Choreography Rehearsal [⋆]

Chiara Bodei[1] and Gian Luigi Ferrari[1]

Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo, 3, I-56127, Pisa, Italy -
{chiara,giangi}@di.unipi.it

Abstract. We propose a methodology for statically predicting the possible interaction patterns of services within a given choreography. We focus on choreographies exploiting the event notification paradigm to manage service interactions. Control Flow Analysis techniques statically approximate which events can be delivered to match the choreography constraints and how the multicast groups can be optimised to handle event notification within the service choreography.

1 Introduction

The ability of supporting programmable coordination policies of heterogeneous services is a key element in the success of the *Service Oriented Computing* (SOC) paradigm. Two different approaches are usually adopted to assemble services: *orchestration* and *choreography*. In the service orchestration, an intermediate entity, the orchestrator, arranges service activities according to the given business process. The service choreography, instead, involves all parties and their associated interactions providing a global view of the system. Relevant standard technologies are the Business Process Execution Language (BPEL) [23], for the orchestration, and Web Service Choreography Description Language (WS-CDL) [24], for the choreography. Notably, the orchestration-choreography issues have led to the development of a variety of foundational models (see e.g. [19, 12, 2, 7, 18, 10] to cite a few). We refer to the surveys in [9, 22] for an analysis of the approaches.

In [15, 21] a middleware, called Java Signal Core Layer (JSCL), supporting the design and implementation of service coordination policies has been introduced. The middleware consists of a set of API for assembling services by exploiting the *event notification* paradigm. A distinguished feature of JSCL consists of the strict interplay among formal semantic foundations, implementation pragmatics and experimental evaluation of the resulting programming mechanisms. More precisely, the programming facilities available in JSCL have been semantically motivated. At the abstract level, the middleware takes the form of the Signal Calculus (SC) [17]. The SC calculus is an asynchronous process calculus with explicit primitives to deal with (multicast) event notification and service

[⋆] Research supported by the EU within the FET-GC II Integrated Project IST-2005-016004 SENSORIA and by the Italian PRIN Project "SOFT".

distribution. The SC-JSCL framework allows one to specify and program services coordination policies (orchestration and choreography) relying on multicast notification only. Moreover, it features sessions as a mechanism to synchronise behaviours of distributed and independent services. Remarkably, the middleware does not assume any centralized mechanism for publishing, subscribing and notifying events. Hence, SC and JSCL have to be properly regarded as a foundational framework and its programming counterpart for specifying, verifying and programming coordination policies of distributed services.

The JSCL framework has been also equipped with a *model driven* development methodology [13, 14, 16]. The methodology exploits a suitable choreography model that takes the form of a process calculus, called Network Coordination Policies (NCP). The two calculi (SC and NCP) lay at two different levels of abstraction. The former is tailored to support the (formal) design of services, the latter is the specification language to declare the coordination policies. Policies are processes specifying service behaviour as seen by an observer standing from a global point of view, hence capable of observing the interactions that are expected to happen, and how these are interleaved. Indeed, certain features can be described at both levels: the NCP specification declares what is expected from the service network infrastructure, while the SC design specifies how to implement it. The gap between the local and global abstraction levels has been formally filled in [13, 17]. It has been proved that for each SC design, there exists an NCP choreography that reflects all the properties of the design. The conformance of an SC design with respect to an NCP specification is formally proved by checking weak asynchronous bisimilarity [1] between them. This notion of conformance has the main benefit of supporting the development of systems in a model driven development fashion. The designer can define a suitable chain of SC models that implement the choreography: each model is obtained by refinement steps that add more details. The conformance of each model with respect to the NCP specification provides the formal machinery to choose the required level of abstraction, so that one can focus on coordination of services, without considering the implementation details, or focus on service design, just trying to match the abstract policies.

The present paper aims at contributing to this line of research. Our long-term goal is to equip the JCSL middleware with semantic-based toolkits supporting its design, development, and deployment. In particular, this paper develops *static* reasoning techniques for the JSCL middleware. We use a specific static technique, *Control Flow Analysis*, based on *Flow Logic* [20]. This kind of static analysis provides a variety of automatic and decidable methods and tools for analysing properties of computing system.

Our first contribution is the definition of a Control Flow Analysis for the SC process calculus, that it is shown to be sound. For simplicity, the analysis is introduced in two stages: first it is developed for a basic fragment of the calculus considering flows and multicast. In the second stage, session management is taken into account. This analysis *safely* approximates the behaviour of an SC design, statically predicting the possible structure of event notifications.

This information offers a basis for studying dynamic properties, by suitably handling the approximation the static analysis constructs. We have indeed an over-approximation of the *exact* behaviour of a system. This means that all those interactions that the analysis *does not* include will *never* happen, while all the interactions that the analysis *does* include *can* happen, i.e. they are only possible. The result of the analysis can therefore be used to predict at compile time all the *possible* event flows emanating from a certain service. Implicitly, this amounts to providing the maximal flow of an event notification and, consequently, an upper bound on the structure of the multicast group implementing the notification. Hence, the analysis provides formal basis to optimise the management of multicast groups of the JSCL run-time.

Our second contribution consists in the development of a Control Flow Analysis for the NCP calculus. The analysis, that computes a safe over-approximation of event interactions, can be used to verify if certain choreography constraints are satisfied. We can assert that events of a certain type have not to be captured by a service and then we can statically verify, by inspecting the analysis, that this assertion is not violated. In other words, the analysis acts in a *descriptive* fashion: if no property violation is statically found then no violation of the property can occur at run-time. However, within the NCP choreography model, the analysis can also be exploited in a *prescriptive* fashion. Intuitively, the analysis can suggest how to instrument the SC design to avoid occurrences of a property violation. For instance, the constraints on event handling mentioned above can be satisfied, by instrumenting the multicast group with a filter discarding the events referring to the unauthorized event.

Our static machinery has been applied to several process calculi, amongst which π-calculus (e.g. [5]) and LySa [4] to establish security properties. In particular, the mixed descriptive/prescriptive approach offered by Control Flow Analysis has been introduced in [3] to deal with type flaws in crypto-protocols.

Plan of the Paper. In Section 2, we present the simplest version of the SC calculus focussing on multicast notification. In Section 3, we completely introduce the Control Flow Analysis for this version of the calculus. This analysis is extended in Section 4 to manage the SC notion of session. The NCP calculus and its Control Flow Analysis are described in Section 5. In Section 6, we show how consistency between a network of SC components and the global coordination policy expressed by NCP specifications is reflected by the correspondence between the analysis results. For lack of space, all the proofs are omitted, but are reported in the extended version of the paper [6].

2 The Calculus

The Signal Calculus (SC) [17], is a process calculus specifically designed to describe coordination of services distributed over a network. The calculus is based on the event notification paradigm. SC building blocks are called *components*, which interact by issuing/reacting to *events*. A component contains a behaviour, for instance, a "simple" service, interacting through an asynchronous signal passing mechanism. Each component stores information about the collection of components that must be notified whenever events are issued (*event flow*). When an event is raised by a component, several envelopes are generated to notify all components in the flow (*multicast notification*). Each envelope, also called *signal*, contains the event itself and the address of the target component. Each component owns a set of signal handlers associated to type event. Usually, in the event notification literature, the type of an event is called *topic*. Signal handlers, called *reactions*, are responsible for the management of the reception of an event notification. Indeed, the reception of a signal acts like a trigger that activates the execution of a new behaviour, described by the compatible reaction within the component.

The component *interface* is defined by its *reactions* and *flows*. The language primitives allow one to *dynamically* modify the component interfaces topology of the coordination, by adding new flows and reactions. Finally, components are structured to build a *network* of services. A network provides the facility to transport signals containing the events exchanged among components.

Let $\mathcal{A} \ni a,\ b,\ c...$ be a finite set of components names, and $\mathcal{T} \ni \tau_1, ..., \tau_k$ be a finite set of topics. We use $\tilde{a}$ to denote a set of names $a_1, ..., a_n$. A *component* is written as $a[B]_F^R$ and represents the service uniquely identified by the name a, i.e. its public address. Each component has internal behaviour B, reaction R and flow F.

$N ::=$		*networks*
	$\mid\ \mathbf{0}$	empty network
	$\mid\ N \mid\mid N$	parallel composition
	$\mid\ a[B]_F^R$	component
	$\mid\ \langle\tau\rangle@a$	signal envelope
$R ::=$		*reactions*
	$\mid\ 0$	empty reaction
	$\mid\ \tau \gtrdot B$	unit reaction
	$\mid\ R \mid R$	parallel composition

$B ::=$		*behaviour*
	$\mid\ 0$	empty behaviour
	$\mid\ \mathsf{rupd}(R); B$	reaction update
	$\mid\ \mathsf{fupd}(F); B$	flow update
	$\mid\ \mathsf{out}\langle\tau\rangle; B$	event emission
	$\mid\ \epsilon; B$	internal behaviuor
	$\mid\ B \mid B$	parallel composition
$F ::=$		*flows*
	$\mid\ 0$	empty flow
	$\mid\ \tau \rightsquigarrow \tilde{a}$	unit flow
	$\mid\ F \mid F$	parallel composition

Fig. 1. Syntax of SC, version 1

The syntax of SC is presented in Fig. 1. A reaction R is a multiset, possibly empty, of unit reactions. A unit reaction $\tau \gtrdot B$ triggers the execution of the behaviour B upon reception of a signal tagged by the topic τ. A flow F is a set, possibly empty, of unit flows. A unit flow $\tau \rightsquigarrow \tilde{a}$ describes the set of component names $\tilde{a}$ where raised events having τ as topic have to be delivered. We define $F \downarrow_\tau$ as the set of $\tilde{b}$ such that $\tau \rightsquigarrow \tilde{b}$ occurs in F.

$$\text{(SKIP)}\quad \frac{}{a[\epsilon; B_1 \mid B_2]^R_F \to a[B_1 \mid B_2]^R_F}$$

$$\text{(RUPD)}\quad \frac{}{a[\mathsf{rupd}(R_1); B_1 \mid B_2]^R_F \to a[B_1 \mid B_2]^{R|R_1}_F}$$

$$\text{(FUPD)}\quad \frac{}{a[\mathsf{fupd}(F_1); B_1 \mid B_2]^R_F \to a[B_1 \mid B_2]^R_{F|F_1}}$$

$$\text{(OUT)}\quad \frac{F \downarrow_\tau = \tilde{b}}{a[\mathsf{out}\langle\tau\rangle; B_1 \mid B_2]^R_F \to a[B_1 \mid B_2]^R_F || \Pi_{b_i \in \tilde{b}} \langle\tau\rangle@b_i}$$

$$\text{(IN)}\quad \frac{}{\langle\tau\rangle@a || a[B_1]^{R|\tau \gg B_2}_F \to a[B_1 | B_2]^{R|\tau \gg B_2}_F}$$

$$\text{(PAR)}\quad \frac{N \to N_1}{N || N_2 \to N_1 || N_2}$$

$$\text{(STRUCT)}\quad \frac{N \equiv N_1 \to N_2 \equiv N_3}{N \to N_3}$$

Fig. 2. Reduction Semantics of SC

A behaviour B is a multiset of simple behaviours. A reaction update $\mathsf{rupd}(R); B$ extends the reaction part of the component interface. Similarly, the flow update $\mathsf{fupd}(F); B$ extends the component flows. The asynchronous event emission $\mathsf{out}\langle\tau\rangle; B$ first spawns into the network a set of envelopes containing the event, one for each component name declared in the flow having topic τ, and then activates B. The behaviour $\epsilon; B$ abstracts from the internal activities performed by the component (at the end of its execution, the component activates the continuation B). Finally, the inactive behaviour $\mathbf{0}$ and the parallel composition $B|B$ have the standard meanings. Reactions, flows and behaviours are defined up-to a structural congruence ($\equiv$). Indeed we assume that $(F, |, 0)$, $(R, |, 0)$ and $(B, |, 0)$ are commutative monoids, i.e. parallel composition is commutative, associative and 0 is the identity. Moreover, we have that $\tau \rightsquigarrow \tilde{a} | \tau \rightsquigarrow \tilde{b} \equiv \tau \rightsquigarrow \tilde{a} \cup \tilde{b}$. We omit the trailing occurrences of 0.

Networks (N) describe the distribution of components and carry signals exchanged among them. The signal envelope $\langle s\rangle@a$ describes a message containing the topic τ, whose target component is named a. The empty network $\mathbf{0}$ and the parallel composition have the standard meanings. In the following, we will use $\prod_{b_i \in \tilde{b}} \langle\tau\rangle@b_i$, with $\tilde{b}$ a finite set of component names, to represent the parallel composition of messages having topic τ.

The operational semantics is defined in the reduction style and states how components, at each step, communicate and update their interfaces. Reduction rules of SC are given in Fig. 2. Rule (SKIP) describes the execution of an internal action, i.e. an action that has no side effect on the system. Rule (RUPD) extends the component reactions with a further unit reaction (the parameter of the primitive). Rule (FUPD) extends the component flows with a unit flow. Rule (OUT) first takes the set of component names $\tilde{a}$ that are linked to the component for the topic τ and then spawns into the network an envelope for each component name in the set. Rule (IN) allows a signal envelope to react with the component whose name is specified inside the envelope. Note that signal emission rule (OUT) and signal receiving rule (IN) do not consume, respectively, the flow and the reaction of the component, i.e. flows and reactions are *persistent*. Finally, rules (STRUCT) and (PAR) are standard.

Example 1. Multicast Notification. Let us consider a component s that requires a set of resources to provide a certain functionality. This component is exploited by several clients c_i, with $i = 1, .., n$, to achieve a common goal. All clients collaborate to the activation of the service supplied by s, providing the required resources. The process is summarized as follows:

$$\begin{array}{lcl} N & \overset{def}{=} & s[out\langle\tau_r\rangle]^{\tau_0 \gg \mathtt{out}\langle\tau_r\rangle | \tau_0 \gg B}_{t_r \rightsquigarrow \{c_1,c_2,c_3\}} || C_1 || C_2 || C_3 \\ C_i & \overset{def}{=} & c_i[0]^{t_r \gg \mathtt{out}\langle\tau_0\rangle | \tau_r \gg 0}_{\tau_0 \rightsquigarrow \{s\}} \end{array}$$

Initially, the service S issues an event to notify its demand of resources.

$$s\,[\mathtt{out}\langle\tau_r\rangle]^{\tau_o \gg \mathtt{out}\langle\tau_r\rangle \ | \ \tau_o \gg B}_{\tau_r \rightsquigarrow \{c_1,c_2,c_3\}} \rightarrow s\,[0]^{\tau_o \gg \mathtt{out}\langle\tau_r\rangle \ | \ \tau_o \gg B}_{\tau_r \rightsquigarrow \{c_1,c_2,c_3\}} \parallel \langle\tau_r\rangle @ c_1 \parallel \langle\tau_r\rangle @ c_2 \parallel \langle\tau_r\rangle @ c_3$$

Upon the reception of a resource request, a client non-deterministically activates one of its two reactions.

$$c_i\,[0]^{\tau_r \gg \mathtt{out}\langle\tau_o\rangle \ | \ \tau_r \gg 0}_{\tau_o \rightsquigarrow \{s\}} \parallel \langle\tau_r\rangle @ c_i \rightarrow c_i\,[\mathtt{out}\langle\tau_o\rangle]^{\tau_r \gg \mathtt{out}\langle\tau_o\rangle \ | \ \tau_r \gg 0}_{\tau_o \rightsquigarrow \{s\}}$$

The client raises events τ_o to notify their agreement to provide a resource.

$$c_i\,[\mathtt{out}\langle\tau_o\rangle]^{\tau_r \gg \mathtt{out}\langle\tau_o\rangle \ | \ \tau_r \gg 0}_{\tau_o \rightsquigarrow \{s\}} \rightarrow c_i\,[0]^{\tau_r \gg \mathtt{out}\langle\tau_o\rangle \ | \ \tau_r \gg 0}_{\tau_o \rightsquigarrow \{s\}} \parallel \langle\tau_o\rangle @ s$$

Upon the reception of a resource bid, the service non-deterministically activates one of its two reactions.

$$s\,[0]^{\tau_o \gg \mathtt{out}\langle\tau_r\rangle \ | \ \tau_o \gg B}_{\tau_r \rightsquigarrow \{c_1,c_2,c_3\}} \parallel \langle\tau_o\rangle @ s \rightarrow s\,[B]^{\tau_o \gg \mathtt{out}\langle\tau_r\rangle \ | \ \tau_o \gg B}_{\tau_r \rightsquigarrow \{c_1,c_2,c_3\}}$$

3 The Control Flow Analysis for SC

We now introduce the Control Flow Analysis for SC. The aim of the analysis is over-approximating all the possible behaviour of SC processes. In particular, we focus on how components communicate and update their interface. The result of analysing a network N is a tuple $(\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E})$, called *estimate* for N, that satisfies the judgements defined by the axioms and rules in the upper (lower, resp.) part of Table 1. Given a certain component a, $\mathcal{B}(a)$ gives an approximation of the possible behaviours of a; $\mathcal{R}(a)$ gives an approximation of the possible reactions of a; $\mathcal{F}(a)$ gives an approximation of the possible flows of a: and $\mathcal{E}(a)$ gives an approximation of the possible envelopes to be received by a.

To *validate* the correctness of a proposed estimate $(\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E})$ we state a set of clauses operating upon judgements for analysing processes $\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E} \models N$, defined in the flavour of Flow Logic [20].

Validation. The analysis is specified in two phases. First, we check that $(\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E})$ describes the initial process. This is done in the upper part of Table 1, where the clauses amount to a structural traversal of process syntax. The clauses rely on the auxiliary functions $\mathbf{A}_B$, $\mathbf{A}_R$, $\mathbf{A}_F$, that given a behaviuor B, reaction R or flow F, keep track of the single unit behaviour occurring in B, reaction actions in R and flows in F, respectively. Their definitions are reported at the beginning of Table 1. In the second phase, we check that $(\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E})$ also takes into account the dynamics of the process under analysis. This is expressed by the closure conditions in the lower part of Table 1 that mimic the semantics, by modelling, without exceeding the precision boundaries of the analysis, the semantic preconditions and the consequences of the possible actions. More precisely, preconditions check, in terms of $(\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E})$, for the possible presence of the redexes necessary for actions to be performed. The conclusion imposes the additional requirements on $(\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E})$, necessary to give a valid prediction of the analysed action. For instance, in the penultimate clause in Table 1, if (i) there exists an occurrence of $\mathsf{out}\langle\tau\rangle$ in $\mathcal{B}(a)$, and (ii) there exists an occurrence of (τ, b) in $\mathcal{F}(a)$, then there is a signal envelope with topic τ to be received by b, i.e. a possible `out` action is predicted.

$\mathbf{A}_B(0) = \emptyset$	
$\mathbf{A}_B(\epsilon; B) = \mathbf{A}_B(B)$	
$\mathbf{A}_B(b; B) = \{b\} \cup \mathbf{A}_B(B)$ where $b ::= \mathsf{fupd}(F) \vert \mathsf{rupd}(R) \vert \mathsf{out}\langle\tau\rangle$	
$\mathbf{A}_B(B_0 \vert B_1) = \mathbf{A}_B(B_0) \cup \mathbf{A}_B(B_1)$	
$\mathbf{A}_R(0) = \emptyset$	
$\mathbf{A}_R(\tau \gtrdot B) = \{(\tau, B)\}$	
$\mathbf{A}_R(R_0 \vert R_1) = \mathbf{A}_R(R_0) \cup \mathbf{A}_R(R_1)$	
$\mathbf{A}_F(0) = \emptyset$	
$\mathbf{A}_F(\tau \rightsquigarrow \tilde{a}) = \{(\tau, a_i) \vert a_i \in \tilde{a}\}$	
$\mathbf{A}_F(F_0 \vert F_1) = \mathbf{A}_F(F_0) \cup \mathbf{A}_F(F_1)$	
$\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E} \models \mathbf{0}$	iff $true$
$\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E} \models N_0 \vert N_1$	iff $\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E} \models N_0 \ \wedge\ \mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E} \models N_1$
$\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E} \models \langle\tau\rangle @ a$	iff $\tau \in \mathcal{E}(a)$
$\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E} \models a[B]^R_F$	iff $\mathbf{A}_B(B) \subseteq \mathcal{B}(a) \wedge \mathbf{A}_R(R) \subseteq \mathcal{R}(a) \wedge \mathbf{A}_F(F) \subseteq \mathcal{F}(a)$
$\mathsf{fupd}(F) \in \mathcal{B}(a) \Rightarrow \mathbf{A}_F(F) \subseteq \mathcal{F}(a)$	
$\mathsf{rupd}(R) \in \mathcal{B}(a) \Rightarrow \mathbf{A}_R(R) \subseteq \mathcal{R}(a)$	
$\mathsf{out}\langle\tau\rangle \in \mathcal{B}(a) \wedge (\tau, b) \in \mathcal{F}(a) \Rightarrow \tau \in \mathcal{E}(b)$	
$\tau \in \mathcal{E}(a) \wedge (\tau, B) \in \mathcal{R}(a) \Rightarrow \mathbf{A}_B(B) \subseteq \mathcal{B}(a)$	

Table 1. Analysis for SC Processes

Example 2 (Multicast notification). Back to our example, we report the main entries of the analysis in Table 2. It is possible to check that $(\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E})$ is a valid estimate, by following the two stages explained above. The analysis correctly

$\mathcal{E}(c_i) \ni \tau_r$	
$\mathcal{B}(s) \ni \mathsf{out}\langle\tau_r\rangle, \mathcal{A}_B(B)$	$\mathcal{B}(c_i) \ni \mathsf{out}\langle\tau_0\rangle$
$\mathcal{R}(s) \ni (\tau_0, \mathsf{out}\langle\tau_r\rangle), (\tau_0, B)$	$\mathcal{R}(c_i) \ni (\tau_r, \mathsf{out}\langle\tau_0\rangle), (\tau_r, 0)$
$\mathcal{F}(s) \supseteq \{(\tau_r, c_i) \| c_i \in \{c_1, c_2, c_3\}\}$	$\mathcal{F}(c_i) \ni (\tau_0, \{s\})$

Table 2. Some Entries of the Example Analysis

approximates the behaviour of N; for instance it predicts that three envelopes $\langle\tau_r\rangle@c_i$ can be spawn ($\mathcal{E}(c_i) \ni \tau_r$).

We prove that our analysis is safe w.r.t. the given semantics, i.e. a valid estimate enjoys the following subject reduction property.

Theorem 1. ***(Subject Reduction)***
If $N \rightarrow N'$ and $\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E} \models N$ then also $\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E} \models N'$.

Proof Sketch The proof is by induction on $N \rightarrow N'$.

The above result can be made more precise, by looking at the single analysis components. As an example, we just show that the analysis component $\mathcal{F}$ captures all the flows that involve the components of a network N. Clearly, similar results hold for the other components.

Theorem 2. ***(Flows $\mathcal{F}$)*** *If $\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E} \models N$ and $N \rightarrow^* N' \rightarrow N''$, such that the last transition $N' \rightarrow N''$ is derived using the rule (FUPD) on the set F in a component* a*, then $\mathbf{A}_F(F) \subseteq \mathcal{F}(a)$.*

Proof Sketch By Theorem 1, we have that $\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E} \models N'$. Therefore, the proof proceeds by induction on the transition rules used to derive $N' \rightarrow N''$.

Our Control Flow Analysis approximates the behaviour of the network under consideration. It provides a *safe over-approximation* of the *exact* behaviour of services: at least all the valid behaviours are captured. More precisely, all those interactions that the analysis does not consider as possible will *never* occur. On the other hand, the interactions deemed as possible may, or may not, occur in the actual dynamic evolution of the network. Therefore, by exploiting the analysis's soundness, we can prove several properties. As an example, we discuss a property related to the flow of a certain service. Before, we introduce some auxiliary notions. Given a network N, the set of networks reachable from N is defined as $\mathit{Reach}(N) = \{N' | N \rightarrow^* N'\}$. Let the flows emanating from a in N be defined as $F(N)(a) = \{F | a[B]_F^R \text{ occurs in } N\}$. The analysis component $\mathcal{F}$ can be used to predict, at compile time, all the *possible* flows emanating from a certain component in a network at run time, as stated by the following result.

Theorem 3. *Given a network N, including a component* a*, and an estimate $(\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E})$ such that $\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E} \models N$, we have that $\{F(N')(a) | N' \in \mathit{Reach}(N)\} \subseteq \mathcal{F}(a)$.*

Proof Sketch Immediate by Theorems 1 and 2.

$N ::=$	*networks*
$\mid$ **0**	empty network
$\mid N \| N$	parallel composition
$\mid a[B]^R_F$	component
$\mid \langle \tau_{\copyright}\tau' \rangle @a$	signal envelope

$R ::=$	*reactions*
$\mid 0$	empty reaction
$\mid R \mid R$	parallel composition
$\mid \tau_{\copyright}\tau' \geqslant B$	check reaction
$\mid \tau\lambda\tau' \geqslant B$	lambda reaction

$B ::=$	*behaviour*
$\mid 0$	empty behaviour
$\mid \mathsf{rupd}(R); B$	reaction update
$\mid \mathsf{fupd}(F); B$	flow update
$\mid \mathtt{out}\langle \tau_{\copyright}\tau' \rangle; B$	event emission
$\mid \epsilon; B$	internal behaviuor
$\mid B \mid B$	parallel composition

$F ::=$	*flows*
$\mid 0$	empty flow
$\mid \tau \rightsquigarrow \tilde{a}$	unit flow
$\mid F \mid F$	parallel composition

Fig. 3. Syntax of SC, version 2

From this static result, we can infer the maximal possible dimension that a flow emanating from a certain component in a network can reach at run time, just by computing $|\mathcal{F}(a)|$, where $|S|$ stands for the cardinality of the set S.

Note that similar static machineries can be exploited in the back-end of JCSL compiler, to optimise the code and the structure of the network interface.

4 Managing Session: a New Version of SC and a New Version of the Analysis

In the first version of SC, information associated to signals is not structured and topics cannot be created dynamically. Furthermore, the notion of session is missing: components cannot keep track of concurrent event notifications. A refined version of SC, whose syntax is presented in Fig. 3, tackles sessions management.

Events are pairs including a topic and a session identifier. The syntax of behaviors is modified by the signal emission primitive ($\mathtt{out}\langle \tau_{\copyright}\tau' \rangle$) and by the capability of generating new topics ($(\nu\tau)B$). Note that both topics and sessions are names and are freely interchangeable. As far as the reactive part is concerned, a *lambda reaction* $\tau\lambda\tau' \geqslant B$ handles all signals with topic τ, regardless of their session. In the behaviour B, τ' is bound by the lambda reaction. A *check reaction* $\tau_{\copyright}\tau' \geqslant B$ can instead handle only signals having the topic τ issued for the session τ' and does not declare bound names. The syntax of flows has been not changed. The *envelope* $\langle \tau_{\copyright}\tau' \rangle @a$ now carries both the topic τ and the session identifier τ'. For the sake of simplicity, we skip the restriction construct.

In Fig. 4, we only give the rules that are different from the ones in Fig. 2 and the new ones. Similarly, in Table 3, we just give the CFA rules that are different from the ones in Table 1. The subject reduction result stated on the previous version of SC can be easily extended to the present version.

$$
\text{(OUT)}\quad \frac{F \downarrow_{\tau} = \tilde{b}}{a[\mathsf{out}\langle \tau_{\copyright}\tau' \rangle; B_1 \mid B_2]_F^R \to a[B_1 \mid B_2]_F^R || \Pi_{b_i \in \tilde{b}} \langle \tau_{\copyright}\tau' \rangle @ b_i}
$$

$$
\text{(CHECK)}\quad \frac{}{\langle \tau_{\copyright}\tau' \rangle @ a \;||\; a[B_1]_F^{R|\tau_{\copyright}\tau' \gtrdot B_2} \to a[B_1|B_2]_F^R}
$$

$$
\text{(LAMBDA)}\quad \frac{}{\langle \tau_{\copyright}\tau' \rangle @ a \;||\; a[B_1]_F^{R|\tau\lambda\tau'' \gtrdot B_2} \to a[B_1|\{\tau'/\tau''\}B_2]_F^{R|\tau\lambda\tau'' \gtrdot B_2}}
$$

Fig. 4. Reduction Semantics of SC

$\mathbf{A}_B(b;B) = \{b\} \cup \mathbf{A}_B(B)$ where $b ::= \mathsf{fupd}(F)\mid\mathsf{rupd}(R)\mid\mathsf{out}\langle \tau_{\copyright}\tau' \rangle$ $\mathbf{A}_B((\nu\tau)B) = \mathbf{A}_B(B)$ $\mathbf{A}_R(\tau_{\copyright}\tau' \gtrdot B) = \{(\tau_{\copyright}\tau', B)\}$ $\mathbf{A}_R(\tau\lambda\tau' \gtrdot B) = \{(\tau\lambda\tau', B)\}$
$\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E} \models \langle \tau_{\copyright}\tau' \rangle @ a$ iff $\tau_{\copyright}\tau' \in \mathcal{E}(a)$
$\mathsf{out}\langle \tau_{\copyright}\tau' \rangle \in \mathcal{B}(a) \wedge (\tau, b) \in \mathcal{F}(a) \Rightarrow \tau_{\copyright}\tau' \in \mathcal{E}(b)$ $\tau_{\copyright}\tau' \in \mathcal{E}(a) \wedge (\tau_{\copyright}\tau', B) \in \mathcal{R}(a) \Rightarrow \mathbf{A}_B(B) \subseteq \mathcal{B}(a)$ $\tau_{\copyright}\tau' \in \mathcal{E}(a) \wedge (\tau\lambda\tau'', B) \in \mathcal{R}(a) \Rightarrow \mathbf{A}_B(\{\tau'/\tau''\}B) \subseteq \mathcal{B}(a)$

Table 3. Analysis for SC Processes, version 2

5 The Network Coordination Policies Calculus and its Analysis

The Calculus We now conclude the presentation of JCSL framework, by introducing the choreography model. This takes the form of an asynchronous calculus, called Network Coordination Policies (NCP) [17]. Intuitively, SC is used to support the design of services, while NCP is the specification language used to declare the coordination policies. Policies are processes that represent the behavior as observed from a *global* point of view, i.e. by observing all the public interactions on the network infrastructure. Hence, an NCP process describes the interactions that are expected to happen and how these are interleaved. The NCP specification declares *what* is expected from the service network infrastructure, whereas the SC design specifies *how* to implement it. NCP adheres to the multicast notification mechanism of SC, however, while SC exploits the notion of flows, NCP manages this information by a global point of view, introducing the notion of *network topologies*. In other words, a network topology represents the flows of all components involved by the coordination.

A NCP specification consists of two entities: a policy and a network topology. The former describes the actions that should be performed by components, while the latter describes the component inter-connection. A *network topology* is a structure $G = (V, E)$, where $V \subseteq A$ consists of the restricted component names of the network and $E \subseteq A \times \mathcal{T} \times A$ are the flow connections among components: $(a, \tau, b) \in E$ represents a a with a flow towards b for signal of topic τ. Note that G induces a directed labelled graph, called *topic-graph*. We will use the following

$P ::=$	*coordination policies*
$\mid \sum_{i \in I} p_i@a_i.P_i$	non-det. guarded choice
$\mid \overline{\tau}\tau'@a.P$	policy
$\mid \langle \tau_{\circledcirc}\tau' \rangle@a$	signal envelope
$\mid \mathsf{fupd}(F)@a.P$	flow update
$\mid \iota.P$	internal activity
$\mid P \| P$	parallel composition

$p ::=$	
$\mid \tau(\tau')$	lambda input
$\mid \tau\tau'$	check input

Fig. 5. Syntax of NCP

$\alpha ::=$	*actions*
$\mid \epsilon$	silent action
$\mid \tau\tau'@a$	free reaction activation
$\mid (\tau\tau'@a)$	message reception
$\mid \langle \tau_{\circledcirc}\tau' \rangle@a$	bound event notification

Fig. 6. NCP actions

auxiliary notations: (i) the *flows emanating from a in G*, $G(a) = \{(\tau, b) | (a, \tau, b) \in E\}$; (ii) the *topic-graph of τ in G*, $G(\tau) = \{(a, \tau', b) \in E | \tau' = \tau\}$; (iii) the *flow projection of τ for a in G*, $G(\tau, a) = \{b | (\tau, b) \in G(a)\}$.

The syntax of NCP is presented in Fig. 5. For the sake of simplicity, we consider the restriction-free fragment of NCP. As a consequence in the semantics, we will skip the rules (OPEN), (CLOSE) and (NEW). Let G be an NCP topology and P an NCP policy, then the pair $\langle G; P \rangle$ is called *NPC state*. NPC states represent the specifications of a system.

An NCP process is called a *coordination policy*. Non-deterministic (guarded) choice is denoted as $\sum_{i \in I} p_i@a_i.P_i$; a policy $p@a.P$ represents an action p executed by the component a with continuation P; prefix $\tau(\tau')$ allows to receive on τ and is called *lambda input* since it corresponds to SC lambda reactions; $\tau\tau'$ allows to receive signals having topic τ and session τ' and is therefore called *check input*. Since a lambda input can handle events regardless their sessions, the name τ' represents a binder for the received session identifier. The policy $\overline{\tau}\tau'$ raises an event on session τ' with topic τ. The component delivers the corresponding notifications to all services that are subscribed on the topic τ. The *envelope* $\langle \tau_{\circledcirc}\tau' \rangle@a$ represents a pending message/notification on the network towards a. Notice that only the target of the envelope is declared. Also in NCP, the emission of an event and its reception are performed in two phases. Initially, the emitter spawns into the network the proper envelopes, according with the actual network topology. Subsequently, a subscriber can react to the received envelope. The policy $\mathsf{fupd}(F)$ adds F to the flows departing from a. Prefix $\iota.P$ represents the execution of an internal activity before the execution of P. Finally, coordination policies can be composed in parallel.

The operational semantics of NCP is specified by the labelled transition system (LTS), reported in Fig. 7. Labels α are defined in Fig. 6.

(SKIP) $\langle G; \iota.P\rangle \xrightarrow{\epsilon} \langle G; P\rangle$

(FUPD) $\langle G; \mathsf{fupd}(F)@a.P\rangle \xrightarrow{\epsilon} \langle G \uplus (a \boxtimes F); P\rangle$ where $a \boxtimes F = \{(a,\tau,b) | (\tau,b) \in F\}$

(EMIT) $\langle G; \overline{\tau}\tau'@a.P\rangle \xrightarrow{\epsilon} \langle G; P || \Pi_{b \in G(\tau,a)} \langle \tau \copyright \tau'\rangle @b\rangle$

(NOTIFY) $\langle G; \langle \tau \copyright \tau'\rangle @a\rangle \xrightarrow{\tau \copyright \tau' @a} \langle G; \mathbf{0}\rangle$

(LAMBDA) $$\frac{j \in I \quad p_j = \tau(\tau_1)}{\langle G; \sum_{i \in I} p_i@a_i.P_i\rangle \xrightarrow{\tau\tau_1'@a} \langle G \uplus \tau_1' \boxdot T; \{\tau_1'/\tau_1\}P_j || p_j@a_j.P_j\rangle}$$
where $\tau_1' \boxdot T = \{(a,\tau,b) | (a,b) \in T\}$

(CHECK) $$\frac{j \in I \quad p_j = \tau\tau'}{\langle G; \sum_{i \in I} p_i@a_i.P_i\rangle \xrightarrow{p_j@a_j} \langle G; P_j\rangle}$$

(ASYNCH) $$\frac{}{\langle G; P\rangle \xrightarrow{(\tau\tau'@a)} \langle G; P || \langle \tau \copyright \tau'\rangle @a\rangle}$$

(COM) $$\frac{\langle G; P_0\rangle \xrightarrow{\tau\tau'@a} \langle G; P_0'\rangle \quad \langle G; P_1\rangle \xrightarrow{\langle \tau \copyright \tau'\rangle @a} \langle G; P_1'\rangle}{\langle G; P_0 || P_1\rangle \xrightarrow{\epsilon} \langle G; P_0' || P_1'\rangle}$$

(PAR) $$\frac{\langle G; P_0\rangle \xrightarrow{\alpha} \langle G'; P_0'\rangle}{\langle G; P_0 || P_1\rangle \xrightarrow{\alpha} \langle G'; P_0' || P_1\rangle}$$

Fig. 7. NPC LTS

Rule (SKIP) trivially fires the silent action. Rule (FUPD) changes the network topology, by appending the sub-network $a \boxtimes F$ to the environment G, i.e. all the flows departing from a in F. Rule (EMIT) allows for multicast communications: it spawns in the network an envelope for each subscriber in $G(\tau)(a)$. Note that the continuation policy P is executed regardless the reception of envelopes as typical in asynchronous communications. Notification of envelopes is ruled by (NOTIFY) as much like as the output in the asynchronous π-calculus. Rules (LAMBDA) and (CHECK) model input actions. In the former, the selected input p_j reads any signal with topic τ and binds τ_1 to τ_1' in an early-style semantics. When a check input is selected, only envelopes of topic τ in session τ_1 can be consumed. Notice that the reception by a check reaction of a topic does not change the network topology, because the two topics involved by the communication are already known. The reception of a fresh name (τ_1') by a lambda reaction, instead, can extend the environment knowledge of the component: the receiver can discover all the existing linkages involving the received name τ_1'. In the spirit of early-style semantics, we allow the rule to extend the topology with any possible graph (T). Differently from SC, these two rules can express external non-deterministic choice and can involve several components. Rule (ASYNCH) permits to any NCP state to perform an input, simply storing the received message for subsequent usages, allowing to arbitrarily delay the communication. Rule (COM) allows the communication of a free session name τ'. Finally, rule (PAR) has the standard meaning.

The Control Flow Logic for NCP We develop a Control Flow Analysis for NCP, with the aim of over-approximating all the possible behaviour of NCP processes. The analysis, still specified in two phases, is reported in Table 4, where $\mathsf{sbj}(P)$ collects all the component names included in P. To emphasise the relation between the two calculi, we overload the analysis component names $\mathcal{B}$ and $\mathcal{E}$ and we use the judgement $\mathcal{B}, \mathcal{E}, G_S \models \langle G; P\rangle$ (and, in turn, $\mathcal{B}, \mathcal{E}, G_S \models P$), that we make more precise, i.e. $\mathcal{B}_{NCP}, \mathcal{E}_{NCP}, G_S \models \langle G; P\rangle$, when needed. There, G_S stands for the static abstraction of the topic-graph. It includes the initial graph and all the possible arcs and vertices that can be added during the computation. The clauses rely on the auxiliary function $\mathbf{A}_P$, that given a process P, keeps track of the single actions in P, and whose definition is in the upper part of Table 4. Hereafter, we denote with el the generic element of a set. This analysis is correct w.r.t. the given semantics. Furthermore, we prove that G_S captures all the flows arising in the topology.

Theorem 4. ***(Subject Reduction)***
Let S a NPC state $\langle G; P\rangle$. If $S \xrightarrow{\alpha} S'$ and $\mathcal{B}, \mathcal{E}, G_S \models S$ then also $\mathcal{B}, \mathcal{E}, G_S \models S'$.

Proof Sketch The proof is by induction on $S \xrightarrow{\alpha} S'$.

$$
\begin{array}{l}
\mathbf{A}_P(\sum_{i\in I} p_i@a_i.P_i) = \bigcup_{i\in I} \mathbf{A}_P(p_i@a_i.P_i)\\
\mathbf{A}_P(p@a.P) = \{((p,P),a)\}\\
\mathbf{A}_P(\overline{\tau}\tau'@a.P) = \{(\overline{\tau}\tau',a)\} \cup \mathbf{A}_P(P)\\
\mathbf{A}_P(\langle \tau \circledcirc \tau'\rangle @a) = \emptyset\\
\mathbf{A}_P(\mathsf{fupd}(F)@a.P) = \{(\mathsf{fupd}(F),a)\} \cup \mathbf{A}_P(P)\\
\mathbf{A}_P(P_0|P_1) = \mathbf{A}_P(P_0) \cup \mathbf{A}_P(P_1)\\
\mathbf{A}_P(\iota.P) = \mathbf{A}_P(P)\\
\mathbf{A}_P(P)(a) = \{el | (el,a) \in \mathbf{A}_P(P)\}\\
\mathbf{E}_P(P) = \begin{cases} \{(\tau \circledcirc \tau', a)\} & \text{if } P = \langle \tau \circledcirc \tau'\rangle @a\\ \emptyset & \text{otherwise}\end{cases}\\
\mathbf{E}_P(P)(a) = \{el | (el,a) \in \mathbf{E}_P(P)\}
\end{array}
$$

$$
\begin{array}{ll}
\mathcal{B}, \mathcal{E}, G_S \models \langle G; P\rangle & \mathsf{iff}\ G \subseteq G_S \wedge \mathcal{B}, \mathcal{E}, G_S \models P\\
\mathcal{B}, \mathcal{E}, G_S \models P & \mathsf{iff}\ \forall a \in \mathsf{sbj}(P).\mathbf{A}_P(P)(a) \subseteq \mathcal{B}(a) \wedge \mathbf{E}_P(P)(a) \subseteq \mathcal{E}(a)
\end{array}
$$

$$
\begin{array}{l}
\mathsf{fupd}(F) \in \mathcal{B}(a) \Rightarrow \mathbf{A}_F(F) \subseteq G_S(a)\\
\overline{\tau}\tau' \in \mathcal{B}(a) \wedge (\tau, b) \in G_S(a) \Rightarrow \tau \circledcirc \tau' \in \mathcal{E}(b)\\
\tau \circledcirc \tau' \in \mathcal{E}(a) \wedge (\tau\tau', P) \in \mathcal{B}(a) \Rightarrow \mathcal{B}, \mathcal{E}, G_S \models P\\
\tau \circledcirc \tau' \in \mathcal{E}(a) \wedge (\tau(\tau''), P) \in \mathcal{B}(a) \Rightarrow G(\tau') \subseteq G_S \wedge \mathcal{B}, \mathcal{E}, G_S \models \{\tau'/\tau''\}P
\end{array}
$$

Table 4. Analysis for NCP

Theorem 5. ***(Flows $\mathcal{F}$)*** *If $\mathcal{B}, \mathcal{E}, G_S \models S$ and $S \to^* S' \xrightarrow{\alpha} S''$, such that the last transition $S' \xrightarrow{\alpha} S''$ is derived using the rule (FUPD) on the set F in a component* a*, then $\mathbf{A}_F(F) \subseteq G_S(a)$.*

Proof Sketch By Theorem 4, we have that $\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E} \models S'$. Therefore, the proof proceeds by induction on the transition rules used to derive $S' \xrightarrow{\alpha} S''$.

Note that the above theorem formally represents the projection of the choreography over a component. Namely, it provides the local view of the choreography policy.

The NCP control flow analysis can be used to verify if certain choreography constraints are satisfied, for instance, on the security side. We can assert when a service does not capture a certain topic, and then statically verify, by inspecting the analysis results, whether this assertion is not violated.

Given a policy P and a graph G, let the set of systems reachable from $\langle G; P\rangle$ be defined as $Reach(\langle G; P\rangle) = \{\langle G'; P'\rangle | \langle G; P\rangle \rightarrow^* \langle G'; P'\rangle\}$. Let the flows emanating from a in $\langle G; P\rangle$ be defined as $F_{topic}(\langle G; P\rangle)(a) = \{(\tau, b) | (a, \tau, b) \in G\}$ and $F_{topic}(\langle G; P\rangle)(a, \tau) = \{b | (a, \tau, b) \in G\}$.

Definition 1. *Given a process P, a graph G, a topic τ, and component* a *occurring in P, we say that a* does not capture τ *if $F_{topic}(\langle G'; P'\rangle)(a, \tau) = \emptyset$ for all $\langle G'; P'\rangle \in Reach(\langle G; P\rangle)$.*

Again, an analysis component, G_S, can be used to predict at compile time whether the constraint is respected. Actually, because of safety, we can assess that if the property is statically guaranteed, then it will also be at run time, as stated by the following result, whose proof is based on Theorem 5.

Theorem 6. *Given a process P, a graph G, a topic τ, and component* a *occurring in P, if $G_S(a) = \emptyset$ then* a does not capture τ.

Proof Sketch The proof proceeds by contradiction, by assuming that *a does capture τ*.

Our analysis here acts in a *descriptive* way, i.e. it describes if a property violation is possible and because of soundness, we can prove that if no violation is found, no violation can arise at run-time. In the same setting, our approach can have a *prescriptive* value. In this case, we aim at *preventing* violation to arise, by suggesting how to instrument the code with the necessary checks, e.g. by enriching the multicast group with a filter discarding the events referring to the unauthorized topic.

6 Checking Choreography

Consistency between network of SC components and the global coordination policies expressed by NCP specifications is formally verified in [17]. Verification is based on the encoding from SC networks to NCP policies, presented in Table 5, and on bisimilarity. This result can also suggest a model driven development approach. The designer can define successive SC models for implementing a choreography model, obtained by incremental refinement.

The basic idea of the encoding is to transform SC reductions into NCP transitions labeled with ϵ. The encoding uses the following functions: (i) $[\![B]\!]_a$ which takes an SC behaviour B, localised within a, and maps it into an NCP policy; (ii) $\llbracket R \rrbracket_a$ which takes a reaction R, installed in the interface of a, and maps it into a policy; and (iii) $\llbracket N \rrbracket$ which takes a network N and maps it into a state.

$\llbracket 0 \rrbracket_a = \mathbf{0}$	$\llbracket B \mid B' \rrbracket_a = \llbracket B \rrbracket_a \Vert \llbracket B' \rrbracket_a$
$\llbracket \epsilon; B \rrbracket_a = \iota.\llbracket B \rrbracket_a$	$\llbracket \mathsf{out}\langle \tau \copyright \tau' \rangle B \rrbracket_a = \overline{\tau}\tau' @a \llbracket B \rrbracket_a$
$\llbracket \mathsf{rupd}(R); B \rrbracket_a = \iota.\llbracket R \rrbracket_a \Vert \llbracket B \rrbracket_a$	$\llbracket \mathsf{fupd}(F); B \rrbracket_a = \mathsf{fupd}(F) @a.\llbracket B \rrbracket_a$
$\llbracket 0 \rrbracket_a = \mathbf{0}$	$\llbracket R \mid R' \rrbracket_a = \llbracket R \rrbracket_a \Vert \llbracket R' \rrbracket_a$
$\llbracket \tau \copyright \tau' \gtrdot B \rrbracket_a = \tau\tau' @a.\llbracket B \rrbracket_a$	$\llbracket \tau \lambda \tau' \gtrdot B \rrbracket_a = \tau(\tau') @a \llbracket B \rrbracket_a$
$\llbracket \emptyset \rrbracket = \langle \mathbf{0}; \mathbf{0} \rangle$	$\llbracket \langle \tau \copyright \tau' \rangle @a \rrbracket = \langle \mathbf{0}; \langle \tau \copyright \tau' \rangle @a \rangle$
$\dfrac{\llbracket N \rrbracket = \langle G; P \rangle \quad \llbracket N' \rrbracket = \langle G'; P' \rangle}{\llbracket N \Vert N' \rrbracket = \langle G \uplus G'; P \Vert P' \rangle}$	
$\left\llbracket a[B]_F^R \right\rrbracket = \langle G; \llbracket B \rrbracket_a \Vert \llbracket R \rrbracket_a \rangle$ where $G = a \boxtimes F$	

Table 5. Encoding of behaviours, reactions and networks

Control Flow Analysis provides us with an approximation of behaviours, both for the choreography model (NCP) and the actual design (SC). The consistency result is reflected by the correspondence between the analysis estimate $(\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E})$ of a network N and that $(\mathcal{B}_{NCP}, \mathcal{E}_{NCP}, G_S)$ of its encoding $\llbracket N \rrbracket$. We need the following auxiliary function that maps each element possibly occurring in $(\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E})$, in the corresponding element occurring in $(\mathcal{B}_{NCP}, \mathcal{E}_{NCP}, G_S)$.

$$\begin{array}{ll}
Enc(\mathsf{out}\langle \tau \copyright \tau' \rangle) = \overline{\tau}\tau' & Enc(\mathsf{fupd}(F)) = \mathsf{fupd}(F) \\
Enc((\tau \copyright \tau', B)) = (\tau\tau', Enc(B)) & Enc((\tau \lambda \tau', B)) = (\tau(\tau'), Enc(B)) \\
Enc(\tau \copyright \tau') = \tau \copyright \tau' & Enc((\tau, b)) = (\tau, b)
\end{array}$$

Example 3. We illustrate this correspondence on the following example, given by a network N having two components: a and b.

$$N = a[0]_{\tau \rightsquigarrow \{b\}}^{\tau\lambda\tau' \gtrdot \mathsf{out}\langle \tau \copyright \tau' \rangle} \Vert\ b[0]_{\tau_1 \rightsquigarrow \tilde{c}}^{\tau\lambda\tau' \gtrdot \mathsf{out}\langle \tau_1 \copyright \tau' \rangle} \Vert \langle \tau \copyright \tau'' \rangle @a \Vert \langle \tau \copyright \tau''' \rangle @b$$

The corresponding encoding is given by the following state S:

$$S = \langle (\emptyset, \{(a, \tau, \{b\}), (b, \tau, \tilde{c})\}); \tau(\tau') @a.\overline{\tau}\tau' @a \Vert \tau(\tau') @b.\overline{\tau_1}\tau' @b \rangle \Vert \langle \tau \copyright \tau'' \rangle @a \Vert \langle \tau \copyright \tau''' \rangle @b$$

The analyses of N and of S, reported in Table 6, show the correspondence between the estimates components.

Now, we formally state the correspondence between the two analyses.

Theorem 7. *Given a network N and $(\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E})$, such that $\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E} \models N$, let $\langle G; P \rangle = \llbracket N \rrbracket$ and $(\mathcal{B}, \mathcal{E}, G_S)$ such that $\mathcal{B}, \mathcal{E}, G_S \models \langle G; P \rangle$. We have that for all* a *in the domain of $(\mathcal{B}, \mathcal{R}, \mathcal{F}, \mathcal{E})$, we have that:*

- $\forall el \in \mathcal{B}(a) : Enc(el) \in \mathcal{B}_{NCP}(a)$ • $\forall el \in \mathcal{R}(a) : Enc(el) \in \mathcal{B}_{NCP}(a)$
- $\forall el \in \mathcal{E}(a) : Enc(el) \in \mathcal{E}_{NCP}(a)$ • $\forall (\tau, b) \in \mathcal{F}(a) : Enc(\tau, b) \in G_S(a)$

$\mathcal{E}(a) \ni \tau_{©}\tau''$	$\mathcal{E}(b) \ni \tau_{©}\tau''', \tau_{©}\tau''$
$\mathcal{B}(a) \ni \mathsf{out}\langle \tau_{1©}\tau'' \rangle$	$\mathcal{B}(b) \ni \mathsf{out}\langle \tau_{1©}\tau''' \rangle$
$\mathcal{R}(a) \ni (\tau\lambda\tau', \mathsf{out}\langle \tau_{©}\tau' \rangle)$	$\mathcal{R}(b) \ni (\tau\lambda\tau', \mathsf{out}\langle \tau_{1©}\tau' \rangle)$
$\mathcal{F}(a) \supseteq \{(\tau, b)\}$	$\mathcal{F}(b) \supseteq \{(\tau_1, c_i) \mid c_i \in \tilde{c}\}$
$\{(a, \tau, \{b\}), (b, \tau, \tilde{c})\} \in G_S$	$G(\tau'') \cup G(\tau''') \in G_S$
$\mathcal{E}_{NCP}(a) \ni \tau_{©}\tau''$	$\mathcal{E}_{NCP}(b) \ni \tau_{©}\tau''', \tau_{©}\tau''$
$\mathcal{B}_{NCP}(a) \ni (\tau(\tau'), \overline{\tau}\tau'@a), \overline{\tau}\tau''$	$\mathcal{B}_{NCP}(b) \ni (\tau(\tau'), \overline{\tau_1}\tau'@b), \overline{\tau_1}\tau'''$

Table 6. Some Entries of the Analysis of N (upper part) and of S (lower part)

Proof Sketch The proof proceeds by structural induction.

The correspondence of the two analyses is made easier by our assumption on the absence of restriction and scope extrusion in NCP. As a consequence, the treatment of internal actions is strongly simplified. The more involved reasonings, needed to cope with the full calculus, require further investigation.

7 Concluding Remarks

We have introduced Control Flow Analysis for the SC-NCP framework for service coordination. Our approach is based on a two layer calculus (in the spirit of [11, 12, 8]). The abstract level (NCP) provides a declarative framework to specify the service coordination, while the concrete level (SC) allows us to design the behaviors of the services. The distinguished feature of our approach is given by the mixed descriptive-prescriptive mechanism offered by the Control Flow Analysis. This provides us flexible facilities to manage a wide range of properties.

The SC-NCP programming model has provided the foundational basis to design and implement the JSCL middleware for services. The correspondence result, stated in Section 6, provides a further formal hook to freely move inside the two-level structure of JSCL. Depending on the level of the structure, one can focus on either the design or the choreography, with the guarantee that the key features are preserved. Differently from the bisimulation mechanism in [17], here we have proceeded in a static way, allowing for choreography rehearsal.

We plan to equip the JSCL framework to include the reasoning machineries available by implementing the analyses developed in the present paper. We intend to exploit the analysis to statically verify that a design is compliant with the specification of the choreography demands, and to instrument the code to avoid occurrence of certain events at run-time.

References

1. R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous pi-calculus. *Theor. Comput. Sci.*, 195(2):291–324, 1998.
2. M. Bartoletti, P. Degano, G. Ferrari, and R. Zunino. Secure service orchestration. In *FOSAD*, volume 4667 of *LNCS*. Springer, 2007.

3. C. Bodei, L. Brodo, P. Degano, and H. Gao. Detecting and preventing type flaws at static time. *Journal of Computer Security*, to appear, 2009.
4. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. R. Nielsoni. Static validation of security protocols. *Journal of Computer Security*, 13(3):347–390, 2005.
5. C. Bodei, P. Degano, F. Nielson, and H. R. Nielsoni. Static analysis for the π-calculus with their application to security. *Info. & Computat.*, 165:68–92, 2001.
6. C. Bodei and G. L. Ferrari. Choreography rehearsal. Technical Report TR-09-11, Dipartimento di Informatica, Univ. Pisa, 2009.
7. M. Boreale, R. Bruni, L. Caires, R. D. Nicola, I. Lanese, M. Loreti, F. Martins, U. Montanari, A. Ravara, D. Sangiorgi, V. T. Vasconcelos, and G. Zavattaro. SCC: A service centered calculus. In *WS-FM*, volume 4184 of *LNCS*, pages 38–57. Springer, 2006.
8. M. Bravetti and G. Zavattaro. A foundational theory of contracts for multi-party service composition. *Fundam. Inform.*, 89(4):451–478, 2008.
9. R. Bruni. Calculi for service oriented computing. In *SFM 2009*, volume 5569 of *LNCS*. Springer, 2009.
10. R. Bruni, I. Lanese, H. C. Melgratti, and E. Tuosto. Multiparty sessions in soc. In *COORDINATION 2008*, volume 5052 of *LNCS*, pages 67–82. Springer, 2008.
11. N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Choreography and orchestration: A synergic approach for system design. In *ICSOC*, volume 3826 of *LNCS*, pages 228–240. Springer, 2005.
12. M. Carbone, K. Honda, and N. Yoshida. Structured communication-centred programming for web services. In *ESOP 2007*, volume 4421 of *LNCS*, pages 2–17. Springer, 2007.
13. V. Ciancia, G. L. Ferrari, R. Guanciale, and D. Strollo. Checking correctness of transactional behaviors. In *FORTE 2008*, volume 5048 of *LNCS*. Springer, 2008.
14. V. Ciancia, G. L. Ferrari, R. Guanciale, and D. Strollo. Global coordination policies for services. In *FACS 2008*, Electronic Notes in Theoretical Computer Science. Elsevier, 2009. To appear.
15. G. L. Ferrari, R. Guanciale, and D. Strollo. Jscl: A middleware for service coordination. In *FORTE 2006*, volume 4229 of *LNCS*, pages 46–60. Springer, 2006.
16. G. L. Ferrari, R. Guanciale, D. Strollo, and E. Tuosto. Refactoring long running transactions. In *WS-FM 2008*, volume 5387 of *LNCS*, pages 127–142. Springer, 2009.
17. R. Guanciale. *The Signal Calculus: Beyond Message-based Coordination for Services*. PhD thesis, Institute for Advanced Studies, IMT, Lucca, 2009.
18. C. Guidi, R. Lucchi, R. Gorrieri, N. Busi, and G. Zavattaro. A calculus for service oriented computing. In *ICSOC*, volume 4294 of *LNCS*. Springer, 2006.
19. A. Lapadula, R. Pugliese, and F. Tiezzi. A calculus for orchestration of web services. In *ESOP*, volume 4421 of *LNCS*, pages 33–47. Springer, 2007.
20. H. R. Nielson and F. Nielson. Flow logic: A multi-paradigmatic approach to static analysis. In *The Essence of Computation*, volume 2566 of *LNCS*. Springer, 2002.
21. D. Strollo. *Designing and Experimenting Coordination Primitives for Service Oriented Computing*. PhD thesis, Institute for Advanced Studies, IMT, Lucca, 2009.
22. J. Su, T. Bultan, X. Fu, and X. Zhao. Towards a theory of web service choreographies. In *Proc. of WS-FM 2007*, Brisbane, Australia, 2007. Springer.
23. O. TC. Business process execution language for web services version 2.0. http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.html.
24. Web services choreography description language version 1. http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/.