22, 4, 7, 21, 9]. Among them, formal methods have been successfully applied for modelling and analysing several different aspects of service security. In practice, these techniques generate a formal abstraction of the services under analysis. Then, a verification procedure is applied to find a formal proof of compliance between the model and the security specifications.

The first difficulty arises from service abstraction. Indeed, it is crucial that services are modelled in a "safe" way, i.e., without neglecting any security-relevant behaviour they can generate. The problem is that this feature is not always guaranteed as specification and implementation is often developed independently.

Although several, effective algorithms for software verification exist, e.g., model checking [11], they often require some modification to be applied to web services. Indeed, the algorithms typically check the compliance between a specification and a model and, if the check fails, they return a description of the detected error, e.g., a behaviour of the model that violates the specification. However, web services are designed and developed separately and they commonly have different and independent security requirements. Moreover, they are oriented to the composition and they can produce many different models, i.e., one for each possible orchestration. Hence, the verification process cannot just focus on an illegal orchestration, but should help in finding valid ones.

Service usages are often based on *security metrics*. Metrics conveniently use mathematical values to represent some "qualities" of a service. Several authors, e.g., see [23, 18], proposed mathematical models for the definition and composition of security metrics.

In this paper we propose an extension of previous work (see [12, 13]) on secure service orchestration integrating facilities for composing and verifying security metrics. In particular, we start from the service

*The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant numbers 257930 (ANIKETOS), 256980 (NESSOS) and 257876 (SPACIOS).

Submitted to:
ICE 2012

model proposed by Bartoletti et al. [4]. Roughly, they propose a type and effect system for producing safe abstractions of the behaviour of web services. Then, the authors verify these abstractions against the security policies, locally specified by each service, to find a valid composition.

We extend their model by introducing metric checks and metric annotations on their abstractions. We use a mathematical structure, called c-semiring, in order to generalise our model and be independent from the metrics used for the analysis, but still be able to reason on these metrics. Metric annotations are obtained through a new, improved type and effect system. In this way, we generate metric-annotated abstractions which contain both security and metric requirements. All the requirements are applied to different portions of the service orchestration through a local scope.

The main advantage of this approach is the possibility to model and compose both security and metric requirements in a single framework. Service developers apply security policies and metric checks to some parts of their services. Our type and effect system extracts *history expressions* from the implementation of the services. History expressions safely denote the behaviour of service invocations. Within a history expression, the type and effect system adds extra annotations for metrics, metric checks and security framings. Then, we adopt the same verification procedure described in [4] with special pre-processing steps for assigning correct metric labels to each service. The final result is a complete framework for defining, modelling, verifying, and enforcing both security and metric requirements in order to find valid service orchestrations.

This paper is structured as follows. Section 2 introduces the working example we will develop during our presentation. In Section 3 we describe our extension of the programming language λ^{req} and we define its operational semantics. Then, Section 4 presents our type and effect system and Section 5 describes the analysis of security and metric requirements. Finally, Section 7 concludes the paper.

2 Running example

The travel agency BestTravel offers a travel planning service to its customers. BestTravel exploits existing services for implementing the complex task of (i) booking a connection (consisting of one or more flights) to the destination, (ii) booking a hotel room, (iii) paying the acquired items (i.e., flights and hotel room), and (iv) providing the customer with a signed receipt. As usual in service-oriented architectures, the four subtask described above are provided by existing web services.

The service developer starts from an abstract workflow describing the behaviour of BestTravel and produces a corresponding implementation. The abstract workflow depicts the atomic operations that the service must implement and how they compose each other. In the case of BestTravel, most of the atomic operations are invocations to other services. Figure 1 shows the abstract workflow of BestTravel.

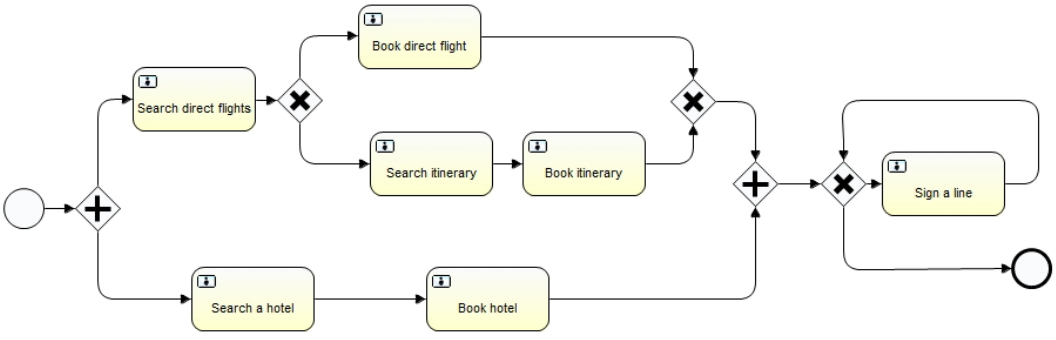

Figure 1: Abstract workflow for BestTravel.

Reading Figure 1 (from left to right), we can understand the service behaviour. In words, a session of BestTravel works as follows. The service runs two procedures in parallel (rooted in ⟡). The first one (upper

e, e'	::=	
	$*$	unit
	r	resource
	x	variable
	$\alpha(e)$	access event
	$\texttt{if}\, b\, \texttt{then}\, e\ \texttt{else}\ e'$	branch
	$\lambda_z x.e$	abstraction
	$e\, e'$	application
	$\varphi[e]$	security framing
	$\gamma\langle e\rangle$	metric framing
	$\texttt{req}_\rho\ \tau \to \tau'$	service request

$$\lambda x.e \stackrel{def}{=} \lambda_z x.e \text{ with } z \notin fv(e)$$

$$\lambda.e \stackrel{def}{=} \lambda x.e \text{ with } x \notin fv(e)$$

$$e; e' \stackrel{def}{=} (\lambda.e')e \qquad \texttt{fork}\, e\, \texttt{and}\, e' \stackrel{def}{=} (e'; \lambda x.x)e$$

$$(\texttt{req}_\rho\ \tau \xrightarrow{\varphi,\gamma} \tau')e \stackrel{def}{=} \varphi\big[\gamma\big\langle(\texttt{req}_\rho\ \tau \to \tau')e\big\rangle\big]$$

where fv is the standard function returning the set of free variables of an expression e.

Table 1: Syntax of λ^{req} and abbreviations

path of the workflow) is responsible for booking a flight connection for the travel destination. In practice, BestTravel invokes a service looking for a direct flight, i.e., *search direct flight*. Then the execution can take two alternative branches ($\diamondsuit$ node): it can invoke a payment service for booking the flight, i.e., *book flight*, or it can start a new research for a multiple-flight connection, namely an *itinerary*, and book it, i.e., *search itinerary* and *book itinerary*. Concurrently, the second process (lower path) invokes services for searching and booking a hotel, i.e., *search hotel* and *book hotel*. When the two parallel procedures terminate, BestTravel iteratively invokes a digital signature service, i.e., *sign line*, for applying integrity and authenticity tokens to the hotel receipt and terminates.

A requirement of BestTravel is to have *risk* level of the performed tasks (in particular, flight booking, hotel reservation and receipt signature) less than 75. Therefore, two problems must be solved: (i) *statically* estimate risk for the composition plans; (2) in case some execution path in the composition plan fails the requirement, *dynamically* check the risk of selected paths and prevent the failure of the requirement if a risky path is selected.

3 Service structure

In this section we present an extended version of λ-calculus, called λ^{req} [4]. First, we extend our previous work with two main novelties: *parallel composition* and *metric facilities*. Parallel agents in this work are defined without modifying the original syntax of the calculus. We obtain it by re-defining the operational semantics of λ^{req}. Second, we incorporate metrics into our formalism using special operations for denoting metric annotations and metric constraints. These operators are interpreted in a c-semiring mathematical structure. Metric facilities allow us to model metrics which are used in service composition.

3.1 Syntax

First, we define the syntax of expressions e, e' as shown in Table 1. Briefly, $*$ is the closed, side effects-free expression, $r, r' \in \mathscr{R}$ denotes system resources and x, y are variables. Access events $\alpha(e), \beta(e')$ represent the access to a certain resource, resulting from the evaluation of the event argument, through a specific operation/channel (e.g., α and β). Conditional term $\texttt{if}\, b\, \texttt{then}\, e\ \texttt{else}\ e'$ represents a branch between two expressions (where b is a boolean guard). A function is defined through the term $\lambda_z x.e$, where e is the function body in which x is the formal parameter and z denotes the function itself (for recursive invocations). Instead, the term $e\, e'$ denotes the application of a function e to a parameter e'. We feel free to use parenthesis for grouping either a function or its argument in order to improve readability. Security framing is used to apply the scope of a security policy φ to a term. We also use metric framing for expressing a term laying in the scope of a metric constraint γ. Finally, a service request $\texttt{req}_\rho\ \tau \to \tau'$ denotes the invocation of a service having a certain functional interface, i.e., $\tau \to \tau'$ shows that the function requires a type τ as input and produces type τ' as output,

1 $\lambda x.$(search_flight_for(x); if is_available then reserve(FLIGHT_No);FLIGHT_No else NO_FLIGHT)

2 $\lambda x.$(search_flight_for(x); if is_available then reserve(FLIGHT_No);FLIGHT_No else if can_overbook then overbook(FLIGHT_No);FLIGHT_No else NO_FLIGHT)

3 $\lambda x.$(generate_travel_to(x);reserve(ITINERARY); insurance(ITINERARY);ITINERARY)

4 $\lambda x.$(generate_travel_to(x);reserve(ITINERARY); ITINERARY)

5 $\lambda x.$(find_hotel_3s(x);book(HOTEL); HOTEL_RESV)

6 $\lambda x.$(if high_season then find_hotel_2s(x) else find_hotel_4s(x); book(HOTEL);HOTEL_RESV)

7 $\lambda x.$((if registered_user then $*$ else var_charge(x));buy(x);RCPT)

8 $\lambda x.$(const_charge(x);buy(x);RCPT)

9 λx.sign_64(x);SIGNED_DOC

10 λx.sign_128(x);SIGNED_DOC

Figure 2: Implementation of the services of Example 1.

and is labelled with a unique identifier ρ. Although, it is hard to create the λ^{req} representation for non experts such the model may be created automatically, similar to transformation of Java code [3].

For the sake of presentation, we introduce some useful abbreviations (see Table 1). Moreover, to improve the readability we feel free to use simple expressions for conditional guards, e.g., `is_available` or `is_empty`, which have a straightforward interpretation in the context we use them. We also use upper cases for resources, e.g., HOTEL and FLIGHT, and lower cases for actions, e.g., book(...) and buy(...).

According to the standard λ^{req} theory, we define security policies through *usage automata* [2]. Usage automata resemble non deterministic finite state automata (NFA) defined over the alphabet of access events. A sequence of actions is compliant with a certain policy if its corresponding usage automata does not reach a final, offending state reading the trace, i.e., valid traces are those rejected by the automata (see [2] for details).

Our main focus in this section is on the definition of metric constraints. Indeed, we introduce a syntax for defining metric checks which then we apply through metric framing. In particular a metric check has the form $\gamma = T \geq_T d$ where T is a metric name, $\geq_T$ is its order relation and d is an element of T. Here we slightly abuse our notation for the sake of simplicity, in order to show that the metric computed for a business process must be better than some predefined value (i.e., threshold). In practice, a metric check is satisfied by a value d' if $d' \geq_T d$. If so we write $d' \in \gamma$.

Example 1. We continue our running example. We assume the (sets of) resources: $\mathscr{I} = \{\texttt{ITINERARY}\}$, $\mathscr{F} = \{\texttt{FLIGHT\_No}, \texttt{NO\_FLIGHT}\}$, $\mathscr{H} = \{\texttt{HOTEL\_RESV}\}$, $\mathscr{B} = \mathscr{I} \cup \mathscr{F} \cup \mathscr{H}$ and $\mathscr{D} = \{\texttt{RCPT}, \texttt{SIGNED\_DOC}\}$. In Figure 2 we propose the λ^{req} implementation of the services informally introduced in Section 2.

Intuitively, service 1 receives an input airport x and searches a direct flight (action `search_flight_for`). Then, depending on the `is_available` boolean flag, the service either reserves a seat (`reserve`) and returns the flight number FLIGHT_No, or returns the NO_FLIGHT value. Service 2 works similarly. The main difference is that, if the flight is not available, it checks whether it is possible to make an overbooking reservation (`can_overbook` flag) and proceeds with the reservation (`overbook`) before returning the flight number or the NO_FLIGHT value. Instead, service 3 finds a sequence of flights for the destination, namely an itinerary (`generate_travel_to`). Then the itinerary is reserved (`reserve`), a travel insurance is stipulated (`insurance`) and the itinerary is returned. Service 4 resembles 3, but no insurance is activated. Hotel booking services, i.e., services 5 and 6, receive a destination city x and book an hotel (action `book`) before returning the hotel reservation HOTEL_RESV. The main difference between the two services is that service 5 looks for a 3 stars hotel (action `find_hotel_3s`) while service 6, after discriminating on the flag `high_season`, searches either a 2 stars or a 4 stars hotel (actions `find_hotel_2s` and `find_hotel_4s`, respectively). Payment services 7 and 8 receive an item identifier x and return an electronic receipt RCPT after performing a purchase operation (action `buy`). However, while 8 charges the operation with a constant, extra amount (action `const_charge`), service

1 $\lambda x.(\gamma\langle\lambda_z y.$
2 $\vdots\ \vdots$ `if is_empty then SIGNED_DOC else` $(\mathtt{req}_{\rho_1}\ \mathscr{D}\to\mathscr{D})y;zy\,\rangle$
3 $\vdots$ `fork` $\gamma\langle(\lambda y'.(\mathtt{req}_{\rho_2}\ \mathscr{B}\to\mathscr{D})y')((\mathtt{req}_{\rho_3}\ \mathscr{C}\to\mathscr{H})\mathtt{CITY})\rangle$
4 $\vdots$ `and` $\gamma\langle\lambda y''.(\mathtt{if\ no\_direct\_flight\ then}\ (\mathtt{req}_{\rho_4}\ \mathscr{B}\to\mathscr{D})((\mathtt{req}_{\rho_5}\ \mathscr{A}\to\mathscr{I})\,\mathtt{AIRPORT})$
5 $\vdots\ \vdots$ $\mathtt{else}\ (\mathtt{req}_{\rho_6}\ \mathscr{B}\to\mathscr{D})y'')((\mathtt{req}_{\rho_7}\ \mathscr{A}\to\mathscr{F})\,\mathtt{AIRPORT})\,\rangle\,)$

Figure 3: Implementation of BestTravel.

7 applies either no commission charge or a variable amount (action `var_charge`). Finally, signing services accept a document x and return a signed version of it `SIGNED_DOC`. The only difference between them is that 9 uses a 64 bit key for the signing process (action `sign_64`) while 10 uses a 128 bit ones (`sign_128`).

Note, that with several alternative services which provide the same functionality we have several different possible execution paths which have different security properties. □

Example 2. We assume the existence of the resources: $\mathscr{A} = \{\mathtt{AIRPORT}\}$ and $\mathscr{C} = \{\mathtt{CITY}\}$. In Figure 3 we propose a λ^{req} implementation of the workflow of the BestTravel service, called e_B.

In words, e_B carries out three tasks: it concurrently runs (i) a hotel booking process (line 4) and (ii) a flight booking one (lines 5-6) and, then, (iii) executes a signature procedure (line 2). The first process consists of an invocation to a hotel search service using the resource `CITY`. The result is then passed as input for (an invocation to) a payment service. Similarly, the second process requests a itinerary searching service using the resource `AIRPORT`. Then, according the evaluation of the guard `no_direct_flight`, the service either starts a new request to flight searching service and proceeds with the payment or just invokes a payment service. The final result of this concurrent execution is the document returned by the first process. This value is then used as the actual parameter of the last operation of the service. It consists of a recursive function which, depending on the guard `is_empty`, can either return the resource `SIGNED_DOC` or invoke a signing service and loop.

All the three tasks are subject to a metric requirement $\gamma = \mathbf{Risk} \leq 75$, i.e., each of them must be executed under a risk factor lower than 75 (\$). □

3.2 C-Semirings

Our framework exploits the notion of *c-semiring* for the abstraction of metrics and operators over metrics [6]. Usage of this mathematical structure allow us to provide a generic framework for all metrics which could be considered as c-semirings. A c-semiring consists of a set of values D (e.g., natural or real numbers), and two types of operators: multiplication ($\otimes$) and summation ($\oplus$) of values and constraints. Formally, a c-semiring is defined as follows (see the work of S. Bistarelli et. al., for more details [6]).

Definition 1. A *c-semiring* T is a tuple $\langle D, \oplus, \otimes, \boldsymbol{0}, \boldsymbol{1}\rangle$ where

- D is a (possibly infinite) set of elements and $\boldsymbol{0}, \boldsymbol{1} \in D$;
- $\oplus$, being an addition defined over D, is a binary, commutative (i.e., $d_1, d_2 \in D \Rightarrow d_1 \oplus d_2 = d_2 \oplus d_1$) and associative (i.e., $d_1, d_2, d_3 \in D \Rightarrow d_1 \oplus (d_2 \oplus d_3) = (d_1 \oplus d_2) \oplus d_3$) operator such that $\mathbf{0}$ is its *unit* element (i.e., $d_1 \in D \Rightarrow (d_1 \oplus \boldsymbol{0} = d_1 = \boldsymbol{0} \oplus d_1)$;
- $\otimes$, being a multiplication over D, is a binary, commutative and associative operator such that $\mathbf{1}$ is its *unit* element and $\mathbf{0}$ is its *absorbing* element (i.e., $d_1 \in D \Rightarrow d_1 \otimes \mathbf{0} = \mathbf{0} = \mathbf{0} \otimes d_1$);
- $\otimes$ is distributive over additive operator $(d_1 \otimes (d_2 \oplus d_3)) = (d_1 \otimes d_2) \oplus (d_1 \otimes d_3)$;

In this work we focus on a special subset of c-semirings:

Definition 2. c^*-semiring is a c-semiring with $\oplus$ satisfying the following condition: $\forall d_1, d_2 \in D\ d_1 \oplus d_2 = d_1$ *or* $d_1 \oplus d_2 = d_2$

$$(\mathtt{S{-}Ev_1})\quad \frac{\langle \eta,d,e\rangle \to_\pi \langle \eta',d',e'\rangle}{\langle \eta,d,\alpha(e)\rangle \to_\pi \langle \eta',d',\alpha(e')\rangle} \qquad (\mathtt{S{-}Ev_2})\quad \frac{F(\alpha,r)=d'}{\langle \eta,d,\alpha(r)\rangle \to_\pi \langle \eta\,\alpha(r),d\otimes d',*\rangle}$$

$$(\mathtt{S{-}Req})\quad \frac{e_\ell : \tau \xrightarrow{H} \tau' \in \mathtt{Srv} \qquad \pi(\rho)=\ell}{\langle \eta,d,(\mathtt{req}_\rho\ \tau\to\tau')v\rangle \to_\pi \langle \eta,d,e_\ell v\rangle} \qquad (\mathtt{S{-}App_1})\quad \frac{\langle \eta,d,e_1\rangle \to_\pi \langle \eta',d',e_1'\rangle}{\langle \eta,d,e_1e_2\rangle \to_\pi \langle \eta',d',e_1'e_2\rangle}$$

$$(\mathtt{S{-}App_2})\quad \frac{\langle \eta,d,e_2\rangle \to_\pi \langle \eta',d',e_2'\rangle}{\langle \eta,d,e_1e_2\rangle \to_\pi \langle \eta',d',e_1e_2'\rangle} \qquad (\mathtt{S{-}App_3})\quad \langle \eta,d,(\lambda_z x.e)v\rangle \to_\pi \langle \eta,d,e\{v/x,\lambda_z x.e/z\}\rangle$$

$$(\mathtt{S{-}Sec_1})\quad \frac{\langle \eta,d,e\rangle \to_\pi \langle \eta',d',e'\rangle \quad \eta' \models \varphi}{\langle \eta,d,\varphi[e]\rangle \to_\pi \langle \eta',d',\varphi[e']\rangle} \qquad (\mathtt{S{-}Sec_2})\quad \frac{\eta \models \varphi}{\langle \eta,d,\varphi[v]\rangle \to_\pi \langle \eta,d,v\rangle}$$

$$(\mathtt{S{-}Met_1})\quad \frac{\langle \eta,d,e\rangle \to_\pi \langle \eta',d',e'\rangle \quad d' \in \gamma}{\langle \eta,d,\gamma\langle e\rangle\rangle \to_\pi \langle \eta',d',\gamma\langle e'\rangle\rangle} \qquad (\mathtt{S{-}Met_2})\quad \frac{d \in \gamma}{\langle \eta,d,\gamma\langle v\rangle\rangle \to_\pi \langle \eta,d,v\rangle}$$

$$(\mathtt{S{-}If})\quad \langle \eta,d,\mathtt{if}\ b\ \mathtt{then}\ e_{tt}\ \mathtt{else}\ e_{ff}\rangle \to_\pi \langle \eta,d,e_{\mathbf{B}(b)}\rangle$$

Table 2: Operational semantics of λ^{req}

Definition 3. $\leq_T$ is a total order over the set D, such that $d_1 \leq_T d_2$ iff $d_1 \oplus d_2 = d_2$.

In this work we need a reverse operation for summation $\oplus^{-1}$ which is defined as follows.

Definition 4. $d_1 \oplus^{-1} d_2 = d_1$ iff $d_1 \oplus d_2 = d_2$.

In words, this operation always returns the worst possible value.

Property 1. Operation $\oplus^{-1}$ is associative, commutative, idempotent, distributive over $\otimes$, and monotone[1].

Example 3. Regarding to the security targets BestTravel is going to use two metrics: trust and risk. Trust is often computed as a probability that the requested service is going to behave as agreed. Thus, trust could be seen as a value between 0 and 1, which is aggregated by multiplying and the higher value is considered better than a lower one. C*-semiring for trust value formally is defined as follows: $\langle [0,1], max, \times, 0, 1\rangle$. This type of c-semirings is known as possibilistic semiring.

Risk, considered as possible losses, has the domain of positive real numbers. Multiplication of risks is summation of possible losses, when the lower value is, naturally, considered more preferable than the higher one. Therefore, c*-semiring for risk could be seen as $\langle N^+ \cup \{\infty\}, min, +, \infty, 0\rangle$, known as tropical semiring. □

3.3 Operational Semantics

Service execution is driven by the operational semantics defined in Table 2. Intuitively, a computation step consists of a reduction from a source configuration to a target one. Configurations are tuples $\langle \eta,d,e\rangle$ where η is an execution trace, i.e., the sequence of events performed so far (ε denotes the empty execution trace); d is the current metric value; and e is a λ^{req} term, which describes the part of the service under evaluation. The operational semantics is driven by a composition plan π which is responsible for providing a mapping between each service request and an actual service, in symbols $\pi(\rho)=\ell$ where ρ and ℓ are request and service identifiers, respectively. In the following we also use $\to_\pi^*$ for the transitive closure of $\to_\pi$.

Below, we provide an informal explanation of the operational semantics rules. To be performed, an action α requires its argument e to be evaluated first (rule ($\mathtt{S{-}Ev_1}$)). If the action target reduces to a resource r, the action takes place and the current history η is extended with the corresponding event $\alpha(r)$ (rule ($\mathtt{S{-}Ev_2}$)). Also,

[1] A link with proofs:http://www.iit.cnr.it/staff/artsiom.yautsiukhin/Resources/ICE-Proofs.pdf.

ACTION	RESOURCE	VALUE
reserve	FLIGHT_No	15
reserve	ITINERARY	15
overbook	FLIGHT_No	20
insurance	ITINERARY	10
find_hotel_2s	CITY	30
find_hotel_3s	CITY	20
find_hotel_4s	CITY	15

ACTION	RESOURCE	VALUE
book	HOTEL	20
var_charge	·	8
const_charge	·	5
buy	FLIGHT_No	10
buy	HOTEL_RESV	10
buy	ITINERARY	20
sign_64	·	1

Table 3: Definition of function $F_{\mathbf{Risk}}$.

the current metric is updated with the metric value for the event $\alpha(r)$. F is a metric and context-dependent predefined function which assigns a metric value to every event. In practice, function F can be found analytically (e.g., risk=probability$\times$impact), derived form past experience, i.e., using monitoring or assigned by experts (e.g., number of successful virus attacks). A conditional expression is reduced to one of its branches (i.e., e_{tt} and e_{ff}[2]) depending on the value of its guard b (rule $(\mathtt{S{-}If})$). Here we assume an evaluation function $\mathbf{B}$, assigning to each possible guard a boolean value, is to be defined. Rules $(\mathtt{S{-}App_1})$, $(\mathtt{S{-}App_2})$ and $(\mathtt{S{-}App_3})$ define the behaviour of function application. Briefly, a function e and its argument e' are both reduced to values, i.e., terms that admit no further reduction. The steps of the two reductions are executed in a non deterministic way, without any fixed priority between the choice of $(\mathtt{S{-}App_1})$ and $(\mathtt{S{-}App_2})$. When both computations generate a value, i.e., a lambda abstraction and its argument, the application reduces to the body of the function where the formal parameter x is replaced by the actual value v and the variable z is substituted with the function itself (rule $(\mathtt{S{-}App_3})$). Note that, along the paper, we use v, v' to denote *values*, i.e., closed, effect-free terms being either $*$, resources, λ-abstractions or service requests. Rules $(\mathtt{S{-}Sec_1})$ and $(\mathtt{S{-}Sec_2})$ define the behaviour of the security framing. Basically, a security framing behaves as its target unless it tries to extend the current history η to an illegal trace. When the target expression reduces to a value, the policy framing can be removed, i.e., the corresponding security check is deactivated, if the current history is a legal one. Similarly, $(\mathtt{S{-}Met_1})$ and $(\mathtt{S{-}Met_2})$ rule metric checks. In words, a metric check forces metric values generated during the execution of a term e to comply with a constraint γ. Finally, service requests (rule $(\mathtt{S{-}Req})$) works by running the service e_ℓ with actual parameter v. Among all the compatible services, i.e., those having the same behavioural interface specified by the request ρ, appearing in the service repository $\mathtt{Srv}$[3], one is selected according to the current composition plan π. Note that the interface of actual services is also annotated with a *history expression H* which represent the service contract (see Section 4 for more details on this point).

Example 4. Let e_1 be the implementation of service 1 proposed in Example 1. We assume $\mathbf{B}(\mathtt{is\_available}) = tt$, and consider the semiring **Risk** introduced in Example 3 and the function $F_{\mathbf{Risk}}$ which returns the values shown in Table 3 (where missing entry evaluate to 0 and · stands for any compatible value). Then, we have the following computation for $\bigstar = \langle \varepsilon, 0, (e_1)\mathtt{AIRPORT} \rangle$ (where AIRPORT is a resource in $\mathscr{A}$).

$$\bigstar \to_\pi \left\langle \varepsilon, 0, \begin{array}{l}\mathtt{search\_flight\_for(AIRPORT);}\\ \mathtt{if\ is\_available}\\ \quad\mathtt{then\ reserve(FLIGHT\_No);}\\ \quad\ \mathtt{FLIGHT\_No}\\ \quad\mathtt{else\ NO\_FLIGHT}\end{array} \right\rangle \to_\pi^* \left\langle \mathtt{search\_flight\_for(AIRPORT)}, 0, \begin{array}{l}\mathtt{if\ is\_available}\\ \quad\mathtt{then}\\ \mathtt{reserve(FLIGHT\_No);}\\ \mathtt{FLIGHT\_No}\\ \quad\mathtt{else\ NO\_FLIGHT}\end{array} \right\rangle$$

$$\to_\pi \left\langle \mathtt{search\_flight\_for(AIRPORT)}, 0, \begin{array}{l}\mathtt{reserve(FLIGHT\_No);}\\ \mathtt{FLIGHT\_No}\end{array} \right\rangle \to_\pi^* \left\langle \begin{array}{c}\mathtt{search\_flight\_for(AIRPORT)}\\ \mathtt{reserve(FLIGHT\_No)}\end{array}, 15, \mathtt{FLIGHT\_No} \right\rangle$$

[2]Where *tt* and *ff* stand for "true" and "false", respectively.

[3]Here we assume a service repository to be always available at runtime. In short, a repository is a finite set of tuples, each of them containing at least the service interface and being uniquely identified by the service location ℓ.

$$H, H' ::= \varepsilon \mid h \mid \alpha(r) \mid H \cdot H' \mid H + H' \mid H \mid H' \mid d\#H \mid \varphi[H] \mid \gamma\langle H \rangle \mid \mu h.H$$

Table 4: Syntax of history expressions

In words, the computation proceeds as follows. The first step consists in applying the rule $(\mathtt{S{-}App_3})$ which, in practice, replaces all the occurrences of x with `AIRPORT`. The second reduction collapses two rules, i.e., $(\mathtt{S{-}Ev_2})$ and $(\mathtt{S{-}App_3})$. As a result of the rule $(\mathtt{S{-}Ev_2})$ a new event, that is `search_flight_for(AIRPORT)`, is added to the execution trace ε. Also, according to the given definition of $F_{\mathbf{Risk}}$, the current metric is updated. Recalling the c*-semiring specified in Example 3, we note that the multiplication operation over risk values is the sum, then $0 \otimes 0 = 0 + 0 = 0$. The subsequent step evaluates the conditional guard `is_available` and chooses the "then" branch (rule $(\mathtt{S{-}If})$). Finally, the last piece of computation repeats the operations described above and updates the current configuration by both adding a new event to the execution history and changing the current metric value (i.e., $0 \otimes 15 = 0 + 15 = 15$). Since the term appearing in the last configuration is a value, i.e., the resource `FLIGHT_No`, the computation terminates. □

4 Type and effect system

In this section we present our proposal for a type and effect system. It derives from the type and effect system presented in [4] from which it inherits most of its rules.

4.1 History expressions

Briefly, a type and effect system carries out the extraction of behavioural description from a certain expression while typing it. We use *history expressions* for representing the behaviour of a program in terms of the execution histories it can generate at runtime.

The main novelties introduced by our type and effect system are (i) parallel composition and (ii) metric annotation. Parallel composition denotes two elements which can run concurrently, in an interleaving fashion. Instead, metric annotation associate a metric value to a certain behaviour. Table 4 reports the syntax of history expressions.

A history expression can be the empty one ε, a variable h or an access event $\alpha(r)$. Valid history expressions are also concatenations ($H \cdot H'$), unions ($H + H'$), parallel compositions ($H \mid H'$), metric-annotated expressions ($d\#H$), security framings ($\varphi[H]$), metric checks ($\gamma\langle H \rangle$) and least fix-point, recursive expressions ($\mu h.H$).

A history expression denotes a set of execution traces. We use a denotational semantics to bind each history expression to the corresponding set of traces. The semantic function $[\![\cdot]\!]_{\cdot}$ is defined in Table 5. Note that we use the environment δ for mapping variables to set of traces.

A ε expression denotes the singleton containing the empty trace (we use ε for both void history expressions and empty traces as they are clearly identified by the context). The semantics of a variable h corresponds to the set of histories associated to it in δ. A history expression $\alpha(r)$ denotes the singleton $\{\alpha(r)\}$. The semantics of a sequence $H \cdot H'$ is the set of traces $\eta\eta'$ such that $\eta \in [\![H]\!]_\delta$ and $\eta' \in [\![H']\!]_\delta$. Similarly, the semantics of a choice is the union between the sets denoted by the two sub-expressions. Parallel history expressions $H \mid H'$ denote the set of all the possible interleaving of traces belonging to the two sub-expressions. Interleaving semantics is defined through the binary operator $\binom{\cdot}{\cdot}$. Intuitively, if one of the two considered histories is ε, the operator $\binom{\cdot}{\cdot}$ returns the other one. Instead, for non-empty traces it generates all the possible sequences representing concurrent executions. This process is obtained by considering all the possible prefixes of one trace, adding the first action of the other trace and recursively applying the $\binom{\cdot}{\cdot}$ operator to the remaining "tails". In the style of [5], security framing denotes execution histories wrapped between two special actions $[_\varphi$ and $]_\varphi$ (for brevity, we write $\varphi[X]$ in place of $[_\varphi \cdot X \cdot]_\varphi$). These special actions mark the activation and deactivation points of a policy. Following a similar reasoning, the semantics of $\gamma\langle H \rangle$ is the set of traces denoted by H wrapped by the special

$$[\![\varepsilon]\!]_\delta = \{\varepsilon\} \qquad [\![\alpha(r)]\!]_\delta = \{\alpha(r)\} \qquad [\![H \cdot H']\!]_\delta = [\![H]\!]_\delta [\![H']\!]_\delta \qquad [\![H + H']\!]_\delta = [\![H]\!]_\delta \cup [\![H']\!]_\delta$$

$$[\![\varphi[H]]\!]_\delta = \varphi[[\![H]\!]_\delta] \qquad [\![d\#H]\!]_\delta = [\![H]\!]_\delta \qquad [\![\gamma\langle H\rangle]\!]_\delta = \gamma\langle [\![H]\!]_\delta\rangle \qquad [\![h]\!]_\delta = \delta(h)$$

$$[\![H \mid H']\!]_\delta = \bigcup_{\eta \in [\![H]\!]_\delta, \eta' \in [\![H']\!]_\delta} \binom{\eta}{\eta'} \qquad [\![\mu h.H]\!]_\delta = \bigcup_{n>0} f^n(\varepsilon) \quad \text{where } f(X) = [\![H]\!]_{\delta\{X/h\}}$$

where the binary function $\binom{\cdot}{\cdot}$ is recursively defined as follows.

$$(1) \quad \binom{\eta}{\varepsilon} = \{\eta\} \qquad (2) \quad \binom{\eta}{\alpha\eta'} = \left\{ \eta_1 \alpha \tilde{\eta} \;\middle|\; \tilde{\eta} \in \binom{\eta'}{\eta_2} \wedge \eta_1\eta_2 = \eta \right\}$$

Table 5: Denotational semantics

actions $\langle_\gamma$ and $\rangle_\gamma$ (with the obvious meaning). Finally, $\mu h.H$ denotes a fix point operation over the set of traces denoted by H (see [5] for further detail).

Moreover, we introduce a partial order relation $\sqsubseteq$ between history expressions such that $H \sqsubseteq H' \Leftrightarrow \forall\delta.[\![H]\!]_\delta \subseteq [\![H']\!]_\delta$.

4.2 Typing relation

In the following we introduce our typing rules. The main difference with respect to the rules proposed in previous works is that here we generate metric annotated history expressions during the typing process. Before presenting the typing rules, we need to introduce *types* and *type environments*.

Definition 5. (Types and type environments)

$$\tau, \tau' ::= \mathit{unit} \mid \mathscr{R} \mid \tau \xrightarrow{H} \tau' \qquad\qquad \Gamma, \Gamma' ::= \emptyset \mid \Gamma; x : \tau$$

A type can be both a simple type, i.e., *unit* or the resource domain $\mathscr{R}$[4], or a function from type τ to type τ'. Functional types also carry a history expression H which represents the latent effect of invoking the function. Then, a type environment Γ, being either the empty one $\emptyset$ or the one obtained through a new binding $\Gamma; x : \tau$, is a mapping from variables to types.

The typing relation has the form $\Gamma, H \vdash e : \tau$. It must be read as "under the environment Γ and carrying the effect H, expression e has type τ". The rules in Table 6 define the typing relation.

Briefly, the expression $*$ has *unit* type and generates no side effects ($H = \varepsilon$, rule (T−Unit)) while a resource r, being also side effect free, has type $\mathscr{R}$ (rule (T−Res)). The type of a variable x depends on the typing context provided by Γ (rule (T−Var)). Abstractions (rule (T−Abs)) has an empty effect and produce a functional type $\tau \xrightarrow{H} \tau'$ from their input to their output types. The latent effect H is the one obtained from typing the function body. Rule (T−Ev) requires more attention. Indeed, we say that an expression $\alpha(e)$, having type *unit*, generates a history expression which is the sequence between the history expression deriving from typing its argument e and the summation (i.e., a finite sequence of choice operators) of all the possible access actions α to a compatible resource $r \in \mathscr{R}$. Also, all of these access events are annotated with the metric value provided by the function F. The application of a function e to an argument e', i.e., rule (T−App), has type equal to the return type of e and a history effect which is the sequence between (1) the two effects of e and e' in parallel and (2) the latent effect of the function. Security and metric framing (rules (T−Frm) and (T−Met)) have the same

[4]For simplicity here we assume a single set $\mathscr{R}$, but, in general, we assume to have a finite number of resource domains $\mathscr{R}_1, \ldots, \mathscr{R}_n$ s.t. $\bigcup_i \mathscr{R}_i = \mathscr{R}$

$$(\texttt{T}-\texttt{Unit})\ \Gamma,\varepsilon \vdash * : \mathit{unit} \quad (\texttt{T}-\texttt{Res})\ \Gamma,\varepsilon \vdash r : \mathscr{R} \quad (\texttt{T}-\texttt{Var})\ \Gamma,\varepsilon \vdash x : \Gamma(x) \quad (\texttt{T}-\texttt{Abs})\ \frac{\Gamma; x:\tau; z:\tau \xrightarrow{H} \tau', H \vdash e : \tau'}{\Gamma,\varepsilon \vdash \lambda_z x.e : \tau \xrightarrow{H} \tau'}$$

$$(\texttt{T}-\texttt{Ev})\ \frac{\Gamma, H \vdash e : \mathscr{R}}{\Gamma, H \cdot \sum_{r\in\mathscr{R}} (F(\alpha,r)\#\alpha(r)) \vdash \alpha(e) : \mathit{unit}} \quad (\texttt{T}-\texttt{App})\ \frac{\Gamma, H \vdash e : \tau \xrightarrow{H''} \tau' \quad \Gamma, H' \vdash e' : \tau}{\Gamma, (H \mid H') \cdot H'' \vdash e\,e' : \tau'} \quad (\texttt{T}-\texttt{Frm})\ \frac{\Gamma, H \vdash e : \tau}{\Gamma, \varphi[H] \vdash \varphi[e] : \tau}$$

$$(\texttt{T}-\texttt{Met})\ \frac{\Gamma, H \vdash e : \tau}{\Gamma, \gamma\langle H\rangle \vdash \gamma\langle e\rangle : \tau} \quad (\texttt{T}-\texttt{If})\ \frac{\Gamma, H \vdash e : \tau \quad \Gamma, H \vdash e' : \tau}{\Gamma, H \vdash \texttt{if}\ g'\ \texttt{then}\ e\ \texttt{else}\ e' : \tau} \quad (\texttt{T}-\texttt{Wkn})\ \frac{\Gamma, H \vdash e : \tau \quad H \sqsubseteq H'}{\Gamma, H' \vdash e : \tau}$$

$$(\texttt{T}-\texttt{Req})\ \frac{I = \{H \mid e_\ell : \tau \xrightarrow{H} \tau' \in \texttt{Srv}\}}{\Gamma,\varepsilon \vdash \texttt{req}_\rho\ \tau \to \tau' : \tau \xrightarrow{\sum_{X\in I} X} \tau'}$$

Table 6: Typing relation

type as their targets and produce wrapped history expressions. Rule $(\texttt{T}-\texttt{Wkn})$ says that we can always type an expression under a more general history expression. Finally, rule $(\texttt{T}-\texttt{Req})$ says that a service request has the same type of its signature but for its latent effect which is obtained as the disjunction of all the (latent effects of the) possible servers appearing in the repository Srv.

Example 5. Consider service 9, we call its implementation e_9, of Example 1. Writing it without abbreviations we obtain: $e_9 = \lambda_z x.(\lambda_w y.\texttt{SIGNED\_DOC})\texttt{sign\_64}(x)$. Then consider the function $F_{\textbf{Risk}}$ of Table 3. We type e_9 as follows.

$$\dfrac{\dfrac{\dfrac{\Gamma', \varepsilon \vdash \texttt{SIGNED\_DOC} : \mathscr{D}}{\Gamma, \varepsilon \vdash \lambda_w y.\texttt{SIGNED\_DOC} : \tau \xrightarrow{\varepsilon} \mathscr{D}} \quad \dfrac{\Gamma(x) = \mathscr{D}}{\Gamma, H_9 \vdash \texttt{sign\_64}(x) : \mathit{unit}}}{\Gamma, H_9 \vdash (\lambda_w y.\texttt{SIGNED\_DOC})\texttt{sign\_64}(x) : \mathscr{D}}}{\emptyset, \varepsilon \vdash e_9 : \mathscr{D} \xrightarrow{H_9} \mathscr{D}}$$

where $H_9 = 1\#\texttt{sign\_64}(\texttt{RCPT}) + 1\#\texttt{sign\_64}(\texttt{SIGNED\_DOC})$, $\Gamma = x : \mathscr{D}; z : \mathscr{D} \xrightarrow{H_9} \mathscr{D}$ and $\Gamma' = \Gamma; y : \tau; w : \tau \xrightarrow{\varepsilon} \mathscr{D}$. Following a similar reasoning we type all the services of example 1 as shown in Figure 4. For brevity, in the following we use H_i to denote the latent effect of service e_i. □

Example 6. Using the notation introduced in the previous examples for denoting the history expressions of services, we type the BestTravel implementation e_B as in Figure 5. We call H_B the latent effect labelling the arrow type of e_B. □

The main result on the type and effect system is *type safety*. In words, type safety guarantees that effects produced by the type and effect system safely denote the behaviour of services.

Theorem 1. *If* $\Gamma, H \vdash e : \tau$ *and* $\langle \varepsilon, d, e\rangle \to_\pi^* \langle \eta, d', v\rangle$ *then* $\forall \delta.\eta \in [\![H]\!]_\delta$.

Interestingly, the extensions presented in this paper do not invalidate this result originally proved by Bartoletti et. al. [4]. In the next section, we show that history expressions safety is also preserved under metric factorization.

5 Security and metric analysis

5.1 History expressions and semirings

Metric annotations are used to label a history expression with metric values which are expected to be produced dynamically. However, metric annotations are locally associated with parts of a history expression while, in general, it would be preferable to have a single value labelling the whole expression. In particular, we are interested in a procedure which turns a history expression into a corresponding normal form.

$$
\begin{array}{ll}
e_1 : \mathscr{A} & \xrightarrow{0\#\texttt{search\_flight\_for(AIRPORT)}\cdot \left(\left(\begin{array}{c}15\#\texttt{reserve(FLIGHT\_No)}+\\ 0\#\texttt{reserve(NO\_FLIGHT)}\end{array}\right)+\left(\begin{array}{c}20\#\texttt{overbook(FLIGHT\_No)}+\\ 0\#\texttt{overbook(NO\_FLIGHT)}+\varepsilon\end{array}\right)\right)} \mathscr{F} \\
e_2 : \mathscr{A} & \xrightarrow{(0\#\texttt{search\_flight\_for(AIRPORT)})\cdot(15\#\texttt{reserve(FLIGHT\_No)}+0\#\texttt{reserve(NO\_FLIGHT)})} \mathscr{F} \\
e_3 : \mathscr{A} & \xrightarrow{(0\#\texttt{generate\_travel\_to(AIRPORT)})\cdot(15\#\texttt{reserve(ITINERARY)})\cdot(10\#\texttt{insurance(ITINERARY)})} \mathscr{I} \\
e_4 : \mathscr{A} & \xrightarrow{(0\#\texttt{generate\_travel\_to(AIRPORT)})\cdot(15\#\texttt{reserve(ITINERARY)})} \mathscr{I} \\
e_5 : \mathscr{C} & \xrightarrow{(20\#\texttt{find\_hotel\_3s(CITY)})\cdot(20\#\texttt{book(HOTEL)})} \mathscr{H} \\
e_6 : \mathscr{C} & \xrightarrow{(30\#\texttt{find\_hotel\_2s(CITY)}+15\#\texttt{find\_hotel\_4s(CITY)})\cdot(20\#\texttt{book(HOTEL)})} \mathscr{H} \\
e_7 : \mathscr{B} & \xrightarrow{\varepsilon+\left(\left(\begin{array}{c}8\#\texttt{var\_charge(ITINERARY)}+\\ 8\#\texttt{var\_charge(FLIGHT\_No)}+\\ 8\#\texttt{var\_charge(NO\_FLIGHT)}+\\ 8\#\texttt{var\_charge(HOTEL\_RESV)}\end{array}\right)\cdot\left(\begin{array}{c}20\#\texttt{buy(ITINERARY)}+\\ 10\#\texttt{buy(FLIGHT\_No)}+\\ 0\#\texttt{buy(NO\_FLIGHT)}+\\ 10\#\texttt{buy(HOTEL\_RESV)}\end{array}\right)\right)} \mathscr{D} \\
e_8 : \mathscr{B} & \xrightarrow{\left(\begin{array}{c}5\#\texttt{const\_charge(ITINERARY)}+\\ 5\#\texttt{const\_charge(FLIGHT\_No)}+\\ 5\#\texttt{const\_charge(NO\_FLIGHT)}+\\ 5\#\texttt{const\_charge(HOTEL\_RESV)}\end{array}\right)\cdot\left(\begin{array}{c}20\#\texttt{buy(ITINERARY)}+\\ 10\#\texttt{buy(FLIGHT\_No)}+\\ 0\#\texttt{buy(NO\_FLIGHT)}+\\ 10\#\texttt{buy(HOTEL\_RESV)}\end{array}\right)} \mathscr{D} \\
e_9 : \mathscr{D} & \xrightarrow{1\#\texttt{sign\_64(RCPT)}+1\#\texttt{sign\_64(SIGNED\_DOC)}} \mathscr{D} \\
e_{10} : \mathscr{D} & \xrightarrow{0\#\texttt{sign\_128(RCPT)}+0\#\texttt{sign\_128(SIGNED\_DOC)}} \mathscr{D}
\end{array}
$$

Figure 4: Types inferred from the services of Example 1.

$$
e_B : unit \xrightarrow{\left(\gamma\left\langle (H_1+H_2)\cdot\left(\begin{array}{c}(H_7+H_8)\\ +\\ ((H_3+H_4)\cdot(H_7+H_8))\end{array}\right)\right\rangle \middle| \gamma\left\langle\begin{array}{c}(H_5+H_6)\\ \cdot\\ (H_7+H_8)\end{array}\right\rangle\right)\cdot\gamma\langle\mu h.((H_9+H_{10})\cdot h+\varepsilon)\rangle} \mathscr{D}
$$

Figure 5: Type of BestTravel.

Definition 6. A history expression H is said to be in *metric normal form* (MNF), iff $H = d\#H'$ and H' contains no metric annotations.

In Table 7 we propose a set of equivalences that we use to move and compose metric annotations appearing in history expressions. The rules in Table 7 define the correspondence between the history expressions and the semiring operators. In particular, we can always add a multiplication-neutral annotation to a history expression, nested annotations are commutative and can be reduced to a semiring multiplication and choice correspond to the inverse of a semiring addition, namely a subtraction. Also parallel composition can be annotated with the (result of the) multiplication between the two subexpressions annotations. A security framing is orthogonal to metric annotation, i.e., they do not affect each other. Instead, metric checks have a precise effect on annotations. As a matter of fact, we can remove a metric check by forcing its target to be annotated with the difference ($\oplus^{-1}$) between the inner annotation and the threshold of γ. Finally, a recursion is annotated with the least fix point of the function Φ that extracts the metric annotation from the inner history expression after annotating the bounded variable h.

A crucial property we want to prove on the equation rules of Table 7 is that they do not invalidate the semantics of history expressions. Such property guarantees that history expression transformations do not affect the safety property stated by theorem 1.

Property 2. For all history expressions H and H' if $H \equiv H'$ then $\forall\delta.[\![H]\!]_\delta = [\![H']\!]_\delta$

$$H \equiv \mathbf{1}\#H \quad d_1\#d_2\#H \equiv d_2\#d_1\#H \equiv d_1 \otimes d_2\#H \quad d_1\#H_1 \cdot d_2\#H_2 \equiv d_1 \otimes d_2\#(H_1 \cdot H_2)$$

$$d_1\#H_1 + d_2\#H_2 \equiv d_1 \oplus^{-1} d_2\#(H_1 + H_2) \quad d_1\#H_1 \mid d_2\#H_2 \equiv d_1 \otimes d_2\#(H_1 \mid H_2) \quad \varphi[d\#H] \equiv d\#\varphi[H]$$

$$\gamma\langle d\#H\rangle \equiv \bar{d}\#\gamma\langle H\rangle \quad \text{where } \gamma = T \geq_T d' \text{ and } \bar{d} = d \oplus^{-1} d'$$

$$\mu h.H \equiv \bar{d}\#\mu h.H' \qquad \text{where} \quad \bar{d} = \bigoplus_n{}^{-1}\Phi^n(\mathbf{0}) \text{ and } \Phi(d) = d' \Leftrightarrow \begin{cases} H[^{d\#h}/_h] \equiv d'\#H' \\ \wedge \\ d'\#H' \text{ is in MNF} \end{cases}$$

Table 7: Equational rules.

Example 7. Having in mind that $\oplus^{-1}$ is *max* for **Risk** , consider the history expression H_2 of Example 5

$$H_2 = (0\#\texttt{search\_flight\_for}(\texttt{AIRPORT})) \cdot (0\#\texttt{reserve}(\texttt{FLIGHT\_No}) + 15\#\texttt{reserve}(\texttt{NO\_FLIGHT}))$$
$$H_2 \equiv 0 \otimes (0 \oplus^{-1} 15)\#(\texttt{search\_flight\_for}(\texttt{AIRPORT}) \cdot (\texttt{reserve}(\texttt{FLIGHT\_No}) + \texttt{reserve}(\texttt{NO\_FLIGHT})))$$

Note, that the right side of the previous equivalence is in MNF. According to the operations of the semiring **Risk** , the resulting annotation value is 15. □

Example 8. We write the MNF of the history expressions of Example 5. For brevity, we write $H_i \equiv d_i\#H_i'$ to emphasise the metric annotation of the MNF without showing the structure of H_i'.

$$H_1 \equiv 20\#H_1' \qquad H_2 \equiv 15\#H_2' \qquad H_3 \equiv 25\#H_3' \qquad H_4 \equiv 15\#H_4' \qquad H_5 \equiv 40\#H_5'$$
$$H_6 \equiv 50\#H_6' \qquad H_7 \equiv 28\#H_7' \qquad H_8 \equiv 25\#H_8' \qquad H_9 \equiv 1\#H_9' \qquad H_{10} \equiv 0\#H_{10}'$$

□

Intuitively, Example 8 shows that every history expression appearing in our working example has an equivalent MNF. In general, we know that all the history expressions can be reduced to a corresponding MNF as stated by the following property.

Property 3. For each history expression H there exists H' such that $H \equiv H'$ and H' is in MNF.

The last property we show is *metric safety*, which characterizes the most important quality of the metric annotations we generate.

Theorem 2. *If $\Gamma, H \vdash e : \tau$ and $H \equiv \bar{d}\#H'$ such that $\bar{d}\#H'$ is in MNF, then for each execution $\langle \eta, d, e\rangle \rightarrow_\pi^* \langle \eta', d', e'\rangle$ holds that $d' \leq_T d \otimes \bar{d}$.*

Similarly to type safety, this theorem guarantees that metric annotations produced by our equational theory provide an upper bound to the metric values generated by the execution of a term. As each of them has a corresponding MNF, this theorem can be universally applied to any history expression.

5.2 Discussion

During the presentation we have shown how our formalism can be applied to the modelling of complex business processes. In this part of the article we describe how the proposed theory can be applied to the verification and analysis of the security properties of web services.

Basically, our proposal offers facilities that can be applied to all the stages of service design, implementation and execution. Statically, service designers can write their policies on execution histories and security metrics. Then, developers apply the scope of the policies to the service implementation. Finally, each service runs with proper checks controlling that the execution complies with the specification.

These steps suffice to carry out the analysis of possible configurations of a complex abstract business process. The goal is to check whether the possible configurations satisfy desired policies. This information is

required in order to decide if we can avoid run-time controls. Naturally, if a configuration satisfies a policy or the worst possible metric value is better than a threshold, there is no need for an additional control.

Note, that we assume that the declared policies/metrics for a specific service are genuine and the services are typed by a trusted type and effect system (implementation). Although this assumption is not true in general, here we focussed on the considered problem, i.e., aggregation of metrics and check of composite properties.

In order to check that a certain business process satisfies properties or has sufficiently good metric value the analyst starts for a λ^{req} implementation of an abstract workflow, as it is shown in Example 2. Then, we assume the service repository and c*-semirings for considered metrics to be defined similar to Examples 1 and 3. The next step is to type the service implementation similar to Example 5 and 6. Finally, we aggregate metrics annotations, as it is done in Example 7. During this process, several analysis on the validity of history expressions can be carried out in order to prevent illegal service compositions. For a description of these techniques we refer the interested reader to [4, 12].

Example 9. We use the history expressions in MNF shown in Example 8 to compute the MNF of H_B. Considering the history expression appearing in Figure 5, we can replace every instance of H_i with the corresponding MNF $d_i \# H_i'$. Then we obtain the following equivalences.

$$H_B \equiv (\gamma\langle H_F\rangle \mid \gamma\langle H_H\rangle)\cdot\gamma\langle H_S\rangle$$

$$H_F \equiv \Big((20\#H_1' + 15\#H_2')\cdot\big((28\#H_7' + 25\#H_8') + ((25\#H_3' + 15\#H_4')\cdot(28\#H_7' + 25\#H_8'))\big)\Big)$$

$$H_H \equiv \Big((40\#H_5' + 50\#H_6')\cdot(28\#H_7' + 25\#H_8')\Big) \qquad H_S \equiv \mu h.((1\#H_9' + 0\#H_{10}')\cdot h + \varepsilon)$$

Applying the rules of Table 7, we can reduce to the following history expression.

$$H_B \equiv (\gamma\langle 73\#H_F'\rangle \mid \gamma\langle 78\#H_H'\rangle)\cdot\gamma\langle\infty\#H_S'\rangle$$

Recalling that $\gamma = \mathbf{Risk} \leq 75$ we conclude with the equivalences below.

$$(\gamma\langle 73\#H_F'\rangle \mid \gamma\langle 78\#H_H'\rangle)\cdot\gamma\langle\infty\#H_S'\rangle \equiv (73\#\gamma\langle H_F'\rangle \mid 75\#\gamma\langle H_H'\rangle)\cdot 75\#\gamma\langle H_S'\rangle \equiv 223\#((\gamma\langle H_F'\rangle \mid \gamma\langle H_H'\rangle)\cdot\gamma\langle H_S'\rangle)$$

Interestingly, we note that, among the three instances of γ, only the first one applies to a history expression satisfying the restriction, i.e., $73 \in \gamma$. We cannot say the same for the other two instances. However, our semantics for metric framing forces the execution of all the parts of the service to respect risk constraints. In this way, even though some parts of the service are labelled with ∞, the overall risk is a finite value, i.e., 223.

Since the last two instances fail the restriction the dynamic analysis is required. Note, that the hotel reservation part of the process may use services H_5 and H_8 with the overall risk level $65 < 75$. Therefore, during the execution we guard the second and the third instances to guarantee the low risk level values. There is no need to guard the first instance, since it satisfies the restriction in any case. Imagine, that during the execution H_6 service has been selected. Before executing the next step the guard must check the resulting value, using the same rules as for the static analysis. In case H_8 is selected the execution is allowed ($75 \leq 75$). Otherwise, if H_7 is chosen the restriction fails ($78 > 75$) and the execution is halted (or another action is performed, e.g., a report about the failure is sent to the customer and provider).

□

6 Related work

Outsourcing processing of sensitive data to external parties requires some assurances, that the data will be well protected while processed and transmitted. Unsurprisingly, several authors claimed that security requirements must be included into the agreement between service customer and service provider [15, 16]. Our work extends the existing state of the art with a unified approach for checking security properties and security metrics of complex business processes which appear as statements in such agreements.

Many authors proposed formal languages for specifying and verifying agreements, also called *contracts*, between a service provider and a customer. Padovani [21] proposes a language for defining service contracts and presents a theory for the automatic generation of service orchestrators. Subcontract relations are used to find a matching between the contract offered by a service and the requirements of its clients. Similarly, Bravetti and Zavattaro [8] present a language for the specification of service contracts. Their contracts have a process algebra-based semantics and allow for the specification of composed services. Contract composition can be verified to guarantee that the interaction of a group of services does not violate the specifications. Even though these works do not focus on security analysis, their contracts can be adapted to model security requirements.

Martinelli and Matteucci [17] presented a framework for the synthesis of a secure orchestrator, i.e., an agent which drives the interaction between two services guaranteeing that a certain security policy is respected. Although, the proposals described above use contracts for the specification and analysis of history-based [1] service properties, none of them allows for the definition of security metrics and restrictions on them.

In order to check whether a complex business process satisfies some quantitative requirements aggregation of security metric values for atomic services is required. For example, Cheng et. al. [10] aggregated downtime metric, considering business process like a simple set of activities, i.e., regardless the operational flow.

In contrast, Jaeger et. al. [19] have shown that some metrics could be aggregated differently depending on the structural activity used for joining the atomic services. In this work all metrics were considered separately. Moreover, the author did not considered security metrics. Yu et. al. [24, 23] applied the idea of Jeager at. al. for selection of the best process among several alternatives. The authors defined aggregation functions for several metrics and aimed at selection of the best alternative which satisfies the constraints specified in the agreement. First the authors defined a utility function and proposed to solve a 0-1 multi-dimension multi-choice knapsack problem (MMKP) only for a sequential order [23]. Solutions for a general workflow were proposed later [24].

Massacci and Yautsiukhin [18] proposed a method and an algorithm for aggregation of security metrics. The authors also solved the problem of selection the best (i.e., more secure) alternative, though a wider range of metrics were considered (these metrics cannot be used in classical algorithms for finding the shortest path). The method was extended for checking several metrics at the same time using Pareto optimality strategy [14].

In our work we do not have a goal to select the business process which has the best metric value. Moreover, we assume that some processes which do have a value worse than desired may still satisfy the policy if a more secure execution path is selected for a specific invocation. Therefore, our proposal allows making a decision at design time and supporting control at run-time.

7 Conclusion

In this paper we presented a novel approach for dealing with the analysis and verification of both security and metric requirements of web services. Our system is developed on existing solutions for modelling security and metric-based requirements. The result is a unified framework for (*i*) the definition and application of security and metric policies within service implementation, (*ii*) the automatic extraction of history expressions carrying metric annotations and (*iii*) the computation (through an equational theory) of metric values which safely predict the expected behaviour of services. Our proposal requested a new type and effect system, extending existing approaches, to be defined. Interestingly, we found that adding metric annotations does not invalidate the type safety property, i.e., annotations are orthogonal to the history expressions.

The present work is a first step toward a complete model for the specification and verification of quantitative and qualitative, non functional requirements for web services. Further effort is requested in order to generalise our approach. In particular, we aim at defining a procedure for generating orchestration plans starting from the history expressions produced by our type and effect system. Such method has been presented in [4] for metric-free history expressions and we believe that similar results can be extended to our proposal. Another limitation of the current model is our static description of metric value for the events. Even though we think that assigning metric values to events is a reasonable way to model the actual behaviour of services, it is not always correct to assume these values to keep unchanged in time. Indeed, many metrics aim at modelling dynamic evolution of some property, e.g., reputation or number of system failures, which we cannot model with our approach.

References

[1] Martín Abadi & Cédric Fournet (2003): *Access Control Based on Execution History*. In: *Proceedings of the Network and Distributed System Security Symposium, NDSS 2003, San Diego, California, USA*, The Internet Society.

[2] Massimo Bartoletti (2009): *Usage Automata*. In: *Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security*, *LNCS* 5511, Springer, pp. 52–69. doi:10.1007/978-3-642-03459-6_4.

[3] Massimo Bartoletti, Gabriele Costa & Roberto Zunino (2009): *Jalapa: Securing Java with Local Policies*. *Electronic Notes in Theoretical Computer Science* 253(5), pp. 145–151, doi:10.1016/j.entcs.2009.11.020.

[4] Massimo Bartoletti, Pierpaolo Degano & Gian Luigi Ferrari (2009): *Planning and verifying service composition*. *Journal of Computer Security (JCS)* 17(5), pp. 799–837. doi:10.3233/JCS-2009-0357

[5] Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari & Roberto Zunino (2007): *Types and Effects for Resource Usage Analysis*. In: *Proc. of FOSSACS-07*, pp. 32–47. doi:10.1007/978-3-540-71389-0_4

[6] Stefano Bistarelli, Ugo Montanari & Francesca Rossi (1997): *Semiring-based constraint satisfaction and optimization*. *Journal of the ACM* 44, pp. 201–236. doi:10.1145/256303.256306

[7] Mario Bravetti, Ivan Lanese & Gianluigi Zavattaro (2008): *Contract-Driven Implementation of Choreographies*. In: *Proc. of TGC-08*, pp. 1–18. doi:10.1007/978-3-642-00945-7_1

[8] Mario Bravetti & Gianluigi Zavattaro (2007): *Towards a unifying theory for choreography conformance and contract compliance*. In: *In Pre-proc. CS-07*, Springer, pp. 34–50. doi:10.1007/978-3-540-77351-1_4

[9] Giuseppe Castagna, Nils Gesbert & Luca Padovani (2008): *A theory of contracts for web services*. *SIGPLAN Notices* 43, pp. 261–272. doi:10.1145/1328438.1328471

[10] Feng Cheng, David Gamarnik, Nitin Jengte, Wanli Min & Bala Ramachandran (2005): *Modelling Operational Risks in Business Process*. Technical Report RC23872, IBM.

[11] E. M. Clarke, E. A. Emerson & A. P. Sistla (1986): *Automatic verification of finite-state concurrent systems using temporal logic specifications*. *TOPLAS* 8, pp. 244–263. doi:10.1145/5397.5399

[12] Gabriele Costa, Pierpaolo Degano & Fabio Martinelli (2010): *Modular Plans for Secure Service Composition*. In *Proc. of ARSPA-WITS-10*, *LCNS* 6186. doi:10.1007/978-3-642-16074-5_4

[13] Gabriele Costa, Pierpaolo Degano & Fabio Martinelli (2011): *Secure service orchestration in open networks*. *Journal of Systems Architecture - Embedded Systems Design* 57(3), pp. 231–239. doi:10.1016/j.sysarc.2010.09.001

[14] Frank Innerhofer-Oberperfler, Fabio Massacci & Artsiom Yautsiukhin (2008): *Pareto-Optimal Architecture according to Assurance Indicators*. In: *Proceedings of the 13th Nordic Workshop on Secure IT Systems*.

[15] Cynthia Irvine & Timothy Levin (2000): *Quality of security service*. In: *Proceedings of the 2000 Workshop on New security paradigms*, ACM, New York, NY, USA, pp. 91–99. doi:10.1145/366173.366195

[16] Günter Karjoth, Birgit Pfitzmann, Matthias Schunter & Michael Waidner (2005): *Service-oriented Assurance - Comprehensive Security by Explicit Assurances*. In: *Proc. of QoP-05*. doi:http://dx.doi.org/10.1007/978-0-387-36584-8_2

[17] Fabio Martinelli & Ilaria Matteucci (2007): *Synthesis of Web Services Orchestrators in a Timed Setting*. In: *Proc. of WS-FM-07*, pp. 124–138. doi:10.1007/978-3-540-79230-7_9

[18] Fabio Massacci & Artsiom Yautsiukhin (2007): *An Algorithm for the Appraisal of Assurance Indicators for Complex Business Processe*. In: *Proc. of QoP-07*, ACM. doi:10.1145/1314257.1314265

[19] G. Rojec-Goldmann M.C. Jaeger & G. Mühl. (2005): *QoS Aggregation in Web Service Compositions*. In: *Proc. of EEE-05)*. doi:dx.doi.org/10.1109/EEE.2005.110

[20] Hanne Riis Nielson & Fleming Nielson (2007): *A flow-sensitive analysis of privacy properties*. In: *Proceedings of the 20-th IEEE Computer Security Foundations Symposium*, pp. 249 –264. doi:10.1109/CSF.2007.4

[21] Luca Padovani (2008): *Contract-Directed Synthesis of Simple Orchestrators*. In: *Proceedings of the 19th international conference on Concurrency Theory*, Springer-Verlag, pp. 131–146. doi:10.1007/978-3-540-85361-9_13

[22] S. Rossi & D. Macedonio (2009): *Information flow security for service compositions*. In: *Proc. of ICUMT-09*, pp. 1 –8. doi:10.1109/ICUMT.2009.5345455

[23] Tao Yu & Kwei-Jay Lin (2005): *A Broker-Based Framework for QoS-Aware Web Service Composition*. In: *Proc. of EEE-05)*. doi:10.1109/EEE.2005.1

[24] Tao Yu, Yue Zhang & Kwei-Jay Lin (2007): *Efficient algorithms for Web services selection with end-to-end QoS constraints*. *ACM Transactions on the Web* 1. doi:10.1145/1232722.1232728