Reversible Semantics in Session-based Concurrency*

Claudio Antares Mezzina[1] and Jorge A. Pérez[2]

[1] IMT School for Advanced Studies Lucca, Italy
[2] University of Groningen, The Netherlands

Abstract. Much research has studied foundations for correct and reliable communication-centric systems. A salient approach to correctness uses session types to enforce structured communications; a recent approach to reliability uses reversible actions as a way of reacting to unanticipated events or failures.
This note describes recent work that develops a simple observation: the machinery required to define monitored semantics can also support reversible protocols. We illustrate a process framework of session communication in which monitors support reversibility. A key novelty in our approach are session types with present and past, which allow us to streamline the semantics of reversible actions.

1 Introduction

The purpose of this short paper is to motivate and describe our ongoing work in reversible models of structured communications [8]. Framed within concurrency theory and process calculi approaches, we are interested in developing rich models of concurrent computation in which communicating processes follow structured interaction protocols (as described by *session types* [3]), and whose underlying operational semantics admits the possibility of reversing their actions. This integration of structured communication and reversibility should inform the design of sound programming abstractions for resilient communicating programs governed by casual consistent semantics.

Models of reversible computation and structured communications have received much attention (cf. [1,3]). Reversing computational steps is an appealing feature in different scenarios; for instance, in the case of a failure in a (concurrent) program or transaction, we might like to undo all steps leading to the failure, so

* Partially supported by EU COST Actions IC1201 (Behavioral Types for Reliable Large-Scale Software Systems) and IC1405 (Reversible Computation - Extending Horizons of Computing).

V. Biló, A. Caruso (Eds.): ICTCS 2016, Proceedings of the 17th Italian Conference on Theoretical Computer Science, 73100 Lecce, Italy, September 7–9 2016, pp. 221–226 published in CEUR Workshop Proceedins Vol-1720 at `http://ceur-ws.org/Vol-1720`

as to return to last known stable state of the system. Indeed, good examples of how reversibility can be used to model transactional models are [2,6]. The design of reversible semantics for models of concurrency is a delicate issue, for we would like to undo computational steps in a *causally consistent* fashion: a step should be undone only if all its causes (if any) have been already undone. In this way, reversibility in a causal consistent model leads to a system state that could have been reached by performing forward steps only.

The key observation that underlies our work is that the design of casually consistent operational semantics for concurrent processes can take advantage of the structured protocols that typically govern their behavior. As session types abstract sequences of communication actions (protocols), they appear as a natural choice for recording the forward and backward actions of interacting processes.

In recent work, we have started to formalize the integration of reversibility and session-based concurrency [8]. In this note, we illustrate the model in [8] by means of a simple example that contains the main ingredients of our approach, namely an operational semantics for untyped processes which is instrumented by *monitors* that contain protocols specified as session types. In order to support both forward and backward steps, we consider session types that describe both *past* and *present* protocol states.

2 Reversible Sessions, By Example

Our proposal builds upon the approach of models of concurrency such as the π-calculus. As such, main ingredients in our approach are *configurations*, *processes*, and *(protocol) types*, whose syntax is given in Figure 1. We assume a language of the expressions $e, e', \ldots$ that includes basic values, variables, and functions on them. The syntax of configurations includes the empty configuration $\mathbf{0}$, name restriction $\nu n.M$, parallel composition $M \parallel N$, but also *running processes* and *monitors*:

- A running process $\langle P \cdot \sigma \cdot \widetilde{u} \rangle_{\widetilde{s}}$ is univocally identified by $\widetilde{s}$, the list of session endpoints occurring in P. The local store σ is a list of pairs of the form $\{x, \widetilde{v}\}$ (i.e., a set of mappings from variables to values); the list $\widetilde{u}$ collects the subjects of actions already performed by P.
- Given a session name s, a monitor $s\lfloor H \cdot \widetilde{e} \rfloor$ contains a protocol (session) type H that describes the structured behavior associated to s (see below) and a list $\widetilde{e}$ containing all the expressions (including variables) used by the process.

Intuitively, the list $\widetilde{u}$ in the running process and the list $\widetilde{e}$ in the monitor will be used to record previously performed actions and reconstruct the process structure accordingly.

The design of the operational semantics for our model is inspired by the approach of [5], in which session types are used as *monitors* that enable communication actions: a synchronization can only occur if the process actions correspond to the intended protocols given by the monitor types. After synchronization, portions of both processes and monitor types are consumed. Our approach consists

$$
\begin{array}{llll}
(\textit{configurations}) & M, N ::= & \mathbf{0} \mid \langle P \cdot \sigma \cdot \widetilde{u} \rangle^{\sim}_{s} \mid s\lfloor H \cdot \widetilde{e} \rfloor \mid \nu n.M \mid M \parallel N & \\
(\textit{processes}) & P, Q ::= & u(x : S).P \mid u\langle x : S \rangle.P \mid k\langle e \rangle.P \mid k(x).P \mid \nu a.P \mid \mathbf{0} & \\
(\textit{actions}) & \alpha, \beta ::= & !U \mid ?U \qquad (\textit{protocol types}) & S, T ::= \quad \mathtt{end} \mid \alpha.S \\
(\textit{history types}) & H, K ::= & \hat{}\, S \mid S\hat{} \mid \alpha_1.\cdots.\alpha_n.\hat{}\, S &
\end{array}
$$

Fig. 1. Syntax of Configurations, Processes, and Types.

in keeping, rather than consuming, these monitor types. For this to work, we need to distinguish the part of the protocol that has been already executed (its past), from the protocol that still needs to execute (its present). We thus introduce session types with present and past (H in the syntax): the type $\alpha_1.\cdots.\alpha_n.\hat{}\, S$ says that actions $\alpha_1, \cdots, \alpha_n$ are past protocol actions, whereas actions in protocol S are yet to be executed. That is, the cursor $\hat{}$ in history types helps us to distinguish the past from the present. Each action α_i corresponds to the input or output of a value; we use U to range over the types of these values (e.g., **int**, **str**, etc.).

We illustrate our approach by means of a simple business protocol example [4]: a slightly modified version of the *two buyers protocol*. It involves three participants: a Buyer, a Seller, and a Buyer's Friend. Buyer is willing to buy a book, and sends to Seller the title of the book he is interested in. Seller replies with the price of the book, and awaits for final information (e.g., *shipping address* and *order confirmation*) from Buyer, before providing a *delivery date*. Once Buyer receives the price, he realizes that he needs a loan from Friend in order to finalize the purchase. To this aim, Buyer contacts Friend and then the transaction is finalized. The set of interactions of Buyer with Seller and Friend are prescribed by the following session types:

$$
\begin{array}{ll}
S_a : ?\mathbf{str}.!\mathbf{int}.?\mathbf{str}.?\mathbf{int}.!\mathbf{cal}.\mathtt{end} & S_b : ?\mathbf{int}.!\mathbf{int}.\mathtt{end} \\
T_a : !\mathbf{str}.?\mathbf{int}.!\mathbf{str}.!\mathbf{int}.?\mathbf{cal}.\mathtt{end} & T_b : !\mathbf{int}.?\mathbf{int}.\mathtt{end}
\end{array}
$$

Above, S_a describes the interaction between Buyer and Seller from Seller's perspective; type T_a is its *dual* and describes the protocol from Buyer's perspective. In session types, duality is essential to (statically) ensure action compatibility between partners (and therefore, to guarantee absence of communication errors). Types T_b and S_b describe the interaction between Buyer and Friend, from each perspective.

Having defined the interaction protocols using types, we proceed to examine some possible process implementations for Buyer, Seller, and Friend. The behavior

of Buyer may be specified by the following process:

$$\textsc{Buyer} \triangleq a\langle z : T_a\rangle.z\langle\text{“dune”}\rangle.z(prc).\\ b\langle w : T_b\rangle.w\langle \texttt{loan}(prc)\rangle.w(cash).z\langle addr\rangle.z\langle cash\rangle.z(date).\mathbf{0}$$

The implementation for Buyer involves the creation of two interleaved sessions: the first one is established with the prefix $a\langle z : T_a\rangle$, which explicitly mentions the session protocol to be executed with the implementation of Seller; the second session is established with the implementation of Friend through the prefix $b\langle w : T_b\rangle$. Process implementations for Seller and Friend can be specified by the following processes:

$$\textsc{Seller} \triangleq a(z : S_a).z(title).z\langle\texttt{quote}(title)\rangle.z(addr).z(paymnt).z\langle\texttt{date}(addr)\rangle.\mathbf{0}$$
$$\textsc{Friend} \triangleq b(w : S_b).w(amount).w\langle loan\rangle.\mathbf{0}$$

Note that functions `loan`(), `quote`() and `date`() are used to calculate the amount of money to be borrowed, the book price and the delivery date, respectively. The overall system specification is then given by the parallel composition of configurations containing the three processes (in what follows, ϵ denotes the empty list):

$$\textsc{System} \triangleq \left\langle \textsc{Buyer}\cdot\epsilon\cdot\epsilon\right\rangle_\epsilon \parallel \left\langle \textsc{Seller}\cdot\epsilon\cdot\epsilon\right\rangle_\epsilon \parallel \left\langle \textsc{Friend}\cdot\epsilon\cdot\epsilon\right\rangle_\epsilon$$

In the following, we will indicate with $\textsc{Buyer}_i$ (resp. $\textsc{Seller}_i$ and $\textsc{Friend}_i$) the process Buyer after performing its i-th action. We will do the same with types.

The operational semantics that we have defined in [8] is based on a reduction relation with both forward and backward steps, denoted $\twoheadrightarrow$ and $\rightsquigarrow$, respectively. The first forward reduction of System is establishing a session between Buyer and Seller, using the fact that T_a and S_a are dual types. We have:

$$\textsc{System} \twoheadrightarrow (\nu s,\overline{s}).\Big(\left\langle \textsc{Buyer}_1\cdot\{z,s\}\cdot a\right\rangle_s \parallel s\lfloor \frown T_a\cdot z\rfloor \parallel \\ \left\langle \textsc{Seller}_1\cdot\{z,\overline{s}\}\cdot a\right\rangle_{\overline{s}} \parallel \overline{s}\lfloor \frown S_a\cdot z\rfloor \parallel \left\langle \textsc{Friend}\cdot\epsilon\cdot\epsilon\right\rangle_\epsilon\Big) \tag{1}$$

As we can see, once a session is established two monitors are created, one per endpoint; their task is to discipline the behavior of the process holding the endpoint. For example, the behavior of Buyer in session s has to obey type S_a. Buyer then sends (according to S_a) to Seller the request for the book, and the entire system evolves as:

$$\twoheadrightarrow(\nu s,\overline{s}).\Big(\left\langle \textsc{Buyer}_2\cdot(\{z,s\})\cdot a,z\right\rangle_s \parallel s\lfloor !\texttt{str}\frown T_{a_1}\cdot z,\text{“dune”}\rfloor \parallel \\ \left\langle \textsc{Seller}_2\cdot(\{z,\overline{s}\},\{title,\text{“dune”}\})\cdot a,z\right\rangle_{\overline{s}} \parallel \\ \overline{s}\lfloor ?\texttt{str}\frown S_{a_1}\cdot z,title\rfloor \parallel \left\langle \textsc{Friend}\cdot\epsilon\cdot\epsilon\right\rangle_\epsilon\Big) = M \tag{2}$$

As effect of the communication, both types register the action and move forward. Another effect is that the information needed to restore back the consumed prefixes is stored into the running configurations and the related monitors. Communication in (2) can be reverted by moving backward the monitor types, by restoring the prefixes and deleting the read value from the receiver store, that is:

$$M \rightsquigarrow (\nu s, \overline{s}).\Big(\big\langle z\langle \text{“dune”}\rangle.\textsc{Buyer}_2 \cdot (\{z,s\} \cdot a\big\rangle_s \parallel s\lfloor \frown !\texttt{str}.T_{a_1} \cdot z\rfloor \parallel$$
$$\big\langle z(\mathit{title}).\textsc{Seller}_2 \cdot \{z,\overline{s}\} \cdot a\big\rangle_{\overline{s}} \parallel \overline{s}\lfloor \frown ?\texttt{str}.S_{a_1} \cdot z\rfloor \parallel \big\langle \textsc{Friend} \cdot \epsilon \cdot \epsilon\big\rangle_{\epsilon}\Big) \tag{3}$$

We can easily check that the configurations in (3) and (1) are equivalent. From M in (2) the interaction between Buyer and Seller goes on, and the system arrives to a point in which Buyer establishes a new session with Friend:

$$M \twoheadrightarrow^* (\nu s, \overline{s}, r, \overline{r}).\Big(\big\langle \textsc{Buyer}_4 \cdot (\{z,s\},\{w,r\}) \cdot \widetilde{u}_1, b\big\rangle_{s,r} \parallel$$
$$r\lfloor \frown T_b \cdot b\rfloor \parallel s\lfloor T'_a \frown T_{a_3} \cdot z, \text{“dune”}, \mathit{prc}, w\rfloor \parallel$$
$$\big\langle \textsc{Seller}_3 \cdot (\{z,\overline{s}\},\{\mathit{title}, \text{“dune”}\}) \cdot a, z, z\big\rangle_{\overline{s}} \parallel$$
$$\overline{s}\lfloor S'_a \frown S_{a_3} \cdot z, \mathit{title}, \texttt{quote}(\mathit{title})\rfloor \parallel \big\langle \textsc{Friend}_1 \cdot \{w,\overline{r}\} \cdot b\big\rangle_{\overline{r}} \parallel \overline{r}\lfloor \frown S_b \cdot w\rfloor\Big) \tag{4}$$

As (4) shows, the running process for Buyer is present in two sessions: one with Seller and another one with Friend, and has two associated monitors, identified by endpoints s, r. The list of subjects stored into the running process allows us to reverse communications (possibly in different sessions) and session establishments in the order in which they were performed, thus respecting causality of actions. In this way, Buyer cannot undo a communication with Seller while the session with Friend is still established.

3 Future Work

We have described recent work on the integration of reversible semantics and session-based concurrency [8]. It represents a fresh approach with respect to previous approaches [9]. Several directions deserve further investigation:

- *Richer (typed) languages.* The process model in [8] is admittedly simple; to model and reason about interesting examples we need support for constructs such as labeled choices. Also, process specifications do not specify reversible actions; this is the role of monitors, history types, and other mechanisms. Since reversibility is independent from specifications, rich types are needed to support controlled forms of reversibility. In recent work we propose alternatives to these challenges [7].

– *Multiparty session communications.* The model in [8] concerns *binary* session types, which codify interaction between exactly two partners. Generalizing our approach to *multiparty* session types [4] should require a finer, coordinated representation of reversible actions, as protocol exchanges may involve more than two participants.
– *Dedicated reasoning techniques.* Session types induce a "simpler" model of concurrency in which reversibility is a better behaved phenomenon. It remains to be seen to what extent such a setting enables the development of tractable reasoning techniques (e.g., axiomatizations, behavioral equivalences, and proof systems).

References

1. Danos, V., Krivine, J.: Reversible communicating systems. In: Proc. of CONCUR 2004. pp. 292–307. LNCS, Springer (2004)
2. Danos, V., Krivine, J.: Transactions in RCCS. In: CONCUR 2005. LNCS, vol. 3653, pp. 398–412 (2005)
3. Honda, K., Vasconcelos, V.T., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: ESOP'98. LNCS, vol. 1381, pp. 122–138. Springer (1998)
4. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: POPL 2008. pp. 273–284. ACM (2008)
5. Kouzapas, D., Yoshida, N., Honda, K.: On asynchronous session semantics. In: Proc. of FMOODS 2011 and FORTE 2011. LNCS, vol. 6722, pp. 228–243. Springer (2011)
6. Lanese, I., Lienhardt, M., Mezzina, C.A., Schmitt, A., Stefani, J.B.: Concurrent flexible reversibility. In: ESOP 2013. LNCS, vol. 7792, pp. 370–390 (2013)
7. Mezzina, C.A., Pérez, J.A.: Reversibility in session-based concurrency: A fresh look (2016), draft available in `http://www.jperez.nl`
8. Mezzina, C.A., Pérez, J.A.: Reversible sessions using monitors. In: Proc. of PLACES 2016. EPTCS, vol. 211, pp. 56–64 (2016), `http://dx.doi.org/10.4204/EPTCS.211.6`
9. Tiezzi, F., Yoshida, N.: Reversible session-based pi-calculus. J. Log. Algebr. Meth. Program. 84(5), 684–707 (2015)