Behavioural Theory for Session-Oriented Calculi

Ivan Lanese[1], Antonio Ravara[2], Hugo T. Vieira[2]

[1] Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy
lanese@cs.unibo.it
[2] CITI and Dep. of Informatics, FCT, Univ. Nova de Lisboa, Portugal
aravara,htv@fct.unl.pt

Abstract. This chapter presents the behavioural theory of some of the SENSORIA core calculi. We consider SSCC, μse and CC as representatives of the session-based approach and COWS as representative of the correlation-based one.
For SSCC, μse and CC the main point is the structure that the session/conversation mechanism creates in programs. We show how the differences between binary sessions, multiparty sessions and dynamic conversations are captured by different behavioural laws. We also exploit those laws for proving the correctness of program transformations.
For COWS the main point is that communication is prioritised (the best matching input captures the output), and this has a strong influence on the behavioural theory of COWS. In particular, we show that communication in COWS is neither purely synchronous nor purely asynchronous.

1 Introduction

In most formal languages, it is common to have several terms denoting the same computational process. To understand when this situation occurs, the language needs to be equipped with a notion of equivalence. This notion relies on an underlying description of the behaviour of such process.

In process calculi the behaviour of systems is usually defined in terms of either a reduction relation or of a labelled transition relation (also called labelled transition system, or LTS). The former describes the possible evolutions of a process in isolation; the latter allows to describe also the potential interactions with the environment (usually, the latter relation includes the former).

The most natural notion of term equivalence is behavioural indistinguishability in any possible context the term may occur in. In fact, it should be possible to replace one equivalent process for another without changing the observable behaviour of the system (or part of it). Then, such a relation guarantees the correctness of program transformations developed for, e.g., improving the global performance or to increase fault tolerance. In service oriented computing, equivalent services can serve the same purpose in a complex orchestration, thus the choice can be driven by their non-functional properties such as cost or performance. Such a notion is called a *contextual* equivalence, and distinguishability is based on a notion of observation. In sequential settings, one may define as

observables the values produced by a computation, or even simply termination; in concurrent settings the observables are usually descriptions of a process interaction potential.

Contextual equivalences are, however, difficult to use. They require an universal quantification over possible contexts (normally infinitely many), thus lacking a practical proof technique. A typical solution is to look for a co-inductive congruence relation which characterises it.

In the realm of process algebras, one can find a myriad of behavioural equivalence notions (van Glabeek presents an overview, interrelating several notions, in [Gla01]). Different equivalences (or pre-orders) take into account different ways of observing the behaviour of processes. Considering the most significant notions, trace equivalence [Hoa85] looks at the sequence of observable actions, (labelled) bisimilarity [Mil89] takes into account also points of choice, and testing equivalences [DH84] consider the interactions between the process and an observer.

Different equivalences can be useful for different purposes; however, two criteria are important: being a congruence, and, in particular, coinciding with the contextual equivalence (which is a congruence by definition). The former result means that given two equivalent processes, placing them in a language context will produce two processes that are again equivalent, thus allowing substituting "equals for equals" in context, not changing the global behaviour. The congruence result also testifies that all the constructs of a calculus may be soundly interpreted as compositional semantic operators on bisimilarity equivalence classes. If an equivalence coincides with a given contextual equivalence, it captures exactly the abstract semantics given by the chosen observables.

In mobile process calculi like the π-calculus [SW01b], standard contextual equivalences are barbed congruence [MS92] and barbed bisimilarity [HY95] (the basic observables are called barbs). Their co-inductive characterisation is based on the notion of *bisimilarity*. For instance, for barbed congruence, it is *full bisimilarity* (the substitution-closed ground bisimilarity) over a(n early) LTS. Bisimilarity is the largest bisimulation, the latter being a relation $\mathcal{R}$ such that if $(P, Q) \in \mathcal{R}$ then for each action of P there is a corresponding action of Q and the two actions lead to processes P' and Q' such that $(P', Q') \in \mathcal{R}$. Bisimilarity is a good tool for proving process equivalence, since it is naturally equipped with a proof technique: to prove that two processes P and Q are bisimilar it is enough to exhibit a bisimulation including the pair (P, Q). There are, in general, two notions of bisimulation: a strong one, taking into account also internal actions, and a weak one, abstracting them away. The latter is particularly interesting, since it allows to prove correctness of program optimisations (in fact equivalent processes need not perform the same number of internal actions).

In this chapter we apply bisimilarity notions to some of the SENSORIA core calculi, namely SSCC, μse, CC and COWS (see Chapter 2-1 for a description of SSCC, CC and COWS, and [BLMT08] for μse). The different features of the calculi have a strong impact on their behavioural theory.

$$
\begin{array}{llll}
P|\mathbf{0} \equiv P & P|Q \equiv Q|P & (P|Q)|R \equiv P|(Q|R) & \\
(\nu n)P|Q \equiv (\nu n)(P|Q) \quad \text{if } n \notin \mathrm{fn}(Q) & & r \rhd (\nu a)P \equiv (\nu a)(r \rhd P) & \\
\mathsf{stream}\,(\nu a)P\,\mathsf{as}\,f = \boldsymbol{v}\,\mathsf{in}\,Q \equiv (\nu a)(\mathsf{stream}\,P\,\mathsf{as}\,f = \boldsymbol{v}\,\mathsf{in}\,Q) & \text{if } a \notin \mathrm{fn}(Q) \cup \{\boldsymbol{v}\} & & \\
\mathsf{stream}\,P\,\mathsf{as}\,f = \boldsymbol{v}\,\mathsf{in}\,(\nu a)Q \equiv (\nu a)(\mathsf{stream}\,P\,\mathsf{as}\,f = \boldsymbol{v}\,\mathsf{in}\,Q) & \text{if } a \notin \mathrm{fn}(P) \cup \{\boldsymbol{v}\} & & \\
(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P & (\nu a)\mathbf{0} \equiv \mathbf{0} & \mathrm{rec}\ X.P \equiv P[^{\mathrm{rec}\ X.P}/_X] &
\end{array}
$$

Fig. 1. Structural congruence

For the session-based calculi, standard notions of equivalence can be used, and the main point is the characterisation of the communication structure of the processes. In fact, the different notions of binary session (for SSCC), multiparty session (for μse) and dynamic conversation (for CC) are captured by different behavioural equations. We discuss those laws, and exploit them to prove the correctness of different kinds of program transformations. For CC we also prove a normal form result.

Communication in COWS is based on the correlation set mechanism, which provides prioritised communication (the best matching input captures the output). The corresponding barbed bisimilarity is captured by a more complex form of bisimilarity with respect to the ones for the other calculi, and shows that communication in COWS is neither purely synchronous nor purely asynchronous.

2 Behavioural Theory for SSCC

In this section we study the behavioural theory of SSCC (Stream-based Service Centred Calculus [LMVR07]), and we apply it to prove the correctness of some program transformations.

We recall that SSCC is a calculus for modelling service oriented systems based on the concepts of services, binary sessions, and streams. The (static) syntax of SSCC has been defined in Chapter 2-1. We refer to this chapter also for an informal description of the operators, while presenting here the LTS semantics and the behavioural theory. We need as auxiliary operators to define the semantics also $r \lhd P$ for client-side session, $r \rhd P$ for service-side session, $(\nu r)P$ for session name restriction and $\mathsf{stream}\,P\,\mathsf{as}\,f = \boldsymbol{v}\,\mathsf{in}\,Q$ for stream with stored values. Processes are herein written in this extended (called run-time) syntax, and considered up to the structural congruence relation inductively defined by the rules in Figure 1. Note that structural congruence is included in bisimilarity (forthcoming Lemma 2).

Operational Semantics. The semantics of SSCC is defined using an LTS in the early style. This LTS is slightly different, but equivalent to its original presentation in [LMVR07]. We require processes to have no free process variables.

Definition 1 (SSCC Labelled Transition System). *The rules in Figure 2, together with the symmetric version of rules* L-PAR, L-PAR-CLOSE, *and* L-SESS-COM-CLOSE, *inductively define the LTS on processes.*

The LTS uses μ as a metavariable for labels. The bound names in labels are r in service definition activation and service invocation and a in extrusion labels (conventionally, they are all in parenthesis). Label $\uparrow v$ denotes sending value v. Dually, label $\downarrow v$ is receiving value v. We use $\updownarrow v$ to denote one of $\uparrow v$ or $\downarrow v$, and we assume that when multiple $\updownarrow v$ appear in the same rule they are instantiated in the same direction, and that $\overline{\updownarrow} v$ denotes the opposite direction. We use similar conventions for other labels.

Continuing with the labels, $a \Leftarrow (r)$ and $a \Rightarrow (r)$ denote respectively the request and the activation of a service, where a is the name of the service, and r is the name of the new session to be created. We use $a \Leftrightarrow (r)$ to denote one of $a \Leftarrow (r)$ or $a \Rightarrow (r)$. Furthermore, label $\Uparrow v$ denotes the feeding of value v into a stream, while label $f \Downarrow v$ reads value v from stream f. When an input or an output label crosses a session construct (rule L-SESS-VAL), we add to the label the name of the session and whether it is a server or a client session (for example, $\downarrow v$ may become $r \lhd \downarrow v$).

The label denoting a conversation step in a free session r is $r\tau$, and if the value passed in the session channel is private, it remains private in the resulting process. A label τ is obtained only when r is restricted (rule L-SESS-RES). Thus a τ action can be obtained in four cases: a communication inside a restricted session, a service invocation, a feed or a read from a stream. Note also that we have two contexts causing interaction: parallel composition and stream. Finally, bound actions, $(a)\mu$, represent the extrusion of a in their respective free counterparts μ.

Some processes, such as $r \lhd r \lhd P$, can be written using the run-time syntax, but they are not reachable from processes in the static syntax. We consider these processes ill-formed, and will not consider them anymore.

Bisimilarity. We study the usual notions of strong and weak bisimilarity. Both are non-input congruences in the class of SSCC processes. One can get a congruence by considering (strong or weak) full bisimilarity, *i.e.,* by closing bisimilarity with respect to service name substitutions (there is no reason to close with respect to session or stream names, since no substitutions are performed on them). Although the general strategy is the same as for the π-calculus, the proof techniques themselves differ significantly. Herein we only present the main results. Detailed proofs can be found in [CF+07].

To define weak bisimilarity we introduce some abbreviations: let $P \stackrel{\tau}{\Rightarrow} Q$ denote $P(\stackrel{\tau}{\rightarrow})^n Q$ (with $n \geq 0$, *i.e.,* zero or more transitions) and let $P \stackrel{\alpha}{\Rightarrow} Q$ denote $P \stackrel{\tau}{\Rightarrow} \stackrel{\alpha}{\rightarrow} \stackrel{\tau}{\Rightarrow} Q$ for $\alpha \neq \tau$.

Definition 2 (Strong and Weak Bisimilarity). *A symmetric binary relation $\mathcal{R}$ on processes is a* (strong) bisimulation *if, for any processes P, Q such that $P \mathcal{R} Q$, if $P \stackrel{\alpha}{\rightarrow} P'$ with $\mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset$, then there exists Q' such that $Q \stackrel{\alpha}{\rightarrow} Q'$*

$$v.P \xrightarrow{\uparrow v} P \qquad (x)P \xrightarrow{\downarrow v} P[v/x] \qquad \mathsf{feed}\, v.P \xrightarrow{\Uparrow v} P \qquad f(x).P \xrightarrow{f \Downarrow v} P[v/x]$$

(L-SEND, L-RECEIVE, L-FEED, L-READ)

$$\frac{P \xrightarrow{\mu} P' \quad \mu \neq \Uparrow v \quad \mathrm{bn}(\mu) \cap (\mathrm{fn}(Q) \cup \{\boldsymbol{w}\}) = \emptyset}{\mathsf{stream}\, P\, \mathsf{as}\, f = \boldsymbol{w}\, \mathsf{in}\, Q \xrightarrow{\mu} \mathsf{stream}\, P'\, \mathsf{as}\, f = \boldsymbol{w}\, \mathsf{in}\, Q} \quad \text{(L-STREAM-PASS-P)}$$

$$\frac{Q \xrightarrow{\mu} Q' \quad \mu \neq f \Downarrow v \quad \mathrm{bn}(\mu) \cap (\mathrm{fn}(P) \cup \{\boldsymbol{w}\}) = \emptyset}{\mathsf{stream}\, P\, \mathsf{as}\, f = \boldsymbol{w}\, \mathsf{in}\, Q \xrightarrow{\mu} \mathsf{stream}\, P\, \mathsf{as}\, f = \boldsymbol{w}\, \mathsf{in}\, Q'} \quad \text{(L-STREAM-PASS-Q)}$$

$$\frac{P \xrightarrow{\Uparrow v} P'}{\mathsf{stream}\, P\, \mathsf{as}\, f = \boldsymbol{w}\, \mathsf{in}\, Q \xrightarrow{\tau} \mathsf{stream}\, P'\, \mathsf{as}\, f = v :: \boldsymbol{w}\, \mathsf{in}\, Q} \quad \text{(L-STREAM-FEED)}$$

$$\frac{Q \xrightarrow{f \Downarrow v} Q'}{\mathsf{stream}\, P\, \mathsf{as}\, f = \boldsymbol{w} :: v\, \mathsf{in}\, Q \xrightarrow{\tau} \mathsf{stream}\, P\, \mathsf{as}\, f = \boldsymbol{w}\, \mathsf{in}\, Q'} \quad \text{(L-STREAM-CONS)}$$

$$\frac{r \notin \mathrm{fn}(P)}{a \Leftarrow P \xrightarrow{a \Leftarrow (r)} r \lhd P} \qquad \frac{r \notin \mathrm{fn}(P)}{a \Rightarrow P \xrightarrow{a \Rightarrow (r)} r \rhd P} \quad \text{(L-CALL, L-DEF)}$$

$$\frac{P \xrightarrow{\mu} P' \quad \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset}{P|Q \xrightarrow{\mu} P'|Q} \qquad \frac{P \xrightarrow{\updownarrow v} P'}{r \bowtie P \xrightarrow{r \bowtie \updownarrow v} r \bowtie P'}$$

(L-PAR, L-SESS-VAL)

$$\frac{P[\mathrm{rec}\ X.P/X] \xrightarrow{\mu} P'}{\mathrm{rec}\ X.P \xrightarrow{\mu} P'} \qquad \frac{P \xrightarrow{r \bowtie \uparrow v} P' \quad Q \xrightarrow{r \overline{\bowtie} \downarrow v} Q'}{\mathsf{stream}\, P\, \mathsf{as}\, f = \boldsymbol{w}\, \mathsf{in}\, Q \xrightarrow{r\tau} \mathsf{stream}\, P'\, \mathsf{as}\, f = \boldsymbol{w}\, \mathsf{in}\, Q'}$$

(L-REC, L-SESS-COM-STREAM)

$$\frac{P \xrightarrow{a \Leftrightarrow (r)} P' \quad Q \xrightarrow{a \overline{\Leftrightarrow} (r)} Q'}{\mathsf{stream}\, P\, \mathsf{as}\, f = \boldsymbol{w}\, \mathsf{in}\, Q \xrightarrow{\tau} (\nu r)\mathsf{stream}\, P'\, \mathsf{as}\, f = \boldsymbol{w}\, \mathsf{in}\, Q'} \quad \text{(L-SERV-COM-STREAM)}$$

$$\frac{P \xrightarrow{r \bowtie \uparrow v} P' \quad Q \xrightarrow{r \overline{\bowtie} \downarrow v} Q'}{P|Q \xrightarrow{r\tau} P'|Q'} \qquad \frac{P \xrightarrow{a \Leftrightarrow (r)} P' \quad Q \xrightarrow{a \overline{\Leftrightarrow} (r)} Q'}{P|Q \xrightarrow{\tau} (\nu r)(P'|Q')}$$

(L-SESS-COM-PAR, L-SERV-COM-PAR)

$$\frac{P \xrightarrow{\mu} P' \quad n \notin \mathrm{n}(\mu)}{(\nu n)P \xrightarrow{\mu} (\nu n)P'} \qquad \frac{P \xrightarrow{r\tau} P'}{(\nu r)P \xrightarrow{\tau} (\nu r)P'} \qquad \frac{P \xrightarrow{\mu} P' \quad \mu \neq \updownarrow v \quad r \notin \mathrm{bn}(\mu)}{r \bowtie P \xrightarrow{\mu} r \bowtie P'}$$

(L-RES,L-SESS-RES, L-SESS-PASS)

$$\frac{P \xrightarrow{r \bowtie (v) \uparrow v} P' \quad Q \xrightarrow{r \overline{\bowtie} \downarrow v} Q' \quad v \notin \mathrm{fn}(Q)}{P|Q \xrightarrow{r\tau} (\nu v)(P'|Q')} \qquad \frac{P \xrightarrow{\mu} P' \quad \mu \in \{\uparrow a, r \bowtie \uparrow a, \Uparrow a\}}{(\nu a)P \xrightarrow{(a)\mu} P'}$$

(L-PAR-CLOSE, L-EXTR)

$$\frac{P \xrightarrow{r \bowtie (v) \uparrow v} P' \quad Q \xrightarrow{r \overline{\bowtie} \downarrow v} Q' \quad v \notin \mathrm{fn}(Q) \cup \{\boldsymbol{w}\}}{\mathsf{stream}\, P\, \mathsf{as}\, f = \boldsymbol{w}\, \mathsf{in}\, Q \xrightarrow{r\tau} (\nu v)\mathsf{stream}\, P'\, \mathsf{as}\, f = \boldsymbol{w}\, \mathsf{in}\, Q'} \quad \text{(L-SESS-COM-CLOSE)}$$

$$\frac{P \xrightarrow{(v) \Uparrow v} P' \quad v \notin \mathrm{fn}(Q) \cup \{\boldsymbol{w}\}}{\mathsf{stream}\, P\, \mathsf{as}\, f = \boldsymbol{w}\, \mathsf{in}\, Q \xrightarrow{\tau} (\nu v)\mathsf{stream}\, P'\, \mathsf{as}\, f = v :: \boldsymbol{w}\, \mathsf{in}\, Q} \quad \text{(L-FEED-CLOSE)}$$

Fig. 2. SSCC Labelled Transition System

and $P' \mathcal{R} Q'$. (Strong) bisimilarity $\sim$ is the largest bisimulation. Two processes P and Q are (strong) bisimilar *if $P \sim Q$.*

Weak bisimilarity $\approx$ is like the strong version, but using weak transitions $P \stackrel{\alpha}{\Rightarrow} Q$ instead of strong transitions $P \stackrel{\alpha}{\rightarrow} Q$.

Also, a full strong (resp. weak) bisimulation is a strong (resp. weak) bisimulation closed under service name substitutions, and we call full strong (resp. weak) bisimilarity $\sim_{\mathrm{f}}$ (resp. $\approx_{\mathrm{f}}$) the largest full strong (resp. weak) bisimulation.

Note that bisimilarity (respectively full bisimilarity) can be obtained as the union of all bisimulations (respectively full bisimulations). Moreover, as desired, structurally congruent processes (cf. Figure 1) are strong bisimilar.

Lemma 1 (Harmony Lemma). *Let P and Q be processes with $P \equiv Q$. If $P \stackrel{\alpha}{\rightarrow} P'$, then $Q \stackrel{\alpha}{\rightarrow} Q'$ with $P' \equiv Q'$, and vice-versa.*

Lemma 2. *Structurally congruent processes are full bisimilar.*

As in the π-calculus, strong and weak full bisimilarity are congruences, *i.e.*, they are closed under arbitrary contexts.

Theorem 1. *Strong and weak full bisimilarity are congruences.*

Useful Axioms. Even if presenting a complete axiomatisation for such a complex calculus is out of the scope of this chapter, we present here some axioms (equational laws correct with respect to strong/weak full bisimilarity) that capture key facts about the behaviour of processes. Some of them are useful to prove the correctness of the transformations presented in the following. To show the axioms we need to define contexts.

An n-ary context is a process where n subterms have been replaced by symbols $\bullet_1, \ldots, \bullet_n$. The application $\mathcal{C}[\![P_1, \ldots, P_n]\!]$ of context $\mathcal{C}[\![\bullet_1, \ldots, \bullet_n]\!]$ to processes $P_1, \ldots, P_n$ is the process obtained by replacing $\bullet_i$ with P_i.

The correctness of the axioms below can be proved by considering as full bisimulation all the instances of the equations together with the identity.

Session Garbage Collection

$$(\nu \mathsf{r})\mathcal{D}[\![\mathsf{r} \triangleright \mathbf{0}, \mathsf{r} \triangleleft \mathbf{0}]\!] \sim_{\mathrm{f}} \mathcal{D}[\![\mathbf{0}, \mathbf{0}]\!] \qquad \text{where } \mathcal{D} \text{ does not bind } r \qquad (1)$$

Stream Garbage Collection

$$\mathsf{stream}\,\mathbf{0}\,\mathsf{as}\,f\,\mathsf{in}\,P \sim_{\mathrm{f}} P \qquad \text{if } f \text{ does not occur in } P \qquad (2)$$

Session Independence

$$r \bowtie \mathsf{Q} \mid s \bowtie \mathsf{P} \sim_{\mathrm{f}} r \bowtie (s \bowtie \mathsf{Q} \mid \mathsf{P}) \qquad \text{if } s \neq r \qquad (3)$$

The same holds if the two sessions have opposite polarities.

Stream Independence

$$\begin{aligned}&\mathsf{stream}\,P\,\mathsf{as}\,f\,\mathsf{in}\,(\mathsf{stream}\,P'\,\mathsf{as}\,g\,\mathsf{in}\,Q)\sim_{\mathrm{f}}\\&\qquad\mathsf{stream}\,P'\,\mathsf{as}\,g\,\mathsf{in}\,(\mathsf{stream}\,P\,\mathsf{as}\,f\,\mathsf{in}\,Q)\qquad\text{if } f\neq g\end{aligned}\tag{4}$$

Streams are Orthogonal to Sessions

$$r\bowtie(\mathsf{feed}\,v\mid P)\sim_{\mathrm{f}}\mathsf{feed}\,v\mid r\bowtie P\tag{5}$$

Stream Locality

$$\mathsf{stream}\,P\,\mathsf{as}\,f\,\mathsf{in}\,(Q\mid Q')\sim_{\mathrm{f}}(\mathsf{stream}\,P\,\mathsf{as}\,f\,\mathsf{in}\,Q)\mid Q'\quad\text{if } f\notin\mathrm{fn}(Q')\tag{6}$$

Parallel Composition Versus Streams

$$\mathsf{stream}\,P\,\mathsf{as}\,f\,\mathsf{in}\,Q\sim_{\mathrm{f}}P|Q\quad\text{if } f\notin\mathrm{fn}(Q)\text{ and }P\text{ does not contain }\mathsf{feed}\tag{7}$$

Interestingly the Session Independence law is strongly dependent on the available operators, and fails in similar calculi such as [B$^+$06,BBDL08]. This captures the fact that in SSCC session nesting is immaterial.

From object-oriented to service-oriented models. We apply now the behavioural theory developed so far to bridge the gap between traditional object-oriented and SENSORIA service-oriented models, so to allow the reuse of existing tools and techniques. We detail a model transformation procedure from a common object-oriented communication pattern into a session-based, service-oriented one, and prove it correct with respect to weak full bisimilarity. Since we have also proved that weak full bisimilarity is a congruence, the behaviour of every composition built exploiting these services remains unchanged when moving from the original programs into their transformed versions.

UML Sequence Diagrams [Amb04] (SDs) describe the exchange of messages among the components in a complex system. We present here a typical SD and show how it can be implemented in SSCC by exploiting suitable macros. We then show how subsessions can be used to simplify the implementation.

Object-Oriented View. The SD on the left of Figure 3 describes a common pattern appearing in scenarios involving (at least) three partners. The description of the communication pattern is as follows.

> Object B receives from object A the value w and forwards it (or a value computed from it) to object C. After receiving the value, object C answers with a value w'. Object B replies with v and finally object C replies with value v'. Then, object B forwards it to object A.

Note that "Object B receives from object A the value w" means that object A invokes a method in object B passing the value w. We can imagine for instance that A is the user of the credit portal in the financial case study (see Chapter 0-2), which invokes the credit portal itself (B). The credit portal then interacts with the rating system (C), possibly exchanging different pieces of information. The final answer is then sent back to the user.

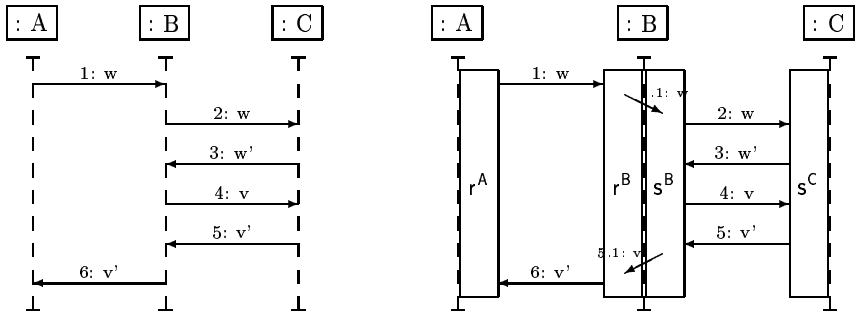

Fig. 3. Sequence diagram communication pattern: object-centred and session-centred view.

Session-Centred View. We want to move to a scenario where components are clients and servers of a service-oriented architecture, and where communication happens via sessions. We refine the diagram by incorporating information about the running sessions, in the diagram on the right of Figure 3, where the slanted arrows mean message passing between sessions. An instance of the credit portal B (let us call these instances *participants*) has a session r running with an instance of client A and another session s running with an instance of the rating system service C. Since sessions involve two partners, a session r between instances of A and B has two sides—called endpoints, r^{A} at the instance of A and r^{B} at the instance of B.

In addition to the normal constructs in SSCC, to model object-oriented systems (that do not follow the laws of session communication), it is useful to have two constructs enabling arbitrary message passing. These can be expressed by exploiting fresh auxiliary services.

$$b \Uparrow \langle v_1, ..., v_n \rangle.P \triangleq \mathsf{stream}\, b \Leftarrow v_1 ... v_n.\mathsf{feed\ unit\ as}\ f\ \mathsf{in}\ f(v).P$$

$$b \Downarrow (x_1, ..., x_n)P \triangleq \mathsf{stream}\ b \Rightarrow (z_1)...(z_n).\mathsf{feed}\ z_1 ... \mathsf{feed}\ z_n\ \mathsf{as}\ f \\ \mathsf{in}\ f(x_1)...f(x_n).P$$

where name v and stream f are not used in P and unit is a value used for synchronisation.

The diagram on the right of Figure 3 is directly implemented in SSCC as

$$\mathsf{SC} \triangleq (\nu b, c)\ (\ \mathsf{A} \mid \mathsf{B} \mid \mathsf{C}\ )$$

where

$$\mathsf{A} \triangleq b \Leftarrow w.(y)P,\ \mathsf{B} \triangleq (\nu b_1, b_2)\ (\ \mathsf{B}_1 \mid \mathsf{B}_2\ ),\ \text{and}\ \mathsf{C} \triangleq c \Rightarrow (x)w'.(y)v'.S,$$

$$\mathsf{B}_1 \triangleq b \Rightarrow (x)b_1 \Uparrow x.b_2 \Downarrow (y)y.Q,\ \text{and}\ \mathsf{B}_2 \triangleq c \Leftarrow b_1 \Downarrow (x)x.(z)v.(y)b_2 \Uparrow y.R.$$

It is easy to check that the behaviour of the process SC above reflects the one described on the right of Figure 3.

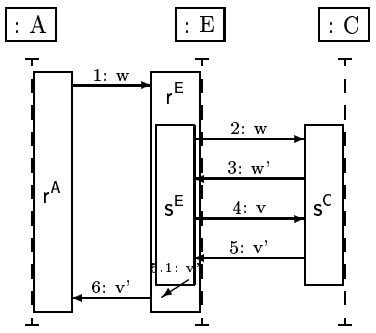

Fig. 4. Sequence diagram: using a subsession.

An Optimisation. When the credit portal B has the value sent by client A, it may immediately send it to the rating system C, calling it (and thus opening a subsession). One simply has to perform a "local" transformation on B. The resulting diagram is in Figure 4, and it is implemented in SSCC as process SC′, where we denote by E the new instance of B.

$$\mathrm{SC}' \triangleq (\nu b, c)\ (\ \mathsf{A} \mid \mathsf{E} \mid \mathsf{C}\ ) \qquad \text{where}$$
$$\mathsf{E} \triangleq b \Rightarrow (x)(\nu b_1)(c \Leftarrow x.(z)v.(y)b_1 \Uparrow y.R \mid b_1 \Downarrow (y)y.Q)$$

Naturally, one asks whether the transformations of SC into SC′ is correct, not changing the observable behaviour of processes. Indeed, this is the case, and this can be proved using the definition of full weak bisimilarity and the axioms presented before.

The correctness of the transformation, *i.e.*, $\mathrm{SC} \approx_{\mathrm{f}} \mathrm{SC}'$, follows from closure under contexts from the following equation.

$$(\nu \mathsf{c})(\mathsf{B} \mid \mathsf{C}) \ \approx_{\mathrm{f}}\ (\nu \mathsf{c})(\mathsf{E} \mid \mathsf{C}) \tag{8}$$

We sketch the correctness proof, while referring to [CF+08] for more details and for other examples of transformations, e.g. replacing auxiliary services with stream-based communications.

Proof (of Equation 8). The proof can be obtained by exhibiting a bisimulation including the two processes. The two processes can mimic each other even if the first one is non deterministic, since the nondeterminism comes from τ steps, whose order is not important, since the processes are confluent. Garbage collection Equations 1 and 2 are used in the proof. After some steps, the two processes have evolved to:

$$(\nu \mathsf{s})(r \rhd \mathsf{Q}[w/x][v'/y] \mid s \lhd \mathsf{R}[w/x][w'/z][v'/y] \mid s \rhd \mathsf{S}[w/x][v/y])$$
$$(\nu \mathsf{s})(r \rhd (s \lhd \mathsf{R}[w/x][w'/z][v'/y]) \mid \mathsf{Q}[w/x][v'/y] \mid s \rhd \mathsf{S}[w/x][v/y])$$

respectively. These processes can be proved equivalent using structural congruence (which is included in full bisimilarity, according to Lemma 2), session independence (Equation 3) and closure under contexts.

S, T	$::=$ $l :: a \Rightarrow P$	Service definition	$\mid$	$l :: P$	Located process
	$\mid$ $S \mid T$	Composition of systems	$\mid$	$(\nu n)S$	New name
P, Q	$::=$ $\mathbf{0}$	Empty process			
	$\mid$ $\overline{x}w.P$	Intra-session output	$\mid$	$x(y).P$	Intra-session input
	$\mid$ $x!w.P$	Intra-location output	$\mid$	$x?(y).P$	Intra-location input
	$\mid$ $\mathsf{install}[a \Rightarrow P].Q$	Service installation	$\mid$	$\mathsf{invoke}\ a.P$	Service invocation
	$\mid$ $\mathsf{merge}^p\ e.P$	Entry point	$\mid$	$r \triangleright P$	Endpoint
	$\mid$ $P \mid Q$	Parallel composition	$\mid$	$(\nu n)P$	New name
	$\mid$ $\mathrm{rec}\ X.P$	Recursive process	$\mid$	X	Recursive call

Fig. 5. Syntax of μse systems and processes

3 From Binary to Multiparty Sessions

In this section we apply the notions of bisimilarity from Definition 2 to μse, a name passing calculus for programming dynamic multiparty sessions proposed in [BLMT08]. Multiparty sessions extend the idea of sessions to multiparty communications, and have been recently object of deep study in the field of service oriented computing [BC08,CHY07,HYC08], since they provide a natural framework to describe the complex interactions among services. The distinctive feature of μse is its ability to dynamically create sessions and merging different existing sessions.

We describe now the syntax and the operational semantics of μse, so to allow to apply bisimilarity in this setting. μse is a calculus featuring names for:

- multiparty sessions (ranged by $r, s, \ldots$),
- services (ranged by $a, b, \ldots$), able to enter sessions upon invocation,
- channels (ranged by $x, y, \ldots$), to route messages inside sessions,
- entry points (ranged by $e, f, \ldots$), allowing to merge running sessions,
- locations (ranged by $l, \ldots$), where services and sessions are located.

Channels, services and entry points are *communicable values* (ranged over by $v, w, \ldots$) while sessions and locations cannot be communicated. We let $n, m, \ldots$ range over all names but locations.

The syntax of μse is defined in Figure 5. Systems (ranged over by $S, T, \ldots$) are parallel compositions of *locations* where services are published and processes executed. A location where a service a is defined is meant to be the domain into which all instances of a are executed.

A μse process can be the empty process, a process prefixed by an action, a process running in a session (endpoint), the parallel composition of processes, a process under a name restriction, a recursive process or a recursive invocation.

Processes (ranged over by $P, Q, \ldots$) communicate via channels according to two modalities: *intra-session* and *intra-location*. Intra-session communications are used to let different endpoints of the same session to interact regardless their

$$\mathcal{A}|\mathcal{A}' \equiv \mathcal{A}'|\mathcal{A} \qquad \mathcal{A}|\mathbf{0} \equiv \mathcal{A} \qquad (\mathcal{A}|\mathcal{A}')|\mathcal{A}'' \equiv \mathcal{A}|(\mathcal{A}'|\mathcal{A}'')$$

$$(\nu n)(\mathcal{A}|\mathcal{A}'') \equiv \mathcal{A}|(\nu n)\mathcal{A}'', \text{ if } n \notin \mathrm{fn}(\mathcal{A})$$

$$(\nu n)(\nu m)\mathcal{A} \equiv (\nu m)(\nu n)\mathcal{A} \qquad (\nu n)\mathcal{A} \equiv \mathcal{A}, \text{ if } n \notin \mathrm{fn}(\mathcal{A})$$

$$l :: P|l :: Q \equiv l :: (P|Q) \qquad l :: (\nu n)P \equiv (\nu n)(l :: P)$$

$$r \rhd (\nu n)P \equiv (\nu n)(r \rhd P), \text{ if } n \neq r \qquad \mathrm{rec}\ X.P \equiv P\{\mathrm{rec}\ X.P/X\}$$

$$r \doteq r \equiv \mathbf{0} \qquad (\nu r)(r \doteq s) \equiv \mathbf{0} \qquad r \doteq s|P \equiv r \doteq s|P\{r/s\} \qquad r \doteq s \equiv s \doteq r$$

$$r \rhd (s \doteq t|P) \equiv s \doteq t|r \rhd P \qquad l :: (r \doteq s|P) \equiv r \doteq s|l :: P$$

Fig. 6. μse structural congruence

running locations. Conversely, intra-location communications allow endpoints (of possibly different sessions) to communicate, provided that they are running in the same location. This is used to model local communications and replaces SSCC streams.

Processes can install new service definitions in their running locations. Service invocations enable processes to activate new endpoints on the service location. Service invocation requires only the service name, not its location, thus if many services with the same name are available one of them is chosen nondeterministically. Finally, the prefix $\mathsf{merge}^p\ e$ is a mechanism for merging existing sessions.

The operational semantics of μse requires a structural congruence relation and an extended syntax, namely *explicit substitutions* $r \doteq s$ of sessions. Let $\mathcal{A}, \mathcal{B}$ range over systems (including explicit substitutions) and processes. The structural congruence relation is defined in Figure 6. This exploits the usual notions of free and bound names: the occurrences of y and n are bound in $x(y).P$, $x?(y).P$, $(\nu n)P$ and $(\nu n)S$. Bound names can be safely alpha renamed.

Structural congruence $\equiv$ includes associativity, commutativity and identity over $\mathbf{0}$ for parallel composition and rules for scope extrusion. Also, $\equiv$ gives the semantics of recursion and $r \doteq s$ in terms of substitutions. Note that any explicit substitution $r \doteq s$ is persistent and can freely "float" in the term structure, unless a restriction on r or s forbids its movements.

The operational semantics of μse is specified through an LTS defined on terms up to structural congruence (thus lemmas corresponding to Lemmas 1 and 2 hold by definition). We use α to range over labels. Bound variables occurring in labels are in round parentheses.

Definition 3 (μse Labeled Transition System). *The μse LTS is the least relation generated by the rules in Figure 7, closed under structural congruence.*

The rules for prefixes simply execute them, moving the information to the transition label. As usual for early semantics, input prefixes guess the actual value and immediately substitute it for the formal variable. Sessions are transparent to

$$\overline{x}v.P \xrightarrow{\overline{x}v} P \qquad x!v.P \xrightarrow{x!v} P \qquad x(y).P \xrightarrow{xv} P\{v/y\} \qquad x?(y).P \xrightarrow{x?v} P\{v/y\}$$

$$l :: a \Rightarrow P \xrightarrow{r \top a} l :: r \rhd P \qquad \mathsf{invoke}\ a.P \xrightarrow{\perp a} P \qquad \mathsf{install}[a \Rightarrow R].P \xrightarrow{a[R]} P$$

$$\mathsf{merge}^p\ e.P \xrightarrow{e^p} P$$

$$\frac{P \xrightarrow{\alpha} Q \quad \alpha \in \{\perp a, xv, \overline{x}v, e^p\}}{r \rhd P \xrightarrow{r\ \alpha} r \rhd Q} \qquad \frac{P \xrightarrow{\alpha} Q \quad \alpha \notin \{\perp a, xv, \overline{x}v, e^p\}}{r \rhd P \xrightarrow{\alpha} r \rhd Q}$$

$$\frac{P \xrightarrow{a[R]} Q}{l :: P \xrightarrow{\tau} l :: Q \mid l :: a \Rightarrow R} \qquad \frac{P \xrightarrow{\alpha} Q \quad \alpha \notin \{a[R], x?(v), x!v\}}{l :: P \xrightarrow{\alpha} l :: Q}$$

$$\frac{P \xrightarrow{x!v} P' \quad Q \xrightarrow{x?v} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \qquad \frac{\mathcal{A} \xrightarrow{\alpha} \mathcal{A}' \quad \mathrm{bn}(\alpha) \cap \mathrm{fn}(\mathcal{B}) = \emptyset}{\mathcal{A}|\mathcal{B} \xrightarrow{\alpha} \mathcal{A}'|\mathcal{B}} \qquad \frac{\mathcal{A} \xrightarrow{r\ \overline{x}v} \mathcal{A}' \quad \mathcal{B} \xrightarrow{r\ xv} \mathcal{B}'}{\mathcal{A}|\mathcal{B} \xrightarrow{\tau} \mathcal{A}'|\mathcal{B}'}$$

$$\frac{\mathcal{A} \xrightarrow{re^+} \mathcal{A}' \quad \mathcal{B} \xrightarrow{se^-} \mathcal{B}'}{\mathcal{A}|\mathcal{B} \xrightarrow{\tau} \mathcal{A}'|\mathcal{B}'|s \doteq r} \qquad \frac{S \xrightarrow{r \top a} S' \quad T \xrightarrow{r \perp a} T'}{S|T \xrightarrow{\tau} S'|T'}$$

$$\frac{\mathcal{A} \xrightarrow{\alpha} \mathcal{A}' \quad n \notin \mathrm{n}(\alpha)}{(\nu n)\mathcal{A} \xrightarrow{\alpha} (\nu n)\mathcal{A}'} \qquad \frac{\mathcal{A} \xrightarrow{\alpha} \mathcal{A}' \quad \alpha \in \{\overline{x}w, x!w, r\ \overline{x}w, r\ x!w\}}{(\nu w)\mathcal{A} \xrightarrow{(w)\alpha} \mathcal{A}'}$$

Fig. 7. μse operational semantics

most of the actions, while a session name is added to the label in case of session-dependent actions (intra-session communications, invoke and merge). Only the name of the innermost session is added. Service definitions can produce sessions, and the session name is guessed in the early style. Install requests are executed when the level of locations is reached. Observe that locations are transparent to all actions but install and intra-location communications. Also, most of the synchronisation rules can be applied both at the process and at the system level. The only exceptions are (i) intra-location communication, which is meaningful only at the process level, and (ii) service invocation, which can be stated only at the system level since definitions are always at the top level. Finally, restriction is dealt with using structural congruence, but the rule for extrusions is necessary for interactions with the environment (and notably for bisimulation).

To prove equivalences of μse processes we use the notion of weak bisimilarity, defined as for SSCC (see Definition 2), but using μse LTS. It is not easy to prove that μse bisimilarity is a congruence, since service installation makes it a (partially) higher order calculus, and proving congruence for higher-order calculi is an hard problem. However, standard techniques can be used to show that full bisimilarity is closed under parallel composition, which is the most used composition operation.

As for SSCC, we show here a few axioms for reasoning on μse processes.

Session Garbage Collection

$$r \rhd \mathbf{0} \sim_{\mathrm{f}} \mathbf{0} \tag{9}$$

Session Independence

$$r \rhd \mathsf{Q} \mid s \rhd \mathsf{P} \sim_{\mathrm{f}} r \rhd (s \rhd \mathsf{Q} \mid \mathsf{P}) \tag{10}$$

Location Garbage Collection

$$l :: \mathbf{0} | \mathcal{A} \sim_{\mathrm{f}} \mathcal{A} \tag{11}$$

Intra-Session Communication is Orthogonal w.r.t. Locations

$$l :: \overline{x}w \sim_{\mathrm{f}} l' :: \overline{x}w \tag{12}$$

Intra-Location Communication is Orthogonal w.r.t. Sessions

$$r \rhd x!w \sim_{\mathrm{f}} r' \rhd x!w \tag{13}$$

The axioms concerning sessions are simpler than in SSCC since there is no need to preserve the invariant that sessions have exactly two partners. Note that also in μse session independence hold, i.e. session nesting is immaterial.

We show here how to use bisimilarity to analyse properties of services and multiparty sessions, in particular to prove that an implementation of a service is compliant (i.e., bisimilar) to a more abstract specification.

Let us consider the service *CalculateRating* from the credit request scenario. We can write the specification in μse as:

$$l :: *CalculateRating \Rightarrow P \quad \text{with } P = data(user).some\_comp.\overline{ret}\ rating \tag{14}$$

The symbol $*$ preceding the service definition means that the service is persistent (this can be programmed using recursion). Also, *some_comp* in P denotes some sequence of actions computing the actual rating, e.g. interacting with some local database.

This service is deterministic: once invoked, it waits until receiving a value in channel *data*, then performs *some_comp*, and finally, it sends the *rating* on channel *ret*. *CalculateRating* may be computationally expensive, so different requests can be served using replicated services. The following implementation asks another service $Calc_i$ non-deterministically chosen from a pool $Calc_1, \ldots, Calc_n$ to do the job:

$$l :: (\nu Calc_1 \ldots Calc_n)\big((\nu av)(\prod_{i=1}^{n} \mathrm{rec}\ X.av!Calc_i.X \mid \\ *CalculateRating \Rightarrow av?(u).\mathsf{invoke}\ u) \mid \prod_{i=1}^{n} *Calc_i \Rightarrow P\big)$$

Instead of directly computing the rating, upon invocation the service receives (through an intra-location communication on the private channel av) the name of the "private" local service $Calc_i$ that actually computes the rating. The proof that this implementation is weak bisimilar to system (14) is a simple application of the behavioural theory developed so far. Note that, removing e.g., the restriction on av breaks the bisimilarity, since the implementation of $CalculateRating$ could then interact with another channel av in the environment, while the specification does not allow this interaction.

This implementation exploits multiparty sessions. In fact, the invoker and services $CalculateRating$ and $Calc_i$ are three endpoints of the same session. Note that $Calc_i$ has been added dynamically by $CalculateRating$, however it can interact directly also with the other endpoint. Programming the same behaviour in SSCC would require two binary sessions and some auxiliary communications.

Another way to create a ternary session is by using the merge primitive, as shown below. For simplicity, we consider just one such session:

$$(\nu e)l :: CalculateRating \Rightarrow \text{rec } Y.(\mathsf{merge}^+\ e.\mathsf{install}[CalculateRating \Rightarrow Y]) \mid \text{rec } X.(\nu r)r \rhd \mathsf{merge}^-\ e.(P|X).$$

In this case, the invocation in the specification is simulated by the invocation in the implementation plus the merge. Note that e should be bound to avoid interference, and that the merge has to be completed before $CalculateRating$ can be made available again. Similarly, r is restricted to avoid different recursive calls to interfere. We prove now the correctness of the transformation, which exploits the axioms presented before.

Proof (Correctness of the transformation). Upon invocation of $CalculateRating$, system 14 becomes:

$$l :: *CalculateRating \Rightarrow P \mid l :: r' \rhd P \tag{15}$$

Its implementation can execute the same transition, and reduce via a sequence of internal actions (merge and install) to:

$$(\nu e)l :: CalculateRating \Rightarrow \text{rec } Y.(\mathsf{merge}^+\ e.\mathsf{install}[CalculateRating \Rightarrow Y]) \mid l :: (\nu r'')r' \rhd \mathbf{0} \mid r'' \rhd (P|\text{rec } X.(\nu r)r \rhd \mathsf{merge}^-\ e.(P|X)) \mid r' \doteq r''$$

We can use Equation 9 to remove $r' \rhd \mathbf{0}$ and structural congruence to apply and remove the explicit substitution, obtaining:

$$(\nu e)l :: CalculateRating \Rightarrow \text{rec } Y.(\mathsf{merge}^+\ e.\mathsf{install}[CalculateRating \Rightarrow Y]) \mid l :: r' \rhd (P|\text{rec } X.(\nu r)r \rhd \mathsf{merge}^-\ e.(P|X))$$

By unfolding recursion we get:

$$(\nu e)l :: CalculateRating \Rightarrow \text{rec } Y.(\mathsf{merge}^+\ e.\mathsf{install}[CalculateRating \Rightarrow Y]) \mid l :: r' \rhd (P|(\nu r)r \rhd \mathsf{merge}^-\ e.(P|\text{rec } X.(\nu r)r \rhd \mathsf{merge}^-\ e.(P|X)))$$

and using structural congruence and Equation 10 we get:

$$(\nu e)l :: \mathit{CalculateRating} \Rightarrow \mathrm{rec}\ Y.(\mathsf{merge}^{+}\ e.\mathsf{install}[\mathit{CalculateRating} \Rightarrow Y]) \mid \\ l :: r' \rhd P \mid (\nu r)r \rhd \mathsf{merge}^{-}\ e.(P|\mathrm{rec}\ X.(\nu r)r \rhd \mathsf{merge}^{-}\ e.(P|X))$$

Using again structural congruence, and in particular folding again the recursion we go back to the original system, with an additional $r' \rhd P$ parallel component. This is exactly what happens for the specification, thus one can use closure under parallel composition to prove by coinduction the correctness of the transformation.

4 Dynamic Conversations

In this section we define the behavioural semantics of the Conversation Calculus (CC) [VCS08] (see also Chapter 2-1) and report results that: (1) corroborate our syntactically chosen constructs at the semantic level; and (2) provide further insight on the communication model of the CC. The operational semantics of the core CC is defined by a labelled transition system, which definition relies on the following notions of transition labels and actions.

Definition 4. *Transition labels and actions are defined as follows:*

$$\begin{array}{ll} \sigma ::= \tau \mid l^d!(a) \mid l^d?(a) \mid \texttt{this} & \text{(Actions)} \\ \lambda ::= c\ \sigma \mid \sigma \mid (\nu a)\lambda & \text{(Transition Labels)} \end{array}$$

An action τ denotes an internal communication, actions $l^d!(a)$ and $l^d?(a)$ represent communications with the environment, and `this` represents a conversation identity access. To capture the observational semantics of processes, transition labels need to register not only the action but also the conversation where the action takes place. So, a transition label λ containing $c\ \sigma$ is said to be *located at* conversation c (or just *located*), otherwise is said to be *unlocated*. In $(\nu a)\lambda$ the distinguished occurrence of a is bound with scope λ (cf., the π-calculus bound output actions). For a communication label λ we denote by $\overline{\lambda}$ the dual matching label obtaining by swapping inputs with outputs, such that, e.g., $\overline{l^d!(a)} = l^d?(a)$ and $\overline{l^d?(a)} = l^d!(a)$. The `this` transition label represents a conversation identity access. Processes can explicitly access the identity of the conversation in which they are located (which is captured by a `this` label), and synchronisations between processes may also require such contextual information.

We may now define the labelled transition system. For the sake of presentation, we split the presentation into two sets of rules, one (in Fig. 8) containing the rules for the basic operators, which are essentially identical to the corresponding ones in the π-calculus, and the other (in Fig. 9) grouping the rules specific to the Conversation Calculus.

Definition 5 (CC Labeled Transition System). *The rules in Fig. 8 and in Fig. 9 inductively define the LTS on processes.*

$$l^d!(a).P \xrightarrow{l^d!(a)} P \ (Out) \qquad l^d?(x).P \xrightarrow{l^d?(a)} P\{a/x\} \ (In) \qquad \frac{\alpha_j.P_j \xrightarrow{\lambda} Q \quad j \in I}{\Sigma_{i\in I}\ \alpha_i.P_i \xrightarrow{\lambda} Q}(Sum)$$

$$\frac{P \xrightarrow{\lambda} Q \quad a \in out(\lambda)}{(\boldsymbol{\nu} a)P \xrightarrow{(\nu a)\lambda} Q}(Open) \qquad \frac{P \xrightarrow{\lambda} Q \quad a \notin na(\lambda)}{(\boldsymbol{\nu} a)P \xrightarrow{\lambda} (\boldsymbol{\nu} a)Q}(Res)$$

$$\frac{P \xrightarrow{\lambda} Q \quad bn(\lambda) \# fn(R)}{P \mid R \xrightarrow{\lambda} Q \mid R}(Par\text{-}l) \qquad \frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\overline{\lambda}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}(Comm)$$

$$\frac{P \xrightarrow{(\nu a)\overline{\lambda}} P' \quad Q \xrightarrow{\lambda} Q' \quad a \notin fn(Q)}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu} a)(P' \mid Q')}(Close\text{-}l) \qquad \frac{P\{\mathsf{rec}\,\mathcal{X}.P/\mathcal{X}\} \xrightarrow{\lambda} Q}{\mathsf{rec}\,\mathcal{X}.P \xrightarrow{\lambda} Q}(Rec)$$

Fig. 8. CC LTS - Basic Operators (π-calculus like).

Transition rules presented in Fig. 8 should be fairly clear to a reader familiar with mobile process calculi. We discuss the intuitions behind the rules shown in Fig. 9. In rule (*Here*) an $\uparrow$ directed message (to the enclosing conversation) becomes $\downarrow$ (in the current conversation), after passing through the conversation access boundary. In rule (*Loc*) an unlocated $\downarrow$ message (in the current conversation) gets explicitly located at the conversation c in which it originates. In rule (*Through*) an already located communication label transparently crosses some other conversation boundary, and likewise for a τ label in rule (*Tau*). In rule (*This*) a `this` label reads the current conversation identity, and originates a c `this` label. A c `this` labelled transition may only progress inside the c conversation, as expressed by the rule (*ThisLoc*), where a `this` label matches the enclosing conversation. In rules (*ThisComm-r*) and (*ThisClose-r*) an unlocated communication matches a communication located at c, originating a c `this` label, thus ensuring the interaction occurs in the given conversation c.

Building on the notion of observation over processes captured by the labelled transition system of the CC, we characterise the CC semantic object by an observational equivalence, expressed in terms of standard notions of strong and weak bisimilarity defined as for SSCC (cf., Definition 2). We prove the expected properties of strong and weak bisimilarity: they are equivalence relations and they are preserved under a standard set of structural laws (cf., π-calculus structural congruence [SW01b]). Then we prove that strong bisimilarity and weak bisimilarity are congruences.

Theorem 2. *Strong bisimilarity and weak bisimilarity are congruences.*

Next, we show other interesting behavioural equations, that confirm basic intuitions about our conversation-based communication model.

Given processes P and Q, the following axioms hold:

$$\frac{P \xrightarrow{\lambda^{\uparrow}} Q}{c \blacktriangleleft [P] \xrightarrow{\lambda^{\downarrow}} c \blacktriangleleft [Q]}(Here) \qquad \frac{P \xrightarrow{\lambda^{\downarrow}} Q}{c \blacktriangleleft [P] \xrightarrow{c.\lambda^{\downarrow}} c \blacktriangleleft [Q]}(Loc)$$

$$\frac{P \xrightarrow{a\ \lambda^{\downarrow}} Q}{c \blacktriangleleft [P] \xrightarrow{a\ \lambda^{\downarrow}} c \blacktriangleleft [Q]}(Through) \qquad \frac{P \xrightarrow{\tau} Q}{c \blacktriangleleft [P] \xrightarrow{\tau} c \blacktriangleleft [Q]}(Tau)$$

$$\mathsf{this}(x).P \xrightarrow{c\ \mathtt{this}} P\{c/x\}\ (This) \qquad \frac{P \xrightarrow{c\ \mathtt{this}} Q}{c \blacktriangleleft [P] \xrightarrow{\tau} c \blacktriangleleft [Q]}(ThisLoc)$$

$$\frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{c\ \overline{\sigma}} Q'}{P \mid Q \xrightarrow{c\ \mathtt{this}} P' \mid Q'}(ThisComm\text{-}r) \qquad \frac{P \xrightarrow{\sigma} P' \quad Q \xrightarrow{(\nu a)c\ \overline{\sigma}} Q'}{P \mid Q \xrightarrow{c\ \mathtt{this}} (\boldsymbol{\nu} a)(P' \mid Q')}(ThisClose\text{-}r)$$

Fig. 9. CC LTS - Conversation Operators.

Conversation Split

$$n \blacktriangleleft [P] \mid n \blacktriangleleft [Q] \sim n \blacktriangleleft [P \mid Q] \tag{16}$$

Conversation Nesting

$$m \blacktriangleleft [n \blacktriangleleft [o \blacktriangleleft [P]]] \sim n \blacktriangleleft [o \blacktriangleleft [P]] \tag{17}$$

Output Nested Up — Output Here

$$n \blacktriangleleft \left[l^{\uparrow}!(\tilde{n}).P\right] \sim l^{\downarrow}!(\tilde{n}).n \blacktriangleleft [P] \tag{18}$$

Input Nested Up — Input Here

$$n \blacktriangleleft \left[l^{\uparrow}?(\tilde{x}).P\right] \sim l^{\downarrow}?(\tilde{x}).n \blacktriangleleft [P] \qquad (n \notin \tilde{x}) \tag{19}$$

Output Nested Here

$$m \blacktriangleleft \left[n \blacktriangleleft \left[l^{\downarrow}!(\tilde{n}).P\right]\right] \sim n \blacktriangleleft \left[l^{\downarrow}!(\tilde{n}).m \blacktriangleleft [n \blacktriangleleft [P]]\right] \tag{20}$$

Input Nested Here

$$m \blacktriangleleft \left[n \blacktriangleleft \left[l^{\downarrow}?(\tilde{x}).P\right]\right] \sim n \blacktriangleleft \left[l^{\downarrow}?(\tilde{x}).m \blacktriangleleft [n \blacktriangleleft [P]]\right] \qquad (\{m,n\} \# \tilde{x}) \tag{21}$$

Equation 16 captures the notion of conversation context as a single medium accessible through distinct pieces. Equation 17 expresses the fact that processes may only interact in the conversation in which they are located and in the enclosing one (via $\uparrow$ communications). Notice however that there are processes P and Q such that:

$$n \blacktriangleleft [m \blacktriangleleft [P] \mid Q] \not\sim m \blacktriangleleft [P] \mid n \blacktriangleleft [Q] \tag{22}$$

For instance, consider processes R_1 and R_2 defined as follows:

$$R_1 \triangleq c \blacktriangleleft \left[b \blacktriangleleft \left[l^{\uparrow}!(a) \right] \mid l^{\downarrow}?(x) \right] \qquad R_2 \triangleq b \blacktriangleleft \left[l^{\uparrow}!(a) \right] \mid c \blacktriangleleft \left[l^{\downarrow}?(x) \right]$$

Since R_1 exhibits a τ transition and R_2 does not, we have that $R_1 \not\sim R_2$. The inequation (22) contrasts with Equation 16: the relation between a conversation and its caller must be preserved. Equations 18-19 illustrate the notion of enclosing conversation: a $\uparrow$ message prefix located in a nested conversation behaves the same as a $\downarrow$ message prefix in the current conversation. Equations 20-21 show that a here ($\downarrow$) message prefix together with the respective conversation can be pulled up to top level in the conversation nesting.

The behavioural laws shown above hint on the abstract spatial model of CC processes, and pave the way for establishing a normal form result: we prove that any CC process is behaviourally equivalent to a process in normal form—considering the depth of a process is the number of enclosing conversation access pieces, we say a process is in normal form if all its active communication prefixes are of (at most) depth two (see [VCS08]).

To conclude this section, we show an example derivation that uses Theorem 2 and the equational laws above. We consider a system where a *Client* and a *FinancePortal* exchange a `login` message in conversation *CreditChat*, each party holding a distinct piece of the conversation. In Fig. 10 we show that such system behaviourally coincides with a system where such message exchange takes place in a single conversation piece. Such behavioural reconfigurations suggest

$$\begin{array}{l}
\quad \mathit{FinancePortal} \blacktriangleleft [\ \mathit{CreditChat} \blacktriangleleft [\ \mathtt{login}^{\downarrow}?(\mathit{uId}).\mathit{ServiceProtocol}\ ]\ ] \\
\quad \mid \\
\quad \mathit{Client} \blacktriangleleft [\ \mathit{CreditChat} \blacktriangleleft [\ \mathtt{login}^{\downarrow}!(\mathit{uId}).\mathit{ClientProtocol}\ ]\ ] \\
\qquad\qquad\qquad\qquad \sim \quad \text{(Theorem 2 and Equation 21)} \\
\mathit{CreditChat} \blacktriangleleft [\ \mathtt{login}^{\downarrow}?(\mathit{uId}).\mathit{FinancePortal} \blacktriangleleft [\ \mathit{CreditChat} \blacktriangleleft [\ \mathit{ServiceProtocol}\ ]\ ]\ ] \\
\mid \\
\mathit{Client} \blacktriangleleft [\ \mathit{CreditChat} \blacktriangleleft [\ \mathtt{login}^{\downarrow}!(\mathit{uId}).\mathit{ClientProtocol}\ ]\ ] \\
\qquad\qquad\qquad\qquad \sim \quad \text{(Theorem 2 and Equation 20)} \\
\mathit{CreditChat} \blacktriangleleft [\ \mathtt{login}^{\downarrow}?(\mathit{uId}).\mathit{FinancePortal} \blacktriangleleft [\ \mathit{CreditChat} \blacktriangleleft [\ \mathit{ServiceProtocol}\ ]\ ]\ ] \\
\mid \\
\mathit{CreditChat} \blacktriangleleft [\ \mathtt{login}^{\downarrow}!(\mathit{uId}).\mathit{Client} \blacktriangleleft [\ \mathit{CreditChat} \blacktriangleleft [\ \mathit{ClientProtocol}\ ]\ ]\ ] \\
\qquad\qquad\qquad\qquad \sim \quad \text{(Equation 16)} \\
\mathit{CreditChat} \blacktriangleleft [\ \mathtt{login}^{\downarrow}?(\mathit{uId}).\mathit{FinancePortal} \blacktriangleleft [\ \mathit{CreditChat} \blacktriangleleft [\ \mathit{ServiceProtocol}\ ]\ ] \\
\qquad\qquad \mid \\
\qquad\qquad \mathtt{login}^{\downarrow}!(\mathit{uId}).\mathit{Client} \blacktriangleleft [\ \mathit{CreditChat} \blacktriangleleft [\ \mathit{ClientProtocol}\ ]\ ]\ ]
\end{array}$$

Fig. 10. Credit Request System Behavioural Reconfiguration.

$$
\begin{array}{lll}
g + \mathbf{0} \equiv g & g_1 + g_2 \equiv g_2 + g_1 & (g_1 + g_2) + g_3 \equiv g_1 + (g_2 + g_3) \\
s \mid \mathbf{0} \equiv s & s_1 \mid s_2 \equiv s_2 \mid s_1 & (s_1 \mid s_2) \mid s_3 \equiv s_1 \mid (s_2 \mid s_3) \\
* \mathbf{0} \equiv \mathbf{0} & * s \equiv s \mid * s & \\
\{\!|\mathbf{0}|\!\} \equiv \mathbf{0} & \{\!|\, \{\!|s|\!\} \,|\!\} \equiv \{\!|s|\!\} & \{\!|[e]\, s|\!\} \equiv [e]\, \{\!|s|\!\} \\
[e]\, \mathbf{0} \equiv \mathbf{0} & [e_1]\, [e_2]\, s \equiv [e_2]\, [e_1]\, s & s_1 \mid [e]\, s_2 \equiv [e]\, (s_1 \mid s_2) \;\; \text{if } e \notin \mathrm{fe}(s_1) \cup \mathrm{fk}(s_2)
\end{array}
$$

Fig. 11. COWS structural congruence

an alternative characterisation of the operational semantics of the CC based on a notion of reduction: we may describe the evolution of the basic representatives of the behavioural equivalence classes, and then close the reduction relation under such equivalence classes.

5 Behavioural Semantics for COWS

In this section, we present the operational semantics of COWS (Calculus for Orchestration of Web-Services [LPT07]), together with some bisimulation-based observational semantics. The syntax of COWS is presented in Chapter 2-1.

Operational semantics. The operational semantics of COWS is defined using an LTS in the 'late' style only for *closed* services, i.e. services without free variables and killer labels (of course, closed services may contain free names). To simplify the rules, we exploit a relation of structural congruence, written $\equiv$. It is defined as the least congruence relation induced by the equational laws shown in Figure 11.

To define the labelled transition relation, we use a few auxiliary functions. As a matter of notation, we shall use $\mathtt{n}$ to range over endpoints that do not contain variables (e.g. $p \bullet o$), and $\mathtt{u}$ to range over endpoints that may contain variables (e.g. $u \bullet u'$). Firstly, we use the function $[\![\_]\!]$ for evaluating *closed* expressions (i.e. expressions without variables): it takes a closed expression and returns a value. It is not explicitly defined since the exact syntax of expressions is deliberately not specified. Secondly, we use the partial function $\mathcal{M}(\_,\_)$ for performing *pattern-matching* on semi-structured data and, thus, determining if a receive and an invoke over the same endpoint can synchronise. Two tuples match if they have the same number of fields and corresponding fields have matching values/variables. Variables match any value, and two values match only if they are identical. When tuples $\bar{w}$ and $\bar{v}$ do match, $\mathcal{M}(\bar{w}, \bar{v})$ returns a substitution for the variables in $\bar{w}$; otherwise, it is undefined. Here substitutions (ranged over by σ) are written as collections of pairs of the form $x \mapsto v$. Application of substitution σ to s is written $s \cdot \sigma$. This may require a preventive α-conversion. In fact, we identify services up to the services' equivalence classes induced by α-conversion, also when this is not explicitly mentioned. We use $\emptyset$ to denote the empty substitution, $|\sigma|$ to denote the number of pairs in σ, and $\sigma_1 \uplus \sigma_2$ to denote the union of σ_1 and σ_2 when they have disjoint domains. Thirdly, we define a function, named $halt(\_)$, that takes a service s as an argument and returns the service obtained by only retaining the protected activities inside s. Function $halt(\_)$ is defined inductively on the syntax of services. The most significant case is $halt(\{\!|s|\!\}) = \{\!|s|\!\}$. In the

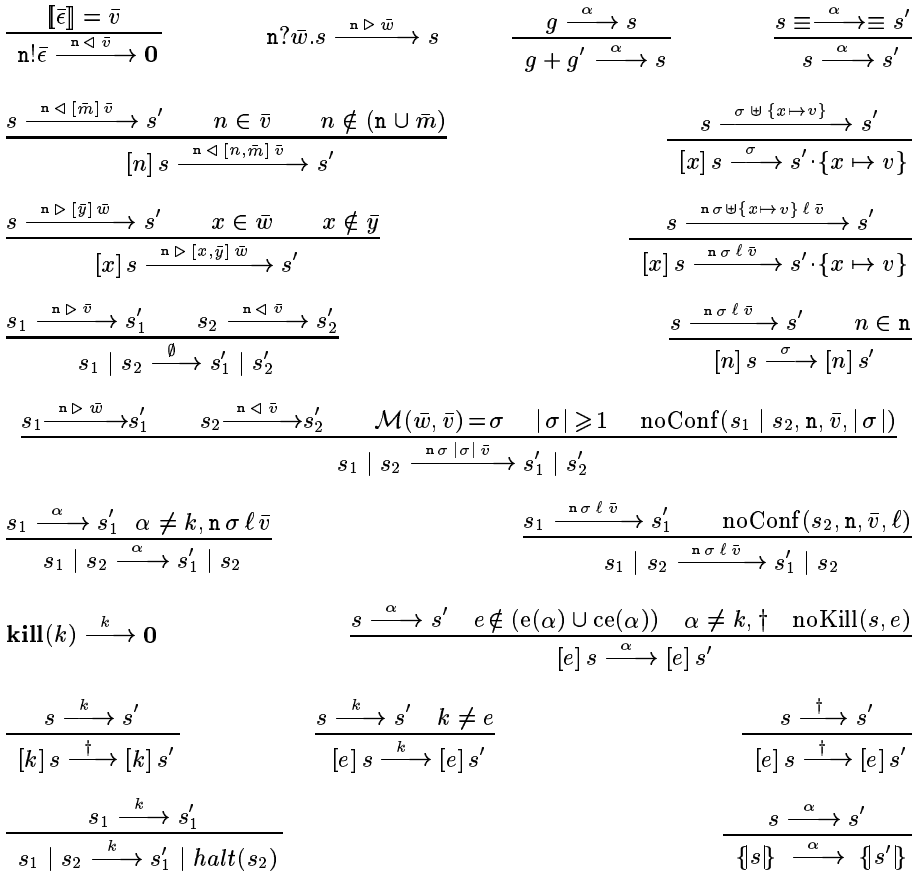

Fig. 12. COWS operational semantics

other cases, $halt(\_)$ returns $\mathbf{0}$, except for parallel composition, delimitation and replication operators, for which it acts as an homomorphism. Finally, we use two predicates: $\mathrm{noKill}(s, e)$ holds true if either e is not a killer label or $e = k$ and s cannot immediately perform a free activity $\mathbf{kill}(k)$; $\mathrm{noConf}(s, \mathtt{n}, \bar{v}, \ell)$, with ℓ natural number, holds true if s does not produce communication *conflicts*, i.e. s cannot immediately perform a receive activity matching $\bar{v}$ over the endpoint $\mathtt{n}$ that generates a substitution with fewer pairs than ℓ. Their inductive definitions can be found in [LPT07].

Definition 6 (COWS Labelled Transition System). *The rules in Figure 12 inductively define the LTS on processes. Labels α are generated by the following grammar:*

$$\alpha \quad ::= \quad \mathtt{n} \lhd [\bar{n}]\,\bar{v} \quad | \quad \mathtt{n} \rhd [\bar{x}]\,\bar{w} \quad | \quad \sigma \quad | \quad \mathtt{n}\,\sigma\,\ell\,\bar{v} \quad | \quad k \quad | \quad \dagger$$

The meaning of labels is as follows: $\mathtt{n} \lhd [\bar{n}]\,\bar{v}$ and $\mathtt{n} \rhd [\bar{x}]\,\bar{w}$ denote execution of invoke and receive activities over the endpoint $\mathtt{n}$ with arguments $\bar{v}$ and

$\bar{w}$, respectively, of which $\bar{n}$ and $\bar{x}$ are bound; σ denotes execution of a communication, not subject to priority check, with generated substitution σ to be still applied; $\mathtt{n}\,\sigma\,\ell\,\bar{v}$ denotes execution of a communication, subject to priority check, over $\mathtt{n}$ with matching values $\bar{v}$, generated substitution having ℓ pairs, and substitution σ to be still applied; k denotes execution of a request for terminating a term from within the delimitation $[k]$, and $\dagger$ denotes taking place of forced termination. In particular, the empty substitution $\emptyset$ and labels of the form $\mathtt{n}\,\emptyset\,\ell\,\bar{v}$ denote *computational steps* corresponding to taking place of communication without pending substitutions, while $\dagger$ denotes a computational step corresponding to taking place of forced termination. We will use $\mathrm{bu}(\alpha)$ to denote the set of names/variables that occur bound in α, $\mathrm{e}(\alpha)$ to denote the set of elements (i.e. names, variables and killer labels) occurring in α, except for $\alpha = \mathtt{n}\,\sigma\,\ell\,\bar{v}$ for which we let $\mathrm{e}(\mathtt{n}\,\sigma\,\ell\,\bar{v}) = \mathrm{e}(\sigma)$, and $\mathrm{ce}(\alpha)$ to denote the names composing the endpoint in case α denotes execution of a communication.

Observational semantics. We now define natural notions of barbed bisimilarities for COWS and prove their coincidence with more manageable characterisations in terms of (labelled) bisimilarities. We want to define a notion of *barbed bisimilarity* for the calculus along the line of [HY95,SW01a]. To this aim, since communication is asynchronous, we consider as *basic observable* only the output capabilities of terms, like for asynchronous π-calculus [ACS98]. The intuition is that an asynchronous observer cannot directly observe the receipt of data that he has sent.

Definition 7 (Basic observable). *Let s be a* COWS *closed term. Predicate $s \downarrow_{\mathtt{n}}$ holds true if there exist s', $\bar{n}$ and $\bar{v}$ such that $s \xrightarrow{\mathtt{n} \lhd [\bar{n}]\,\bar{v}} s'$.*

Definition 8 (Barbed bisimilarity for COWS). *A symmetric binary relation $\mathcal{R}$ on* COWS *closed terms is a* barbed bisimulation *if it is barb preserving, and computation and context closed. Two closed terms s_1 and s_2 are* barbed bisimilar*, written $s_1 \simeq s_2$, if $s_1 \mathcal{R} s_2$ for some barbed bisimulation $\mathcal{R}$. $\simeq$ is called barbed bisimilarity.*

Context closure condition enables compositional reasoning, since it implies that $\simeq$ is a congruence on COWS closed terms, but requires considering all possible language contexts. To avoid this universal quantification, we provide a purely co-inductive notion of bisimulation that only requires considering transitions of the LTS defining the semantics of the terms under analysis. Because in COWS only the output capability of names can be exported, we define a COWS bisimulation as a family of relations indexed with sets of names corresponding to the names that cannot be used by contexts (to test) for reception since they are dynamically exported private names.

Definition 9 (Names-indexed family of relations). *A names-indexed family $\mathcal{F}$ of relations is a set of symmetric binary relations $\mathcal{R}_{\mathcal{N}}$ on* COWS *closed terms, one for each set of names $\mathcal{N}$, i.e. $\mathcal{F} = \{\mathcal{R}_{\mathcal{N}}\}_{\mathcal{N}}$.*

To be a congruence, bisimilarity must explicitly take care of the terms resulting from application of function $halt(\_)$, that gets the same effect as of plunging its argument term within the context $[k]\,(\mathbf{kill}(k) \mid [\![\cdot]\!])$.

Definition 10 (COWS bisimilarity). *A names-indexed family of relations $\{\mathcal{R}_\mathcal{N}\}_\mathcal{N}$ is a* COWS bisimulation *if, whenever $s_1 \mathcal{R}_\mathcal{N} s_2$ then the following two conditions hold:*

a. $halt(s_1)\,\mathcal{R}_\mathcal{N}\,halt(s_2)$ *and*

b. *if* $s_1 \xrightarrow{\alpha} s_1'$, *where* $\mathrm{bu}(\alpha)$ *are fresh, then:*

1. *if* $\alpha = \mathtt{n} \rhd [\bar{x}]\,\bar{w}$ *then one of the following holds:*
   (a) $\exists\, s_2' : s_2 \xrightarrow{\mathtt{n} \rhd [\bar{x}]\,\bar{w}} s_2'$ *and* $\forall\, \bar{v}$ *s.t.* $\mathcal{M}(\bar{x}, \bar{v}) = \sigma$ *and* $\mathrm{noConf}(s_2, \mathtt{n}, \bar{w}\sigma, |\,\bar{x}\,|) : s_1'\sigma\,\mathcal{R}_\mathcal{N}\,s_2'\sigma$
   (b) $|\,\bar{x}\,| = |\,\bar{w}\,|$ *and* $\exists\, s_2' : s_2 \xrightarrow{\emptyset} s_2'$ *and* $\forall\, \bar{v}$ *s.t.* $\mathcal{M}(\bar{x}, \bar{v}) = \sigma$ *and* $\mathrm{noConf}(s_2, \mathtt{n}, \bar{w}\cdot\sigma, |\,\bar{x}\,|) : s_1'\cdot\sigma\,\mathcal{R}_\mathcal{N}\,(s_2' \mid \mathtt{n}!\bar{v})$ *or* $s_1'\cdot\sigma\,\mathcal{R}_\mathcal{N}\,(s_2' \mid \{\!|\mathtt{n}!\bar{v}|\!\})$
2. *if* $\alpha = \mathtt{n}\,\emptyset\,\ell\,\bar{v}$ *and* $\ell = |\,\bar{v}\,|$ *then one of the following holds:*
   (a) $\exists\, s_2' : s_2 \xrightarrow{\mathtt{n}\,\emptyset\,\ell\,\bar{v}} s_2'$ *and* $s_1'\,\mathcal{R}_\mathcal{N}\,s_2'$ (b) $\exists\, s_2' : s_2 \xrightarrow{\emptyset} s_2'$ *and* $s_1'\,\mathcal{R}_\mathcal{N}\,s_2'$
3. *if* $\alpha = \mathtt{n} \lhd [\bar{n}]\,\bar{v}$ *where* $\mathtt{n} \notin \mathcal{N}$ *then* $\exists\, s_2' : s_2 \xrightarrow{\mathtt{n} \lhd [\bar{n}]\,\bar{v}} s_2'$ *and* $s_1'\,\mathcal{R}_{\mathcal{N} \cup \bar{n}}\,s_2'$
4. *if* $\alpha = \emptyset$, $\alpha = \dagger$ *or* $\alpha = \mathtt{n}\,\emptyset\,\ell\,\bar{v}$ *with* $\ell \neq |\,\bar{v}\,|$, *then* $\exists\, s_2' : s_2 \xrightarrow{\alpha} s_2'$ *and* $s_1'\,\mathcal{R}_\mathcal{N}\,s_2'$

Two closed terms s_1 and s_2 are $\mathcal{N}$-bisimilar, *written $s_1 \sim^\mathcal{N} s_2$, if $s_1 \mathcal{R}_\mathcal{N} s_2$ for some $\mathcal{R}_\mathcal{N}$ in a* COWS *bisimulation. They are* COWS bisimilar, *written $s_1 \sim s_2$, if they are $\emptyset$-bisimilar. $\sim^\mathcal{N}$ is called $\mathcal{N}$-bisimilarity, while $\sim$ is called* COWS *bisimilarity.*

This definition is more complex than Definition 2, since it has to account for priority of COWS communication, kill, and asynchrony. Our main results prove that COWS bisimilarity $\sim$ is a congruence for COWS and is *sound* and *complete* with respect to barbed bisimilarity.

Theorem 3 (Congruence). *$\sim$ is a congruence for* COWS *closed terms.*

Theorem 4 (Soundness and completeness of $\sim$ w.r.t. $\simeq$). *Given two* COWS *closed terms s_1 and s_2, $s_1 \sim s_2$ if and only if $s_1 \simeq s_2$.*

Our semantic theories extend in a standard way to the weak case so that results of congruence and coincidence still hold. We refer the interested reader to the extended version of [PTY09] for the exact definitions and a full account of the proofs.

Examples. We show now that, differently from asynchronous π-calculus, in COWS it is not true that receive activities are always unobservable. To illustrate this, we consider a tailored version of the input absorption law characterising asynchronous bisimulation in asynchronous π-calculus (*i.e.*, the equation $a(b).\,\bar{a}b + \tau = \tau$ presented in [ACS98]):

$$[x]\,(\,\emptyset + \mathtt{n}?\langle x, v\rangle.\,\mathtt{n}!\langle x, v\rangle\,)\;=\;\emptyset \tag{23}$$

where, for the sake of presentation, we exploit the context $\emptyset + [\![\cdot]\!] \triangleq [\mathtt{m}]\,(\mathtt{m}!\langle\rangle \mid \mathtt{m}?\langle\rangle + [\![\cdot]\!])$ and the term $\emptyset \triangleq [\mathtt{m}]\,(\mathtt{m}!\langle\rangle \mid \mathtt{m}?\langle\rangle)$. Communication along the private endpoint $\mathtt{m}$ models the τ action of π-calculus, while activities $\mathtt{n}?\langle x, v\rangle$ and $\mathtt{n}!\langle x, v\rangle$ recall the π-calculus actions $a(b)$ and $\bar{a}b$, respectively. Intuitively, the equality means that a service that emits the data it has received behaves as a service that simply performs an unobservable action, which means that receive activities cannot be observed. In COWS, however, the context $\mathbb{C} \triangleq [y, z]\,\mathtt{n}?\langle y, z\rangle.\,\mathtt{m}!\langle\rangle \mid \mathtt{n}!\langle v', v\rangle \mid [\![\cdot]\!]$ can tell the two terms above apart. In fact, we have

$$\mathbb{C}[\![\emptyset]\!] \xrightarrow{\mathtt{n}\,\emptyset\,2\,\langle v', v\rangle} \mathtt{m}!\langle\rangle \mid \emptyset$$

where the term $(\mathtt{m}!\langle\rangle \mid \emptyset)$ satisfies the predicate $\downarrow_{\mathtt{m}}$. Instead, the other term cannot properly reply because the receive $\mathtt{n}?\langle x, v\rangle$ has higher priority than $\mathtt{n}?\langle y, z\rangle$ when synchronising with the invocation $\mathtt{n}!\langle v', v\rangle$. Thus, $\mathbb{C}[\![[x]\,(\,\emptyset + \mathtt{n}?\langle x, v\rangle.\,\mathtt{n}!\langle x, v\rangle\,)]\!]$ can only evolve to terms that cannot immediately satisfy the predicate $\downarrow_{\mathtt{m}}$. From this, we have

$$[x]\,(\,\emptyset + \mathtt{n}?\langle x, v\rangle.\,\mathtt{n}!\langle x, v\rangle\,) \;\not\simeq\; \emptyset$$

Indeed, in COWS receive activities that exercise a priority (i.e. receives whose arguments contain some values) can be detected by an interacting observer.

Now, consider the term $[x, x']\,(\,\emptyset + \mathtt{n}?\langle x, x'\rangle.\,\mathtt{n}!\langle x, x'\rangle\,)$. Since $\mathtt{n}?\langle x, x'\rangle$ does not exercise any priority on parallel terms, we have that

$$[x, x']\,(\,\emptyset + \mathtt{n}?\langle x, x'\rangle.\,\mathtt{n}!\langle x, x'\rangle\,) \simeq \emptyset \qquad \mathbb{C}[\![[x, x']\,(\,\emptyset + \mathtt{n}?\langle x, x'\rangle.\,\mathtt{n}!\langle x, x'\rangle\,)]\!] \simeq \mathbb{C}[\![\emptyset]\!]$$

Similarly, taken $\mathbb{D} \triangleq \mathtt{n}?\langle\rangle.\,\mathtt{m}!\langle\rangle \mid \mathtt{n}!\langle\rangle \mid [\![\cdot]\!]$, we have that

$$\emptyset + \mathtt{n}?\langle\rangle.\,\mathtt{n}!\langle\rangle \simeq \emptyset \qquad \mathbb{D}[\![\emptyset + \mathtt{n}?\langle\rangle.\,\mathtt{n}!\langle\rangle]\!] \simeq \mathbb{D}[\![\emptyset]\!]$$

Therefore, communication in COWS is neither purely asynchronous nor purely synchronous. Indeed, receives having the smallest priority (i.e. whose arguments are, possible empty, tuples of variables) cannot be observed, while, by exploiting proper contexts, the other receives can be detected.

6 Conclusions

In this chapter we have studied behavioural equivalences in the context of different SENSORIA calculi.

For the session-based calculi, SSCC, μse and CC, we have presented a few axioms, and then applied them to prove the correctness of different kinds of

program transformations. Observing the behavioural identities characterised for the three calculi one may find similarities. For instance, we may notice that, in some sense, feeding streams, intra-site communication and communication to the caller context are disconnected from the particular subsidiary session or conversation—this is reflected in (5), (13) and (18). On the other hand, the fact that interaction under a session or conversation is independent from the location or caller context is reflected by (12) and (20). However, the presented models also present quite clear distinctions in the behavioural identities shown. For example, (3) and (10) contrast with (22): in CC the relation between caller context and subsidiary conversation must, in general, be preserved, since processes interacting in a subsidiary conversation may also continuously interact in their caller contexts. However, distinct sessions may be nested in μse and SSCC without any consequence to the behaviour of processes, as they either interact in the session or interact under the location (μse)/feed streams (SSCC), that can be accessed regardless of session nesting. Another distinction may be noticed at the level of the split rule for CC systems shown in Equation 16, which may not be reproduced in the either SSCC or μse.

For the correlation-based calculus COWS instead, the priority mechanism required by correlation has a strong impact on the classical behavioural theory. In particular, we have shown that a particular notion of labelled bisimilarity is needed to capture barbed bisimilarity, and that communication is neither purely synchronous nor purely asynchronous.

Acknowledgements

The work reported herein is the result of a collaborative effort of many researchers, not just of the authors. Special thanks to Rosario Pugliese and Francesco Tiezzi, who wrote the section on COWS.

Antnio Ravara was partially supported by the Security and Quantum Information Group, Instituto de Telecomunicações, Portugal.

References

[ACS98] R.M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous pi-calculus. *Theoretical Computer Science*, 195(2):291–324, 1998.

[Amb04] S.W. Ambler. *The Object Primer: Agile Model-Driven Development with UML 2.0.* Cambridge University Press, 2004.

[B+06] M. Boreale et al. SCC: a Service Centered Calculus. In *Proc. of WS-FM'06*, volume 4184 of *LNCS*, pages 38–57. Springer, 2006.

[BBDL08] M. Boreale, R. Bruni, R. De Nicola, and M. Loreti. Sessions and pipelines for structured service programming. In *Proc. of FMOODS'08*, volume 5051 of *LNCS*, pages 19–38. Springer, 2008.

[BC08] E. Bonelli and A. Compagnoni. Multipoint session types for a distributed calculus. In *Proc. of TGC'07*, volume 4912 of *LNCS*, pages 240–256. Springer, 2008.

[BLMT08] R. Bruni, I. Lanese, H. Melgratti, and E. Tuosto. Multiparty sessions in SOC. In *Proc. of COORDINATION'08*, volume 5052 of *LNCS*, pages 67–82. Springer, 2008.

[CF+07] L. Cruz-Filipe et al. Bisimulations in SSCC. DI/FCUL TR 07–37, Department of Informatics, Faculty of Sciences, University of Lisbon, 2007.

[CF+08] L. Cruz-Filipe et al. Behavioural theory at work: Program transformations in a service-centred calculus. In *Proc. of FMOODS'08*, volume 5051 of *LNCS*, pages 59–77. Springer, 2008.

[CHY07] M. Carbone, K. Honda, and N Yoshida. Structured communication-centred programming for web services. In *Proc. of ESOP'07*, volume 4421 of *LNCS*, pages 2–17. Springer, 2007.

[DH84] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[Gla01] R.J. van Glabbeek. The linear time – branching time spectrum I; the semantics of concrete, sequential processes. In *Handbook of Process Algebra*, chapter 1, pages 3–99. Elsevier, 2001. Available at `http://boole.stanford.edu/pub/spectrum1.ps.gz`.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[HY95] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 151(2):437–486, 1995.

[HYC08] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *Proc. of POPL'08*, pages 273–284. ACM Press, 2008.

[LMVR07] I. Lanese, F. Martins, V.T. Vasconcelos, and A. Ravara. Disciplining orchestration and conversation in service-oriented computing. In *Proc. of SEFM'07*, pages 305–314. IEEE Computer Society, 2007.

[LPT07] A. Lapadula, R. Pugliese, and F. Tiezzi. A calculus for orchestration of web services. In *Proc. of ESOP'07*, volume 4421 of *LNCS*, pages 33–47. Springer, 2007. Extended version available at `http://rap.dsi.unifi.it/cows/papers/cows-esop07-full.pdf`.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[MS92] R. Milner and D. Sangiorgi. Barbed bisimulation. In *Proc. of ICALP'92*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.

[PTY09] R. Pugliese, F. Tiezzi, and N. Yoshida. On observing dynamic prioritised actions in soc. In *Proc. of ICALP'09 (2)*, volume 5556 of *LNCS*, pages 558–570. Springer, 2009. Extended version available at `http://rap.dsi.unifi.it/cows/papers/bis4cows-full.pdf`.

[SW01a] D. Sangiorgi and D. Walker. On barbed equivalences in pi-calculus. In *Proc. of CONCUR'01*, volume 2154 of *LNCS*, pages 292–304. Springer, 2001.

[SW01b] D. Sangiorgi and D. Walker. *Pi-Calculus: A Theory of Mobile Processes.* Cambridge University Press, 2001.

[VCS08] H.T. Vieira, L. Caires, and J.C. Seco. The conversation calculus: A model of service-oriented computation. In *Proc. of ESOP'08*, volume 4960 of *LNCS*, pages 269–283. Springer, 2008.