Towards trustworthy multiparty sessions⋆ (extended abstract)

Roberto Bruni[1], Ivan Lanese[2], Hernán Melgratti[3], Leonardo Gaetano Mezzina[4], and Emilio Tuosto[5]

[1] Dipartimento di Informatica, Università di Pisa, Italy
[2] Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy
[3] Departamento de Computación, Universidad de Buenos Aires, Argentina
[4] IMT Lucca, Institute for Advanced Studies, Italy
[5] Department of Computer Science, University of Leicester, UK

1 Introduction

A lively research direction in concurrent and distributed computing is pushing for a synergic design of theoretical frameworks and programming mechanisms that help theoreticians and practitioners to cope with the complexity of large system development and analysis. Among others, type-based approaches look rather handy both for theoretical and applied research. Process calculi provide a paradigmatic example: they offer a convenient formal setting for modeling modern systems (e.g., Service Oriented Computing (SOC) [2, 8, 4]) and yield a natural framework for enforcing desired properties using behavioral types [5].

Recently, researchers have shown a growing interest around the notion of a multiparty session [6, 1], because typical modern distributed applications can involve the coordination of many endpoints. Remarkably, SOC adds a further difficulty as endpoints may be dynamically discovered and assembled. In fact, SOC aims to the seamless and trustworthy integration of separately developed computational entities, called *services*. To achieve this, matching criteria must be available for comparing a service description against the requirements of the invoker before binding the two.

$\mu\mathsf{se}$ (read "muse", after MUltiparty SEssions) [3], is a process calculus for expressing computations where endpoints dynamically join existing multiparty sessions. In this context, it is crucial to have type systems for the early detection of possible sources of incompatibility.

In this paper we consider $\mu\mathsf{se}$ without name passing and intra-site communication (§ 2) and sketch a type system aimed to guarantee a weak form of compatibility (§ 3), in the sense that all the interactions required by each local task can be provided either by endpoints currently participating to its session or by endpoints that can join that session later.

This is work in progress and we plan to refine the type system to have stronger guarantees. Due to space limitation we cannot give a comprehensive presentation of our ideas, which will be spelled out in the full version of this paper.

⋆ Research supported by the FET-GC2 EU project IST-2005-16004 Sensoria by the UK project HiDeA4SOC and by the Italian FIRB project TOCAI.

$$
\begin{array}{rlll}
S, T ::= & l :: a \Rightarrow P & \text{Service definition} \\
\mid & l :: P & \text{Located process} \\
\mid & S|T & \text{Parallel composition} \\
\mid & (\nu n)S & \text{New name} \\
\mid & r \doteq s & \text{explicit substitution}
\end{array}
\qquad
\begin{array}{rlll}
P, Q ::= & \mathbf{0} & \text{Empty process} \\
\mid & c.P & \text{Action prefix} \\
\mid & \mathsf{install}[a \Rightarrow P].Q & \text{Service installation} \\
\mid & \mathsf{invoke}\ a.P & \text{Service invocation} \\
\mid & \mathsf{merge}^p\ e.P & \text{Entry-point} \\
\mid & r \rhd P & \text{Endpoint} \\
\mid & P|Q & \text{Parallel composition} \\
\mid & (\nu n)P & \text{New name} \\
\mid & \mathsf{rec}\ X.P & \text{Recursive process} \\
\mid & X & \text{Recursive call}
\end{array}
$$

Fig. 1. Syntax of systems and processes

$$
c.P \xrightarrow{c} P \qquad \mathsf{merge}^p\ e.P \xrightarrow{e^p} P \qquad \mathsf{invoke}\ a.P \xrightarrow{\perp a} P \qquad a \Rightarrow P \xrightarrow{r \top a} r \rhd P
$$

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

$$
\mathsf{install}[a \Rightarrow R].P \xrightarrow{a[R]} P \qquad \frac{P \xrightarrow{a[R]} Q}{l :: P \xrightarrow{\tau} l :: Q \,|\, l :: a \Rightarrow R} \qquad \frac{P \xrightarrow{\alpha} Q \quad \alpha \notin \{a[R]\}}{l :: P \xrightarrow{\alpha} l :: Q}
$$

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

$$
\frac{\mathcal{A} \xrightarrow{\alpha} \mathcal{A}'}{\mathcal{A}|\mathcal{B} \xrightarrow{\alpha} \mathcal{A}'|\mathcal{B}} \qquad \frac{\mathcal{A} \xrightarrow{\alpha} \mathcal{A}' \quad n \notin \mathrm{n}(\alpha)}{(\nu n)\mathcal{A} \xrightarrow{\alpha} (\nu n)\mathcal{A}'} \qquad \frac{\mathcal{A} \equiv \mathcal{A}' \quad \mathcal{A}' \xrightarrow{\alpha} \mathcal{B}' \quad \mathcal{B}' \equiv \mathcal{B}}{\mathcal{A} \xrightarrow{\alpha} \mathcal{B}}
$$

Fig. 2. Operational semantics

2 μse Basics

The syntax of μse is in Fig. 1. We assume given sets of names for services (ranged by $a, ...$), sessions ($r, s, ...$), communication channels ($x, ...$), co-actions ($\overline{x}, ...$), entry-points ($e, ...$), and sites ($l, ...$). We let $c, ...$ range over $x, \overline{x}, ...$ and let $\overline{\overline{c}} = c$.

Processes $P, Q, ...$ comprise ordinary operators such as the nil process $\mathbf{0}$, parallel composition $P|Q$, communication prefixes $c.P$, recursion $\mathsf{rec}\ X.P$ and name restriction $(\nu n)P$, together with primitives $\mathsf{invoke}\ a.P$ for service invocation, $\mathsf{install}[a \Rightarrow P].Q$ for dynamic installation of new services, $r \rhd P$ for session endpoints and $\mathsf{merge}^p\ e.P$ for dynamic merge of sessions on suitable entry-points. Merge prefixes are polarized with $p \in \{+, -\}$. Trailing occurrences of $\mathbf{0}$ will be omitted. Systems $S, T, ...$ comprise located service definitions $l :: a \Rightarrow P$ or processes $l :: P$, parallel composition $S|T$, name restriction $(\nu n)S$, and explicit fusions of session names $r \doteq s$. We let $\mathcal{A}, \mathcal{B}, ...$ range over systems and processes.

The operational semantics of μse is in Fig. 2. It relies on the structural congruence in Fig. 3. Roughly, the following interactions are possible. If service definition $l :: a \Rightarrow P$ is available, an invocation to a within a session r creates a new endpoint $l :: r \rhd P$ for r on site l. The prefix $\mathsf{install}[a \Rightarrow R].P$ dynamically

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

$$(\nu n)(\nu m)\mathcal{A} \equiv (\nu m)(\nu n)\mathcal{A} \qquad \left.\begin{array}{c}(\nu n)\mathcal{A} \equiv \mathcal{A} \\ (\nu n)(\mathcal{A}|\mathcal{A}'') \equiv \mathcal{A}|(\nu n)\mathcal{A}''\end{array}\right\} \text{ if } n \notin \mathrm{fn}(\mathcal{A})$$

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

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

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

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

Fig. 3. Structural congruence rules

installs a new service definition $a \Rightarrow R$ at the top of the site where it runs. Dual action prefixes $c.P$ and $\overline{c}.Q$ can synchronize only if they run on endpoints of the same session. Complementary merges in sessions r and s on the same entry-point e (i.e., $\mathsf{merge}^+\ e$ and $\mathsf{merge}^-\ e$) can synchronize, releasing the fusion $r \doteq s$.

Example 1. Consider $T_1 = (\nu r_1, r_2, r_3)(l_1 :: P_1 \mid l_2 :: P_2 \mid l_3 :: P_3)$, where:

$$P_1 = r_1 \rhd \mathsf{merge}^+\ e_1.x.\overline{y}$$
$$P_2 = r_2 \rhd \mathsf{merge}^-\ e_1.\mathsf{merge}^+\ e_2.\overline{x}$$
$$P_3 = r_3 \rhd \mathsf{merge}^-\ e_2.y$$

Session r_1 and r_2 join together on the entry-point e_1 and session r_2 and r_3 join on e_2. After two steps, S reduces to $(\nu r_1)(l_1 :: r_1 \rhd x.\overline{y} \mid l_2 :: r_1 \rhd \overline{x} \mid l_3 :: r_1 \rhd y)$, where $x.\overline{y}$, $\overline{x}$ and y run in the same session.

Example 2. The two-buyers-protocol below is inspired by an example in [6]. Let $T_2 = (\nu r_1, r_2)(l_1 :: \mathit{Buy}_1 \mid l_2 :: \mathit{Buy}_2 \mid l_s :: \mathit{Sell})$, where

$$\mathit{Buy}_1 = r_1 \rhd \mathsf{invoke}\ \mathit{sell}.\overline{\mathit{title}}.\mathit{quote}.\overline{\mathit{bid}}.Q_1$$
$$\mathit{Buy}_2 = r_2 \rhd \mathsf{invoke}\ \mathit{offer}.\mathit{title}.\mathit{quote}.\mathit{bid}.Q_2$$
$$\mathit{Sell} = \mathit{sell} \Rightarrow \mathit{title}.\mathsf{install}[\mathit{offer} \Rightarrow \mathsf{merge}^+\ e.(\overline{\mathit{title}}.\overline{\mathit{quote}}.\overline{\mathit{quote}}.Q)].\mathsf{merge}^-\ e$$

The service *sell* waits for a buyer to require a quote for a book (*title*). The buyer at l_1 invokes *sell* so that the new service *offer* is installed. Upon the invocation from the buyer at l_2, *offer* provides the book's *title* so that quotes are communicated to both buyers after the sessions are merged by the service. Finally, the buyers communicate over *bid* and the negotiation is concluded by the interactions among Q, Q_1 and Q_2 (not modeled here).

3 Types for Dynamic Multiparty Sessions

We consider types generated from the following CCS-like grammar:

$$\rho, \sigma ::= \mathsf{0} \ \mid\ c.\rho \ \mid\ \sigma|\rho \ \mid\ \beta \ \mid\ \mu\beta.\rho$$

$$\frac{}{\Gamma;\emptyset \vdash \mathbf{0} : \{\mathsf{0} \nearrow \mathsf{0}\}}\ (\textsc{Tzero}) \qquad \frac{\Gamma;\Delta \vdash P : \{\sigma \nearrow \rho\}}{\Gamma;\Delta \vdash c.P : \{c * \sigma \nearrow \rho\}}\ (\textsc{Taction}) \qquad \frac{}{\Gamma, X : \Phi;\Delta \vdash X : \{\Phi\}}\ (\textsc{Tvar})$$

$$\frac{\Gamma;\Delta \vdash P : \{\sigma_1|\sigma_2 \nearrow \rho\} \quad \Gamma(a^+) = \sigma \nearrow \rho' \quad \Gamma(a^-) = \mathsf{0} \nearrow \sigma_2}{\Gamma;\Delta \vdash \mathsf{invoke}\ a.P : \{\sigma|\sigma_1 \nearrow \sigma_2|\rho\}}\ (\textsc{Tinvoke}) \qquad \frac{\Gamma, X : \Phi;\Delta \vdash P : \{\Phi\}}{\Gamma;\Delta \vdash \mathsf{rec}\ X.P : \{\Phi\}}\ (\textsc{Trec})$$

$$\frac{\Gamma;\Delta \vdash P : \{\sigma_1|\sigma_2 \nearrow \rho\} \quad \Gamma(a^+) = \sigma_1 \nearrow \sigma_2 \quad \Gamma(a^-) = \mathsf{0} \nearrow \rho' \quad \Gamma;\Delta \vdash Q : \{\Phi\}}{\Gamma;\Delta \vdash \mathsf{install}[a \Rightarrow P].Q : \{\Phi\}}\ (\textsc{Tinstall})$$

$$\frac{\Gamma;\Delta \vdash P : \{\sigma_1|\sigma_2|\sigma_3 \nearrow \rho\} \quad \Gamma(e^p) = \sigma|\sigma_2 \nearrow \sigma_3 \quad \Gamma(e^{\overline{p}}) = \sigma'|\sigma'' \nearrow \rho' \quad \sigma \approx \sigma''}{\Gamma;\Delta \vdash \mathsf{merge}^p\ e.P : \{\sigma'|\sigma_1|\sigma'' \nearrow \sigma_2|\sigma_3|\rho\}}\ (\textsc{Tmerge})$$

$$\frac{\Gamma;\Delta_1 \vdash P : \{\sigma \nearrow \rho\} \quad \Gamma;\Delta_2 \vdash Q : \{\sigma' \nearrow \rho'\}}{\Gamma;\Delta_1|\Delta_2 \vdash P|Q : \{\sigma|\sigma' \nearrow \rho|\rho'\}}\ (\textsc{Tpar}) \qquad \frac{\Gamma;\Delta \vdash P : \{\sigma \nearrow \rho\}}{\Gamma;\Delta, r : \sigma \vdash r \rhd P : \{\mathsf{0} \nearrow \mathsf{0}\}}\ (\textsc{Tses})$$

$$\frac{\Gamma;\Delta \vdash P : \{\sigma_1|\sigma_2 \nearrow \rho\} \quad \Gamma(a^+) = \sigma_1 \nearrow \sigma_2 \quad \Gamma(a^-) = \mathsf{0} \nearrow \rho'}{\Gamma;\Delta \vdash l :: a \Rightarrow P : \{\mathsf{0} \nearrow \mathsf{0}\}}\ (\textsc{Tservice})$$

$$\frac{\Gamma;\Delta \vdash P : \{\Phi\}}{\Gamma;\Delta \vdash l :: P : \{\Phi\}}\ (\textsc{Tloc}) \qquad \frac{\Gamma;\Delta_1 \vdash S : \{\sigma \nearrow \rho\} \quad \Gamma;\Delta_2 \vdash T : \{\sigma' \nearrow \rho'\}}{\Gamma;\Delta_1|\Delta_2 \vdash S|T : \{\sigma|\sigma' \nearrow \rho|\rho'\}}\ (\textsc{Tspar})$$

$$\frac{\Gamma, n^+ : (\sigma \nearrow \rho), n^- : (\sigma' \nearrow \rho');\Delta \vdash S : \{\Phi\} \quad \rho \approx \rho'}{\Gamma;\Delta \vdash (\nu n)S : \{\Phi\}}\ (\textsc{Tnew}) \qquad \frac{\Gamma;\Delta, r : \sigma \vdash S : \{\Phi\} \quad \sigma \in \Downarrow_{\mathsf{0}}}{\Gamma;\Delta \vdash (\nu r)S : \{\Phi\}}\ (\textsc{TnewR})$$

Fig. 4. Type system

with the expected structural properties and labeled transition relation $\overset{c}{\mapsto}$.

Type judgments for processes take the form $\Gamma;\Delta \vdash P : \{\sigma \nearrow \rho\}$, meaning that: (i) P must perform communication activities in σ and ρ, (ii) it plans to interact as σ with the current participants of its session, (iii) it has delegated the interaction ρ to other endpoints that P itself will allow to join its session (via merge or service invocation). For a pair $\Phi = \sigma \nearrow \rho$, we call σ the *current* type, and ρ the *delegated* type.

The type environment Γ is a finite partial mapping from variables X and polarized service / entry-point names n^p (with $p \in \{+, -\}$) to type pairs $\sigma \nearrow \rho$, with the understanding that actions in ρ are delegated to $n^{\overline{p}}$. The *linear* session environment Δ is a finite partial mapping from session names r to types σ, such that $\Delta(r)$ represents the parallel composition of the current types of all endpoints of r. We assume $\Delta(r) = \mathsf{0}$ when $r \notin \Delta$. We let $\Delta_1|\Delta_2$ denote the environment Δ such that $\Delta(r) = \Delta_1(r)|\Delta_2(r)$ for each $r \in \Delta_1 \cup \Delta_2$

Our type judgments are in Fig. 4. They are parametric w.r.t. three notions: (1) task *separation* $c * \sigma$, (2) *type compatibility* $\approx$, (3) *session completion* $\Downarrow_{\mathsf{0}}$.

Task separation is used to project the activities of P in separate threads (in case they must be delegated). Here we take the most relaxed form of separation, where $c * \sigma = c|\sigma$. Other possibilities exploit prefixes in types to take care of causality information.

Type compatibility $\sigma \approx \rho$ says that σ and ρ are complementary. Let $I(\sigma) = \{c \mid \exists \sigma' : \sigma \overset{c}{\mapsto} \sigma'\}$ denote the set of initial actions σ can perform. Here we take the largest relation on types such that whenever σ is compatible with ρ it holds that either $I(\sigma) = I(\rho) = \emptyset$, or $K = I(\sigma) \cap \overline{I(\rho)} \neq \emptyset$ and, for each $x \in K$ and for each σ' and ρ' such that $\sigma \overset{x}{\mapsto} \sigma'$ and $\rho \overset{\overline{x}}{\mapsto} \rho'$, then σ' and ρ' are compatible.

The completion set $\Downarrow_0$ contains those types σ that express admissible interactions of multiple endpoints. Here we define $\Downarrow_0$ as the largest set of types σ such that: (i) for each $c \in I(\sigma)$ such that $\overline{c} \notin I(\sigma)$ and for each $\sigma \overset{\tau}{\mapsto} \sigma'$ there exists σ'' such that $\overline{c} \in I(\sigma'')$ and $\sigma' \overset{\tau}{\mapsto}{}^{*} \sigma''$, (ii) if $\sigma \overset{\tau}{\mapsto} \sigma'$ then $\sigma' \in \Downarrow_0$.

We say that Γ is *well-formed* if: (i) whenever $\Gamma(n^p) = \sigma \nearrow \rho$, then $\Gamma(n^{\overline{p}}) = \sigma' \nearrow \rho'$ for some $\rho' \approx \rho$, and (ii) whenever $\Gamma(a^-) = \sigma \nearrow \rho$, then $\sigma = \mathsf{0}$. We say that Δ is *fully-formed* if whenever $\Delta(r) = \sigma$, then $\sigma \in \Downarrow_0$. We say that a system S is *self-typeable* if $\Gamma; \Delta \vdash S : \{\mathsf{0} \nearrow \mathsf{0}\}$ for some well-formed Γ and fully-formed Δ. For simplicity, we shall omit the type rules for explicit fusion of session names and restrict to self-typeable systems S without nested sessions and with no free session name, i.e. such that $\Gamma; \emptyset \vdash S : \{\mathsf{0} \nearrow \mathsf{0}\}$ for some well-formed Γ.

Rule (TMERGE) best illustrates the flavor of our type system. Note that if Γ is well-formed, it must be $\rho' \approx \sigma_3$. Then: (1) P matches σ_3 to ρ', (2) P delegates σ_2 to the other endpoints of the merged session, (3) the other endpoints of the session of P delegate σ to the merging endpoint, and symmetrically (4) the merging endpoint and its partners delegate $\sigma'|\sigma''$ to the partners of P.

The systems T_1 and T_2 from § 2 are self-typeable. The type derivation for T_1 is reported below, where we omit some labels and rules due to space limitation.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma; \emptyset \vdash \mathbf{0} : \{\mathsf{0} \nearrow \mathsf{0}\}}\,(\textsc{Tzero})}{\Gamma; \emptyset \vdash \overline{x} : \{\overline{x} \nearrow \mathsf{0}\}}\,(\textsc{Taction})}{\Gamma; \emptyset \vdash \mathsf{merge}^{+}\ e_2.\overline{x} : \{\overline{x}|y \nearrow \mathsf{0}\}}\,(\textsc{Tmerge})}{\Gamma; \emptyset \vdash \mathsf{merge}^{-}\ e_1.\mathsf{merge}^{+}\ e_2.\overline{x} : \{\mathsf{0} \nearrow \overline{x}|y\}}\,(\textsc{Tmerge})}{\Gamma; \Delta_2 \vdash P_2 : \{\mathsf{0} \nearrow \mathsf{0}\}}\,(\textsc{Tses})}{\Gamma; \Delta_2 \vdash l_2 :: P_2 : \{\mathsf{0} \nearrow \mathsf{0}\} \quad (\dagger)}\,(\textsc{Tloc})
$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma; \emptyset \vdash \mathsf{merge}^{+}\ e_1.x.\overline{y} : \{\mathsf{0} \nearrow x|\overline{y}\}}}{\Gamma; \Delta_1 \vdash P_1 : \{\mathsf{0} \nearrow \mathsf{0}\}}}{\Gamma; \Delta_1 \vdash l_1 :: P_1 : \{\mathsf{0} \nearrow \mathsf{0}\}} \qquad \dfrac{(\dagger) \qquad \dfrac{\dfrac{\vdots}{\Gamma; \emptyset \vdash \mathsf{merge}^{-}\ e_2.y : \{\mathsf{0} \nearrow y\}}}{\vdots}}{\Gamma; \Delta_2|\Delta_3 \vdash l_2 :: P_2 \mid l_3 :: P_3 : \{\mathsf{0} \nearrow \mathsf{0}\}}}{\Gamma; \Delta \vdash l_1 :: P_1 \mid l_2 :: P_2 \mid l_3 :: P_3 : \{\mathsf{0} \nearrow \mathsf{0}\}}}{\dfrac{\vdots}{\Gamma; \emptyset \vdash T_1 : \{\mathsf{0} \nearrow \mathsf{0}\}}}
$$

where $\Gamma = \{\ e_1^{+} : (\mathsf{0} \nearrow x|\overline{y}),\ e_1^{-} : (\mathsf{0} \nearrow \overline{x}|y),\ e_2^{+} : (\mathsf{0} \nearrow \mathsf{0}),\ e_2^{-} : (y \nearrow \mathsf{0})\ \}$, $\Delta_1 = \{r_1 : \mathsf{0}\}$, $\Delta_2 = \{r_2 : \mathsf{0}\}$, $\Delta_3 = \{r_3 : \mathsf{0}\}$, and $\Delta = \{r_1 : \mathsf{0}, r_2 : \mathsf{0}, r_3 : \mathsf{0}\}$. The condition of $\approx$ for e_1 is satisfied since $x|\overline{y} \approx \overline{x}|y$ holds and the condition for e_2 is satisfied since $\mathsf{0} \approx \mathsf{0}$.

For T_2, we assume $Q \equiv Q_3|Q_4$ with $Q_1 \approx Q_3$ and $Q_2 \approx Q_4$, write b for *bid*, t for *title*, q for *quote*, and have $\Gamma;\emptyset \vdash T_2 : \{0 \nearrow 0\}$ for Γ shown below (together with some excerpts of type derivations).

$$\begin{aligned}
\Delta =&\ \{\ r_1 : 0,\ r_2 : 0\ \}\\
\Gamma =&\ \{\ \mathit{sell}^+ : (0 \nearrow b|t|\overline{q}|Q_3),\\
&\quad \mathit{sell}^- : (0 \nearrow \overline{b}|\overline{t}.q|Q_1),\\
&\quad \mathit{offer}^+ : (0 \nearrow \overline{b}|t|\overline{q}|Q_4),\\
&\quad \mathit{offer}^- : (0 \nearrow b|\overline{t}|q|Q_2),\\
&\quad e^- : (\overline{b} \nearrow 0),\\
&\quad e^+ : (\overline{q}|b|Q_3 \nearrow 0)\ \}
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma;\emptyset \vdash \overline{t}.q.\overline{b}.Q_1 : \{\overline{b}|\overline{t}|q|Q_1 \nearrow 0\}}}{\Gamma;\emptyset \vdash \mathsf{invoke}\ \mathit{sell}.\overline{t}.q.\overline{b}.Q_1 : \{0 \nearrow \overline{b}|\overline{t}|q|Q_1\}}}{\Gamma;\Delta \vdash r_1 \rhd \mathsf{invoke}\ \mathit{sell}.\overline{t}.q.\overline{b}.Q_1 : \{0 \nearrow 0\}}$$

Similarly, $\Gamma;\Delta \vdash r_2 \rhd \mathsf{invoke}\ \mathit{offer}.\overline{t}.q.b.Q_2 : \{0 \nearrow 0\}$

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma;\emptyset \vdash t.\overline{q}.\overline{q}.(Q) : \{\overline{q}|t|\overline{q}|Q \nearrow 0\}}}{\Gamma;\emptyset \vdash \mathsf{merge}^+\ e.(t.\overline{q}.\overline{q}.(Q)) : \{\overline{b}|t|\overline{q}|Q_4 \nearrow \overline{q}|Q_3\}}}{\Gamma;\emptyset \vdash \mathit{offer} \Rightarrow \mathsf{merge}^+\ e.(t.\overline{q}.\overline{q}.(Q)) : \{0 \nearrow 0\}} \qquad \dfrac{\Gamma;\emptyset \vdash \mathbf{0} : \{0 \nearrow 0\}}{\Gamma;\emptyset \vdash \mathsf{merge}^-\ e : \{b|\overline{q}|Q_3 \nearrow 0\}}}{\Gamma;\emptyset \vdash \mathit{sell} \Rightarrow t.\mathsf{install}[\mathit{offer} \Rightarrow \mathsf{merge}^+\ e.(t.\overline{q}.\overline{q}.(Q))].\mathsf{merge}^-\ e : \{0 \nearrow 0\}}$$

This is still a preliminary study, but we conjecture that self-typeability enjoys subject reduction and that any non self-typeable system has at least one session that can deadlock on some pending communication. On the other hand, we point out that some self-typeable systems can deadlock, because, e.g., rule (TACTION) relaxes the order of execution of actions, and the type system checks neither that the number of positive merges on an endpoint is the same as the number of negative merges on such endpoint nor that the number of invocations to a service is less than the number of times such service is available / installed. We guess that stronger guarantees can be obtained by restricting the syntax of endpoints, by tuning the definitions of $c * \sigma$, $\approx$ and $\Downarrow_0$, and by extending the types with capability / obligation level annotations à la Kobayashi [7].

References

1. E. Bonelli and A. Compagnoni. Multipoint session types for a distributed calculus. *TGC'07*, vol. 4912 of *LNCS*, pp. 240–256. Springer, 2008.
2. M. Boreale, *et al.* SCC: a service centered calculus. *WS-FM'06*, vol. 4184 of *LNCS*, pp. 38–57. Springer, 2006.
3. R. Bruni, I. Lanese, H. Melgratti, and E. Tuosto. Multiparty sessions in SOC. *COORDINATION'08*, vol. 5052 of *LNCS*, pp. 67–82. Springer, 2008.
4. M. Carbone, K. Honda, and N. Yoshida. Structured communication-centred programming for web services. *ESOP'07*, vol. 4421 of *LNCS*, pp. 2–17. Springer, 2007.
5. K. Honda, V. Vasconcelos, and M. Kubo. Language primitives and type disciplines for structured communication-based programming. *ESOP'98*, vol. 1381 of *LNCS*, pp. 22–138. Springer, 1998.
6. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. *POPL'08*, pp. 273–284. ACM, 2008.
7. N. Kobayashi. Typical tool. http://www.kb.ecei.tohoku.ac.jp/~koba/typical/.
8. I. Lanese, V. Vasconcelos, F. Martins, and A. Ravara. Disciplining orchestration and conversation in service-oriented computing. *SEFM'07*, pp. 305–314. IEEE, 2007.