Università di Pisa

Dipartimento di Informatica

Technical Report: TR-08-11

Global coordination policies for services

Vincenzo Ciancia, Gian Luigi Ferrari, Roberto Guanciale, Daniele Strollo

30 May 2008

Address: Largo B. Pontecorvo 3, 56127 Pisa, Italy. Tel: +39 050 2212700 Fax: +39 050 2212726

Abstract

An important issue of the service oriented approach is the possibility to aggregate, through programmable coordination patterns, the activities involved by service interactions. Two different approaches can be adopted to tackle service coordination: *orchestration* and *choreography*. In this paper, we introduce a formal methodology for the with the aim of handling coordination among services from the perspective of a global observer in the spirit of choreography models. In particular, we address the problem of verifying compliance and consistency between the design of service interactions and the choreography constraints.

1 Introduction

The web service protocol stack (e.g. WSDL, UDDI, SOAP) provides *basic* support for the development of service-oriented architectures by exploiting facilities to publish, discover and invoke network-available services. The service protocol stack has been extremely valuable to highlight the key innovative features of the service oriented computing approach. Most of the current development methodologies and standards are focused on composition of services. Two different approaches can be adopted: *orchestration* and *choreography*. In the orchestration, an intermediate component, the *orchestrator* is responsible to arrange service activities according to the work-flow plan. This strategy provides a *local view* of the participants. From the other hand, the *choreography* model involves all parties and their associated interactions providing a *global view* of the system. Relevant standard technologies have emerged to model coordination policies. Among them, particular relevance is given to the Business Process Execution Language (BPEL) [17], for the orchestration, and Web Service Choreography Description Language (WS-CDL) [19], for the choreography. However, it is not infrequent that such standards have drawbacks. In fact, constructs are often informally specified. This usually leads to ambiguities or redundancy. Several research efforts are currently devoted to provide foundational models for orchestration and choreography (see COWS [13], Global Calculus [5], λ_{req} [2] ORC [16], SCC [3], SOCK [11] to cite a few). However, research is still underway.

A well known paradigm for specifying and programming distributed systems is the *event notification* paradigm (EN, for short), where distributed computational components can act as publishers and/or subscribers. When a component intends to send data to or requests a service from other components, it issues an event that eventually shall trigger a reaction from subscribers that previously subscribed for such kind of events. The EN paradigm provides a suitable framework to deal with *service oriented architectures* (SOAs) that require components to be loosely coupled. Specifically, the EN paradigm features high level coordination mechanisms that allow programmers/designers to decouple components and rely entirely on event handling. In [8] a middleware for service coordination called *Java Signal Core Layer* (JSCL) has been introduced. The middleware consists of a set of API for programming services interacting through suitable events. JSCL is equipped with a graphical environment, an eclipse plug-in, providing capabilities for designing components and their inter-connections. The JSCL API's are available at `www.tao4ws.net`. A distinguished feature of JSCL consists in the strict interplay among formal semantic foundations, implementation pragmatics and experimental evaluation of the resulting programming constructs. More precisely, all the programming facilities available in JSCL have been motivated semantically. Indeed, at the abstract level, the middleware takes the form of the *Signal Calculus* [8,9,6]. The SC calculus is a variant of the π-calculus [18] with explicit primitives to deal with event notification and component distribution. The SC & JSCL framework allows one to specify and program services coordination policies (orchestration and choreography) relying on event notification only. Moreover, it features sessions as a mechanism to synchronize workflows of distributed and independent components. Remarkably, the middleware does not assume any centralized mechanism for publishing, subscribing and notifying events. Indeed, each subscriber explicitly defines the class of events it is interested in. In [15] this pattern is referred to as *non brokered*, in contrast with the *brokered* solutions that implements publish/subscribe mechanisms on top of a classification of signals without taking into account the involved components. Basically, brokered solutions rely on global state space e.g. *linda tuple spaces* [10]. All SC notions are reflected in the JSCL API's. Indeed, the design choices underlying the JSCL implementation have been formally motivated in terms of the SC calculus. Hence, SC and JSCL can be regarded as a full-fledged framework for specifying, verifying and programming coordination policies of distributed services.

In SC, and coherently in JSCL, components are thought of as isolated and their behavior is independent from the network context they are going to operate. Only once plugged into the network, components receive information regarding their neighbors, namely the subscribers that are directly connected. Hence, a local view of coordination has been considered. In this paper, we introduce a formal methodology for the SC & JSCL framework with the aim of handling coordination among services from the perspective of a global observer in the spirit of choreography models. In particular, we address the problem of verifying compliance and consistency between the design of service interactions and the choreography constraints.

Our approach is based on process calculi techniques. We introduce a process calculus, called Network Coordination Policies (NCP) that extends and equips our framework with a choreography model. The two languages (SC and NCP) lay at two different levels. The former is aimed to the (formal) design of services, the latter is a specification language that expresses coordination policies. Policies take the form of processes that express the behavior that is seen by an observer standing from a *global* point of view, thus seeing all the public communications that happen on the network. Each process thus describes what are the interactions that are expected to happen, and how these are interleaved. Indeed, certain features can be described at both levels: the specification describes *what* is expected from a network of services, the design specifies *how* to implement it. Formally, this means that the two languages are similar in certain aspects, and the two semantics are related by a correctness result: for each SC network, there is an NCP policy that reflects all the properties of the network. We establish this by giving a semantics of SC into NCP that is fully abstract. The converse is not true: as it is typical of distributed systems, not every property that one can specify is implementable. As an example, consider a causality requirement on two different components that do not share a direct or indirect communication link.

In this work, we provide an operational semantics of NCP, that allow us to reason about the behavior of SC services when plugged into suitable network contexts with certain choreography constraints. In particular, it distinguishes services that behave differently in the same network context. This feature is useful to evaluate how the invocation of a service is successful (e.g. meets the SLA constraints) only within certain kinds of choreographies.

Technically, our approach is inspired by the "directed HT bisimulation" for the asynchronous π-calculus as presented in [12,1]. As it happens in the π-calculus, NCP features scope extrusion as the fundamental capability to model fresh resouce generation. However, names that are object of freshness and scope extrusion are not *pure names*, but rather they carry a network topology which is considered fresh and is extruded when received. On one hand, this comes from the fact that the basic structure of NCP are network topologies and not just channel names. On the other hand, this gives a model for private subnetworks in a process calculus. While not adding or removing any expressive power to the calculus, this "network binding" operation is a natural model for all these real-world situations in which an entire sub-network can be hidden or discovered, independently from the presence of a single access point like it happens in the π-calculus.

The embedding of the SC semantics into NCP also provides a way to validate a design against its specification by bisimulation checking. The main result of this paper, namely the correctness result that bridges the gap between the choreography model and the actual design, is formally proved by checking bisimilarity between the specification of the choreography model, i.e. the NCP specification, and the SC design.

2 Preliminaries: The Signal Calculus

In this section we review the syntax and the operational semantics of the Signal Calculus (SC). We assume a countable set $\mathcal{T}$ of *topic* names (ranged over by τ) and a countable set of component names A, ranged over by $a, b, c, \ldots$. We adopt the notation $\vec{a}$ to denote a set of component names.

The calculus is centered around the notion of *component*. A component is the container of a service. A component is uniquely identified by a name a (the public address of the service) and has an internal behavior. Components exchange messages, called *signals*. Signals are pairs of topics $\tau©\tau'$, where the first element is the signal type (a unique name identifying the kind of event) and the second element is the session identifier. Session identifiers and event kinds are freely interchangeable, and can be dynamically generated. When an event is raised by a component (the *publisher*), it is notified to the components interested in handling it (the *subscribers*). Notice that notifications are not anonymous, namely subscription relates both the event topic and the publisher. Therefore, components behave as reactive agents that declare the set of event kinds they are interested in together with the associated tasks to perform for their handling (*reactions*), and the set of target components for the notification delivery (*flows*). The calculus provides two different kinds of reaction: the *lambda reactions* and the *check reactions*. Lambda reactions are activated independently from the signal session, while check reactions handle signals belonging to a well defined session. Lambda reactions, once installed, remain persistent in the component interface, the check reactions, instead, once executed, are removed from the component interface.

We now introduce the syntax of the calculus. We start by illustrating the syntax of reactions.

$$R \quad ::= \quad 0 \quad | \quad \langle\rho\rangle \to B \quad | \quad R|R$$

where the *input prefix* $\langle\rho\rangle$ is either a lambda reaction $(\tau©\lambda\tau')$ or check reaction $(\tau©\tau')$.

The *lambda reaction* $\tau©\lambda\tau' \to B$ is triggered by signals having topic τ independently from their session. Conversely, the *check reaction* $\tau©\tau' \to B$ reacts only to signals having topic τ issued for the session τ'. Once a reaction to a certain signal occurs, the behavior B starts its execution in parallel with the already active behavior. Notice that for a lambda reaction the name τ' is bound in the behavior B, while, for a check reaction, it is free. *Reaction composition* allows a component to react to different kinds of signal in different ways.

Now, we introduce the syntax of *behaviors*, i.e. the constructs components execute to deal with coordination issues. Behaviors are described by the following grammar:

$$\begin{array}{rcll} B & ::= & \mathbf{out}\langle\tau©\tau'\rangle.B & \textit{(Signal emission)} \\ & | & (\nu\tau)B & \textit{(Topic restriction)} \\ & | & \mathbf{rupd}\,(R)\,.B' & \textit{(Reaction update)} \\ & | & \mathbf{fupd}(F).B' & \textit{(Flow update)} \\ & | & B \mid B' & \textit{(Parallel composition)} \\ & | & 0 & \textit{(Empty behavior)} \end{array}$$

where a *flow* F is a set of pairs of the form (τ, b) such that τ is a topic and b a component name.

The signal emission $\mathbf{out}\langle\tau©\tau'\rangle.B$ spawns into the network a signal of topic τ and session τ', and then continues as B. Topics can be dynamically generated via the *restriction* operator acting as a binder, namely, the occurrences of τ in B are bound. The calculus provides two primitives to allow a component to dynamically change its interface: the *reaction update* $\mathbf{rupd}\,(R)\,.B'$ and the *flow update* $\mathbf{fupd}(F).B'$. The former installs a new reaction R in the interface part of the component and the latter appends F to its flows. The remaining constructs have the obvious meaning.

Networks describe the component distribution and the signals exchanged among components.

$$N ::= \quad \emptyset \quad | \quad a[B]_F^R \quad | \quad N \parallel N \quad | \quad \langle\tau©\tau'\rangle@a \quad | \quad (\nu\tau)N$$

A network can be empty $\emptyset$, a single component $a[B]^R_F$, or the parallel composition of networks $N \parallel N'$. Networks carry signals exchanged among components. The signal emission spawns into the network, for each target component, an "envelope" $\langle \tau \copyright \tau' \rangle @ a$ containing the signal and the target component name a. The last operator allows one to extend the scope of dynamic topics within networks. Hereafter, we assume that components are uniquely identified by their names. Hence, we will always consider well formed networks, namely networks where components with the same name are not allowed.

2.1 SC *Operational Semantics*

We briefly outlines the SC reduction semantics as given in [9]. We first define structural congruence. This is the smallest equivalence relation that satisfies the commutative monoidal laws for $(R, |, 0)$, $(B, \ |\ , 0)$ and $(N, \|, \emptyset)$. Additionally, the following laws hold:

$$\begin{array}{cc}
(\nu\tau)0 \equiv 0, & ((\nu\tau)B) \mid B' \equiv (\nu\tau)(B \mid B'), \text{ if } \tau \notin fn(B') \\
(\nu\tau)\emptyset \equiv \emptyset, & ((\nu\tau)N) \parallel N' \equiv (\nu\tau)(N \parallel N'), \text{ if } \tau \notin fn(N') \\
(\nu\tau)(\nu\tau')B \equiv (\nu\tau')(\nu\tau)B & (\nu\tau)(\nu\tau')N \equiv (\nu\tau')(\nu\tau)N
\end{array}$$

and, if $B \equiv B'$,

$$\tau \copyright \lambda\tau' \to B \equiv \tau \copyright \lambda\tau' \to B' \tag{1}$$

$$\tau \copyright \tau' \to B \equiv \tau \copyright \tau' \to B' \tag{2}$$

where τ' can be alpha converted in (1). Finally, in the case of networks, the following equations hold:

$$\frac{F_1 \equiv F_2 \quad B_1 \equiv B_2 \quad R_1 \equiv R_2}{a[B_1]^{R_1}_{F_1} \equiv a[B_2]^{R_2}_{F_2}}, \qquad \frac{\tau \notin fn(R) \cup fn(F) \cup \{a\}}{a[(\nu\tau)B]^R_F \equiv (\nu\tau)a[B]^R_F}.$$

The reduction semantics describes how components can communicate and update their interfaces. The reduction relation $\to$ is depicted in Figure 1. The intuitive interpretation of the reduction rules is straightforward. Notice that rule *emit* introduces in the network a set of envelopes, i.e. an envelope for each of the subscriber components. The rule exploits the auxiliary operator $(F(\tau))$, defined as follows:

$$F(\tau) = \{b \mid (\tau, b) \in F\}$$

Rules *check* and *lambda* describes the activation of *check* reactions, that require the exact match of the session identifier, and of *lambda* reactions, receiving the session identifier as argument.

$$\frac{N \to N'}{N \parallel M \to N' \parallel M}\ (npar) \qquad \frac{N \equiv N' \quad N' \to M' \quad M' \equiv M}{N \to M}\ (struct)$$

$$\frac{a[B]^R_F \to a[B']^{R'}_{F'}}{a[B \mid B_1]^R_F \to a[B' \mid B_1]^{R'}_{F'}}\ (par) \qquad \frac{N \to N_1}{(\nu\tau)N \to (\nu\tau)N_1}\ (new)$$

$$a[\mathbf{rupd}\,(R')\,.B]^R_F \to a[B]^{R|R'}_F \quad (rupd) \qquad a[\mathbf{fupd}(F').B]^R_F \to a[B]^R_{F\cup F'} \quad (fupd)$$

$$\frac{F(\tau) = \{b_1, \ldots, b_n\}}{a[\mathbf{out}\langle\tau©\tau'\rangle.B]^R_F \to a[B]^R_F \parallel \langle\tau©\tau'\rangle@b_1 \parallel \ldots \parallel \langle\tau©\tau'\rangle@b_n}\ (emit)$$

$$\langle\tau©\tau'\rangle@a \parallel a[B]^{\tau©\tau'\to B'|R}_F \to a[B|B']^R_F \quad (check)$$

$$\langle\tau©\tau'\rangle@a \parallel a[B]^{\tau©\lambda\tau_1\to B'|R}_F \to a[B|\{\tau'/\tau_1\}B']^{\tau©\lambda\tau_1\to B'|R}_F \quad (lambda)$$

Fig. 1. Operational semantics

3 The choreography model

In this section we introduce the syntax and the operational semantics of the Network Coordination Policy calculus (`NCP`). This calculus has been specifically designed to be the choreography model for `SC`. Basically, `NCP` is an extension of the asynchronous π-calculus supporting multi-cast communication and multi-layered dynamic topologies with hidden network layers. Network layers are first order entities: they can be dynamically created and exchanged in communications. Many other process calculi has been designed to deal with process distribution. The novel feature of `NCP` is the capability to restrict a part of the network topology.

The syntax of the language is defined as follows:

$$\begin{array}{lcl} P & ::= & P \parallel P \quad | \quad (\nu\tau : T)\,P \quad | \quad skip.P \quad | \\ & & \sum_{i\in I} \rho_i@a_i.P_i \quad | \quad \langle\tau©\tau'\rangle@a \quad | \quad \overline{\tau}\tau'@a.P \quad | \quad \mathbf{fupd}(F)@a.P \\ \rho & ::= & \tau\,(\tau') \quad | \quad \tau\tau' \end{array}$$

where T is a non empty set of pairs of the form (a, b), $a \neq b$, called *linkage*. Finally, I is a finite set of indices. We use $\emptyset$ as a shorthand to denote the empty guarded sum $\sum_{i\in\emptyset} \rho_i@a_i.P_i$.

A `NCP` process is called a *coordination policy*. We use the word policy to emphasize the fact that the calculus has been introduced to specify and constrain the behaviour of SC networks.

A policy $\rho@a.P$ describes from a global standpoint the execution of the reaction ρ by the component a, with continuation P. Besides reactions, we have other forms of prefixing. Prefix $\tau\,(\tau')$ describes the action of receiving any topic as input by listening on topic τ. We call this kind of action *lambda* action because it is tailored to describe SC *lambda reaction*. Similarly, Prefix $\tau\tau'$ describes the action of receiving signals having topic τ and session τ'. We call *check* this action. As it will be clearer later, the two actions above, lambda and check, provide NCP with a (restricted) form of recursion and a (restricted) form of matching. The policy $\overline{\tau}\tau'$ describes the emission of an envelope on session τ' for those services that are listening on topic τ. The action $\mathbf{fupd}(F)$ allows one to describe the operation that updates the linkages of a SC component. The *envelope* $\langle\tau©\tau'\rangle@a$ represents a message (whose destination is a) still pending in the network. Notice that the source component generating the enveloped is not specified. $(\nu\tau : T)\,P$ defines the scope of the topic τ, with an associated linkage T that represent an hidden network layer, in the policy P. The topic τ is assumed to be fresh. For example, the restriction $(\nu\tau : \{(a,b)\})\,P$ extends the network topology with a new linkage between a and b for the new topic τ. Notice that this mechanism permits to express network hiding, via the name restriction, and multi layer network scoping, via the name binding. Finally the policy *skip.P*, represents the execution of an internal activity before the execution of P.

Free names $\mathrm{fn}(P)$ and bound names $\mathrm{bn}(P)$ of a policy are defined as usual. The operational semantics of NCP defines the meaning of policies from a global standpoint. Since, policies describes not only the interactions among components but also reflects the structure of the event topology within a network, we need to introduce a suitable notion of state.

Define the *topic-driven topology* to be a set of triples (a, τ, b) where a, b are component names and τ is a topic name. Hereafter, we use μ to range over topic-driven topologies. We introduce some useful auxiliary operations on topic-driven topology.

- Let μ be a topic-driven topology, let a be a component name and F be a flow. Define $\mu \oplus (a \times\ F)$ to be the topic-driven topology $\mu \cup \{(a, \tau, b) \mid (\tau, b) \in\ F\}$.
- Let μ be a topic-driven topology, let τ be topic. Define $\mu(\tau)$ to be the linkage $\{(a, b) \mid (a, \tau, b) \in \mu\}$. Similarly, function $\mu(\tau)(a)$ is defined to the the set $\{b \mid (a, \tau, b) \in \mu\}$.
- Let μ be a topic-driven topology, let τ be a topic name and T be a linkage. Define $\mu \ominus (\tau \times\ T)$ to be the topic-driven topology $\mu - \{(a, \tau, b) \mid (a, b) \in\ T\}$, where $-$ denotes the difference between sets.

Let be μ a *topic-driven topology* and P a *coordination policy*, then the pair (μ, P) is called NCP state.

We now introduce the labelled transition system semantics of NCP. The operational rules are similar in spirit to the rules given by Honda and Tokoro [12], and Amadio, Castellani and Sangiorgi [1] in the case of the asynchronous π-calculus. Our operational semantics exploits the notion of topic-driven topology to manage explicitly the global view of a choreography. Indeed, the evaluation of a coordination policy depends on the state of affairs of the topology of the events in the network This enables us to model in a natural way multi-cast communication. For example, listening on the topic τ in the action $\overline{\tau}\tau'$ is not sufficient to receive messages on that topic. In fact, we require that the topic-driven topology must link the sender and the receiver for the topic τ.

We start by introducing the set of observable actions α.

$$\alpha \quad ::= \quad \epsilon \quad | \quad \tau\tau'@a \quad | \quad (\tau\tau'@a) \quad | \quad \langle \tau © \tau' \rangle @a \quad | \quad \langle \tau © (\tau' : T) \rangle @a$$

The action ϵ models unobservable activities, like internal communications. Action $\langle \tau © \tau' \rangle @a$ is the *free* (asynchronous) event notification of kind τ, session τ' and destination a. Action $\langle \tau © (\tau' : T) \rangle @a$ represents a *bound* event notification; the linkage T is exploited to delimit the scope of the event in the network. Action $\tau\tau'@a$ is a free reaction activation and is inspired by the semantics of the asynchronous π-calculus in the early instantiation style. Finally $(\tau\tau'@a)$ represents the action of receiving a message and storing it in parallel with the current process. This action is observable in any system, thus including the empty policy. Hereafter, we use $n(\alpha)$ to denote the set of names in the action α.

The labelled transition system semantics of `NCP` is defined by the rules depicted in Figure 2, where $\equiv_n$ denotes the syntactic identity modulo $\alpha-$conversion. We use $\mu, P \xrightarrow{\alpha} \mu', P'$ to represent that the coordination policy P, plugged in the topology μ, by performing the action α evolves to the policy P' and the network topology to μ'. For simplicity, we omit symmetric rules for *par*, *com*, *close*, *new* and *open*. Rules *struct* and *par* have the standard meaning. *Skip* represents an internal computation. *Fupd* extends the topic-driven topology μ with the linkages outgoing from a derived from the flow information F. The rule *emit* models the asynchronous multi-casting communication. The rule checks the state of the topology $(\mu(\tau)(a))$ to derive the set of subscriber components for the topic τ. Then, for each subscriber one envelope is spawned into the network. The rule *notify* describes the notification of an envelope to a component and corresponds to the output rule for the asynchronous π-calculus. Rules *check* and *lambda* model the execution of reactions. If a check reaction is selected $(\rho_j = \tau\tau')$, the policy can read only envelopes having the same topic and session. If a lambda reaction is selected $(\rho_j = \tau\,(\tau'))$, the policy can read any signal having the topic τ independently of the identity of the session (τ' acts as a variable). Notice that, after the communication has occurred, all competitor inputs are garbaged. The rule *async* permits any policy to perform an input, simply storing the received message for subsequent usages (thus allowing to arbitrarily delay the communication). The rule *com* allows the communication of a session (τ') that is not under the scope of a restriction. The rule *new* allows one to extend the topology $(\mu \otimes (\tau \times T))$ for a fresh generated topic (τ). Notice that the rule hides the updates of the topic topology dependent from the generated name outside its scope $(\mu' \ominus (\tau' \times T'))$. Finally the rules *open* and *close* model the scope extrusion of the name (τ'), and of its bound communication.

$$\frac{}{\mu, skip.P \xrightarrow{\epsilon} \mu, P}\ (skip) \qquad \frac{}{\mu, \mathbf{fupd}(F)@a.P \xrightarrow{\epsilon} \mu \oplus (a \times F), P}\ (fupd)$$

$$\frac{\mu(\tau)(a) = \vec{b}}{\mu, \overline{\tau}\tau'@a.P \xrightarrow{\epsilon} \mu, P \parallel \prod_{b \in \vec{b}} \langle \tau \copyright \tau' \rangle @b}\ (emit) \qquad \frac{j \in I \quad p_j = \tau\tau'}{\mu, \sum_{i \in I} p_i@a_i.P_i \xrightarrow{\tau\tau'@a_j} \mu, P_j}\ (check)$$

$$\frac{j \in I \quad p_j = \tau\,(\tau'')}{\mu, \sum_{i \in I} p_i@a_i.P_i \xrightarrow{\tau\tau'@a_j} \mu, \{\tau'/\tau''\}P_j \parallel p_j@a_j.P_j}\ (lambda)$$

$$\frac{}{\mu, \langle \tau \copyright \tau' \rangle @a \xrightarrow{\langle \tau \copyright \tau' \rangle @a} \mu, \emptyset}\ (notify) \qquad \frac{}{\mu, P \xrightarrow{(\tau\tau'@a)} \mu, P \parallel \langle \tau \copyright \tau' \rangle @a}\ (async)$$

$$\frac{\mu \oplus (\tau' \times T), P \xrightarrow{\langle \tau \copyright \tau' \rangle @a} \mu', P' \quad T' = \mu'(\tau')}{\mu, \big(\nu\tau' : T\big)\, P \xrightarrow{\langle \tau \copyright (\tau' : T') \rangle @a} \mu' \ominus (\tau' \times T'), P'}\ (open)$$

$$\frac{\mu, P_1 \xrightarrow{\tau\tau'@a} \mu, P_1' \quad \mu, P_2 \xrightarrow{\langle \tau \copyright (\tau' : T) \rangle @a} \mu, P_2'}{\mu, P_1 \parallel P_2 \xrightarrow{\epsilon} \mu, \big(\nu\tau' : T\big)\, (P_1' \parallel P_2')}\ (close)$$

$$\frac{\mu \oplus (\tau \times T), P \xrightarrow{\alpha} \mu', P' \quad \tau \notin n(\alpha) \quad T' = \mu'(\tau)}{\mu, (\nu\tau : T)\, P \xrightarrow{\alpha} \mu' \ominus (\tau \times T'), \big(\nu\tau : T'\big)\, P'}\ (new)$$

$$\frac{\mu, P_1 \xrightarrow{\tau\tau'@a} \mu, P_1' \quad \mu, P_2 \xrightarrow{\langle \tau \copyright \tau' \rangle @a} \mu, P_2'}{\mu, P_1 \parallel P_2 \xrightarrow{\epsilon} \mu, P_1' \parallel P_2'}\ (com)$$

$$\frac{\mu, P \xrightarrow{\alpha} \mu', P'}{\mu, P \parallel P_1 \xrightarrow{\alpha} \mu', P' \parallel P_1}\ (par) \qquad \frac{\mu, P \equiv_n \mu_1, P_1 \xrightarrow{\alpha} \mu_2, P_2 \equiv_n \mu', P'}{\mu, P \xrightarrow{\alpha} \mu', P'}\ (struct)$$

Fig. 2. LTS semantics

3.1 Examples

To better highlight the main features of NCP, we introduce two illustrative examples.

3.1.1 NCP *hidden communications*

Let μ be a topic topology. The following NCP state describes the evolution of a component b raising an event having the same session of the one received by a:

$$\mu, \tau\left(\tau'\right)@a.skip.\overline{\tau_1}\tau'@b$$

Intuitively, this specification models a coordination policy where the component a can receive signals having topic τ. After some internal activity has taken place, the component b raises a signal having the same session (τ') of the one received by a.

This behavior constrains the components a and b exchange the name of session τ', however this communication is not explicitly represented.

The operational rules *lambda*, *skip* and *emit* details the required behaviour. The *lambda* rule handles the early instantiation of the input, allowing the transition for any name τ''. Notice that the lambda reaction remains active and that the envelopes spawned by the component b have the same session of the received one. The actual operational derivation is given below.

$$\mu, \tau\,(\tau')\,@a.skip.\overline{\tau_1}\tau'@b$$
$$\xrightarrow{\tau\tau''@a}\xrightarrow{\epsilon}\xrightarrow{\epsilon}$$
$$\mu, \tau\,(\tau')\,@a.skip.\overline{\tau_1}\tau'@b \parallel \textstyle\prod_{c\in G(b)(\tau_1)}\langle\tau_1 \copyright \tau''\rangle@c$$

3.1.2 `NCP` *scope of topology*

Let $\mu = \{(a, \tau, b)\})$ the topology describing a single connection from the component a to the component b for the topic τ. Let us consider the following `NCP` state.

$$\mu, \tau\,(\tau_1)\,@b.\overline{\tau_1}\tau_s@a \parallel (\left(\nu\tau' : \emptyset\right)\,\mathbf{fupd}(\{(\tau', b)\})@a.\langle\tau \copyright \tau'\rangle@b)$$

When the coordination begins, the topology for the topic τ' is hidden outside the right part of the parallel policy. The reception of the signal for the component b (by the lambda reaction) performs the extrusion of the name τ' within the topology. Hence, a can emit signals having τ' to the recipient b.

The behaviour described above is represented by the following operational derivation.

$$\cfrac{\cfrac{\begin{array}{c}\mu, \mathbf{fupd}(\{(\tau', b)\})@a.\langle\tau \copyright \tau'\rangle@b\\ \xrightarrow{\epsilon}\\ \mu \oplus \{(a, \tau', b)\}, \langle\tau \copyright \tau'\rangle@b\end{array}}{\begin{array}{c}\mu, (\nu\tau' : \emptyset)\,\mathbf{fupd}(\{(\tau', b)\})@a.\langle\tau \copyright \tau'\rangle@b\\ \xrightarrow{\epsilon}\\ \mu, (\nu\tau' : \{(a, b)\})\,\langle\tau \copyright \tau'\rangle@b\end{array}}}{\begin{array}{c}\mu, \tau\,(\tau_1)\,@b.\overline{\tau_1}\tau_s@a \parallel ((\nu\tau' : \emptyset)\,\mathbf{fupd}(\{(\tau', b)\})@a.\langle\tau \copyright \tau'\rangle@b)\\ \xrightarrow{\epsilon}\\ \mu, \tau\,(\tau_1)\,@b.\overline{\tau_1}\tau_s@a \parallel ((\nu\tau' : \{(a, b)\})\,\langle\tau \copyright \tau'\rangle@b)\end{array}}$$

Then the two parallel policies can communicate, extruding the τ' linkage:

$$\dfrac{\dfrac{}{\mu, \tau\,(\tau_1)\,@b.\overline{\tau_1}\tau_s@a \xrightarrow{\tau\tau'@b} \mu, \overline{\tau'}\tau_s@a} \qquad \dfrac{\mu \oplus \{(a,\tau',b)\}, \langle \tau \copyright \tau' \rangle@b \xrightarrow{\langle \tau \copyright \tau' \rangle@b} \mu \oplus \{(a,\tau',b)\}, \emptyset}{\mu, (\nu\tau' : \{(a,b)\})\, \langle \tau \copyright \tau' \rangle@b \xrightarrow{\langle \tau \copyright (\tau' : \{(a,b)\}) \rangle@b} \mu, \emptyset}}{\mu, \tau\,(\tau_1)\,@b.\overline{\tau_1}\tau_s@a \parallel ((\nu\tau' : \{(a,b)\})\, \langle \tau \copyright \tau' \rangle@b) \xrightarrow{\epsilon} \mu, (\nu\tau' : \{(a,b)\})\, (\overline{\tau'}\tau_s@a \parallel \emptyset)}$$

3.2 Bisimulation Semantics

To conclude this section, we introduce the notion of observational equivalence for NCP. Honda-Tokoro [12] and Amadio, *et ali* [1] have studied bisimilarity for asynchronous calculi. We use these results (in particular, the *directed HT labelled transition systems* defined in [1]) to define our bisimulation semantics.

Following these approaches, in the *bisimulation game*, any process can act as a "buffer" that reads any possible message and stores it without consuming the message. This is done, in our case, by rule *async*. On the other hand, "effective" inputs that actually consume messages are not observed at all in the bisimulation game, whereas synchronizations induced by these inputs are. Thus, in defining bisimilarity, we keep into account the transitions induced by the rule *async*, but not those obtained by *check* or *lambda*.

Definition 3.1 *Given two coordination policies P_1 and P_2, and two topic topologies μ_1 and μ_2, we say that (μ_1, P_1) is bisimilar to (μ_2, P_2), written $(\mu_1, P_1) \sim (\mu_2, P_2)$, if:*

- *For each transition $\mu_1, P_1 \xrightarrow{\alpha} \mu_1', P_1'$, with $\alpha \in \{\epsilon, \langle \tau \copyright \tau' \rangle@a, (\tau\tau'@a)\}$ there is a transition $\mu_2, P_2 \xrightarrow{\alpha} \mu_2', P_2'$ and $(\mu_1', P_1') \sim (\mu_2', P_2')$.*
- *For each transition $\mu_1, P_1 \xrightarrow{\langle \tau \copyright (\tau' : T) \rangle@a} \mu_1', P_1'$ with $\tau' \notin \mathrm{fn}(P_2)$, there is a transition $\mu_2, P_2 \xrightarrow{\langle \tau \copyright (\tau' : T') \rangle@a} \mu_2', P_2'$ and $(\mu_1', P_1') \sim (\mu_2', P_2')$.*

A key difference between NCP and the asynchronous π-calculus is the awareness of topic topologies in the semantics. However, it would be too restrictive to require that only policies with the same topology can be bisimilar. For example, the empty network is bisimilar to itself under *any* topology. This is also reflected in the definition of the clause for the bound output: when a bound output transition is matched in the bisimulation relation, the two hidden topologies associated to the transition are not taken in account, and, therefore, can be different.

4 Checking Choreography

In this section, we introduce a formal methodology to verify correctness of a network of `SC` components against global coordination policies as given by `NCP` specifications. The first step of our methodology consists of providing an encoding from `SC` networks to `NCP` policies. The basic idea of the encoding is to transform `SC` reactions into `NCP` transitions labelled with ϵ.

The encoding function $[\![B]\!]_a$ takes a `SC` behavior B, localized within the component a, and maps it into a `NCP` policy. This function is defined as follows:

$$[\![0]\!]_a = \emptyset \qquad\qquad [\![B \mid B']\!]_a = [\![B]\!]_a \parallel [\![B']\!]_a$$

$$[\![(\nu\tau)B]\!]_a = (\nu\tau : \emptyset)\, [\![B]\!]_a \qquad\qquad [\![\mathbf{out}\langle \tau \copyright \tau' \rangle.B]\!]_a = \overline{\tau}\tau'@a.[\![B]\!]_a$$

$$[\![\mathbf{rupd}\,(R)\,.B]\!]_a = skip.[\![R]\!]_a \parallel [\![B]\!]_a \qquad [\![\mathbf{fupd}(F)]\!]_a = \mathbf{fupd}(F)@a.[\![B]\!]_a$$

The function $[\![R]\!]_a$ takes a `SC` reaction R, installed in the interface of the component a, and maps it into a policy. The function is defined as follows:

$$[\![0]\!]_a = \emptyset \qquad\qquad [\![R|R']\!]_a = [\![R]\!]_a \parallel [\![R']\!]_a$$

$$[\![\tau \copyright \tau' \rightarrow B]\!]_a = \tau\tau'@a.[\![B]\!]_a \qquad [\![\tau \copyright \lambda\tau' \rightarrow B]\!]_a = \tau\,(\tau')\,@a.[\![B]\!]_a$$

Finally, the function $[\![N]\!]$ takes a `SC` network N and maps it into a `NCP` state. The function is defined as follows:

$$[\![\emptyset]\!] = \emptyset, \emptyset \qquad\qquad [\![\langle \tau \copyright \tau' \rangle @a]\!] = \emptyset, \langle \tau \copyright \tau' \rangle @a$$

$$\frac{[\![N]\!] = \mu, P \quad [\![N']\!] = \mu', P'}{[\![N \parallel N']\!] = \mu \cup \mu', P \parallel P'} \qquad \frac{[\![N]\!] = \mu, P \quad T = \mu(\tau)}{[\![(\nu\tau)N]\!] = \mu \ominus (\tau \times T), (\nu\tau : T)\, P}$$

$$[\![a[B]_F^R]\!] = \mu, [\![B]\!]_a \parallel [\![R]\!]_a \text{ where } \mu = \emptyset \oplus a \times F$$

The correctness of the encoding is "up-to" bisimilarity as shown by the following theorem.

Theorem 4.1 *Let N and N' be* `SC` *networks. It holds that $N \rightarrow N'$ if and only if $[\![N]\!] \xrightarrow{\epsilon} (\mu, P)$ and $(\mu, P) \sim [\![N']\!]$*

Proof (outline) *The proof is done by induction on the structure of* SC *networks ("if" side), and by induction on* SC *transition rules ("only if" side) and is quite straightforward. The case of parallel composition make use of a* weakening lemma *that shows compositionality of* NCP *bisimulation with respects to network contexts: if* $(\mu, P) \sim (\mu', P')$*, then for every graph* σ *we have* $(\mu \cup \sigma, P) \sim (\mu' \cup \sigma, P')$. □

The previous theorem allows us to derive the choreography model of a SC network. The next step of our methodology consists of making verification to be *compositional*. Once a choreography has been verified, it should be possible to "plug" it into a distributed network of components, without altering verified properties. This is formalized in the rest of this section.

First, we have to define SC network contexts. We use the notions of *occurrence* of a symbol in a term, and of substitution that can be defined in the standard way.

Definition 4.2 *The set* $\mathcal{C}$ *of one-hole* SC *network contexts is defined as the least subset of terms generated by the following grammar, where the number of occurrences of the placeholder* $*$ *is one.*

$$N ::= \emptyset \quad | \quad a[B]_F^R \quad | \quad N \parallel N \quad | \quad \langle \tau \copyright \tau' \rangle @ a \quad | \quad (\nu \tau) N \quad | \quad *$$

Assume that $C \in \mathcal{C}$*, and let* N *be a* SC *network. The application* $C[N]$ *of* C *to* N *is defined as the syntactic substitution of the single occurrence of* $*$ *in* C *with* N.

We have the following compositionality result.

Theorem 4.3 *Let* N_1 *and* N_2 *be* SC *networks such that* $[\![N_1]\!] \sim [\![N_2]\!]$*. For all* $C \in \mathcal{C}$*, it holds that* $[\![C[N_1]]\!] \sim [\![C[N_2]]\!]$.

Proof (outline) *The proof is done by induction on the structure of contexts, and by co-induction on the bisimulation relation. The only two possible kinds of one-hole contexts are* $(\nu\tau)*$ *and* $N \parallel *$*, for* N *network and* τ *topic name. The interesting one is the former, and in particular the case for the open rule in the coinduction scheme: there, the graph topology is affected. However, in both the reached states (by a bound output transition) the bisimulation relation is restricted to those observation whose subject is not the restricted name, thus it is preserved in one step, and then by coinduction we obtain the proof.* □

Putting the contents of this section together, we have a definition of satisfaction of a policy: let N be a SC network, P be a NCP policy and let μ denote a topic-driven topology. We say that N implements the choreography (μ, P) provided that $[\![N']\!] \sim (\mu, P)$. This is a semantical notion of satisfaction, since we define it up-to bisimulation, and can be mechanically checked in the finite-state case exploiting bisimulation-checking techniques such as those of [7].

5 Concluding remarks

In this paper we extended the SC & JSCL framework with a choreography model as specified by the NCP calculus. We presented an encoding from the design language (SC) to the choreography language in order to verify whether the SC network respects a global NCP policy. This is done via bisimulation checking.

Some research efforts have addressed the problem of relating choreography and orchestration. For instance, the notion of simulation conformance have been considered [4,14]. The methodological idea of providing separated languages to describe the global and local view of service coordination has been also considered in [5]. However, our framework introduces some new notions, like multi-layered networks and multi-party sessions.

Our long term goal is to provide modal logic and model checking algorithms, that be defined on the grounds of the labelled semantics of NCP. Additionally, we plan to implement and integrate these verification techniques within the JSCL design environment.

References

[1] Roberto M. Amadio, Ilaria Castellani, and Davide 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 *Lecture Notes in Computer Science*. Springer, 2007.

[3] Michele Boreale, Roberto Bruni, Luís Caires, Rocco De Nicola, Ivan Lanese, Michele Loreti, Francisco Martins, Ugo Montanari, António Ravara, Davide Sangiorgi, Vasco Thudichum Vasconcelos, and Gianluigi Zavattaro. Scc: A service centered calculus. In *WS-FM*, volume 4184 of *Lecture Notes in Computer Science*, pages 38–57. Springer, 2006.

[4] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and orchestration: A synergic approach for system design. In *ICSOC*, volume 3826 of *Lecture Notes in Computer Science*, pages 228–240. Springer, 2005.

[5] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centred programming for web services. In *ESOP 2007*, volume 4421 of *Lecture Notes in Computer Science*, pages 2–17. Springer, 2007.

[6] Gian Luigi Ferrari, Roberto Guanciale, and Daniele Strollo. Event based service coordination over dynamic and heterogeneous networks. In Asit Dan and Winfried Lamersdorf, editors, *ICSOC*, volume 4294 of *Lect. Notes in Comput. Sci.*, pages 453–458. Springer-Verlag, 2006.

[7] Gian Luigi Ferrari, Ugo Montanari, and Emilio Tuosto. Coalgebraic minimization of hd-automata for the pi-calculus using polymorphic types. *Theor. Comput. Sci.*, 331(2-3):325–365, 2005.

[8] Gianluigi Ferrari, Roberto Guanciale, and Daniele Strollo. Jscl: A middleware for service coordination. In *FORTE*, volume 4229 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2006.

[9] Gianluigi Ferrari, Roberto Guanciale, Daniele Strollo, and Emilio Tuosto. Coordination via types in an event-based framework. *27th IFIP WG 6.1 International Conference on Formal Methods for Networked and Distributed Systems FORTE*, 4574:66–80, 2007.

[10] David Gelernter. Generative communications in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.

[11] Claudio Guidi, Roberto Lucchi, Roberto Gorrieri, Nadia Busi, and Gianluigi Zavattaro. A calculus for service oriented computing. In *ICSOC*, volume 4294 of *Lecture Notes in Computer Science*, pages 327–338. Springer, 2006.

[12] Koei Honda and Mario Tokoro. An object calculus for asynchronous communication. *Lecture Notes in Computer Science*, 512:133–147, 1991.

[13] Alessandro Lapadula, Rosario Pugliese, and Francesco Tiezzi. A calculus for orchestration of web services. In *ESOP*, volume 4421 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2007.

[14] Jing Li, Huibiao Zhu, and Geguang Pu. Conformance validation between choreography and orchestration. In *First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering, TASE 2007*, pages 473–482. IEEE Computer Society, 2007.

[15] Ying Liu and Beth Plale. Survey of publish subscribe event systems. Technical Report TR574, Computer Science Department, Indiana University, 2003.

[16] Jayadev Misra. A programming model for the orchestration of web services. In *SEFM*, pages 2–11. IEEE Computer Society, 2004.

[17] OASIS Bpel Specifications. OASIS - BPEL. http://www.oasis-open.org/cover/bpel4ws.html.

[18] Davide Sangiorgi and David Walker. *The π-Calculus: a Theory of Mobile Processes*. Cambridge University Press, 2002.

[19] W3C. Web Services Choreography Description Language (v.1.0). Technical report.