A Process Calculus of Atomic Commit

Laura Bocchi [1] Lucian Wischik [2]

Department of Computer Science
University of Bologna, Italy
www.wischik.com/lu/research/acommit.html

Abstract

This article points out a strong connection between process calculi and atomic commit. Process calculus rendezvous is an abstract semantics for atomic commitment. An implementation of process-calculus rendezvous is an atomic commit protocol. Thus, the traditional correctness properties for atomic commit are entailed by a bisimulation proof of a calculus implementation.

Actually, traditional rendezvous as found in the pi calculus corresponds to just a special case of atomic commit called a binary cohesion. If we take the general case of atomic commit, this induces a richer form of calculus rendezvous similar to the join calculus [10]. As an extended example of the analogy between calculus and atomic commit, we use the induced calculus to reformulate an earlier 2PCP correctness result by Berger and Honda [1].

Key words: synchronous rendezvous, pi calculus bisimulation, atomic commit protocol, 2PCP

1 Introduction

This article outlines a strong connection between process calculi and atomic commit. Process calculi are models or languages for concurrent and distributed interactive systems. They also underpin emerging Web Service standards [7,8,11]. Process calculi work through *synchronous rendezvous*, where one party (such as a web service) waits for data over a communication channel; another party (the client) sends data over the channel; once the data has been exchanged, both parties continue. The canonical process calculus is the pi calculus [12].

Transactions and calculi have met in recent years. First, process calculi have been used to formalize protocols used in transactions such as two-phase

[1] Email: bocchi@cs.unibo.it
[2] Email: lu@wischik.com

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

commit [1], compensation triggering [4,6], nesting [5] and consensus [13]. Second, features from transactions have been added to process calculi: [3] adds primitives for Atomicity, Isolation and Durability, and [4] adds a primitive for compensational transactions.

This paper describes a more fundamental connection between process calculi and transactions: the basic synchronous rendezvous mechanism *itself* is already a special case of atomic commit. This means that an atomic commit protocol constitutes an implementation of a process calculus, and an implementation of a process calculus constitutes a special case of an atomic commit protocol – we remark that traditional synchronous rendezvous is just a special case of atomic commit, 'binary cohesor'. Moreover, the elegant concept of bisimulation between a process calculus and its implementation *entails* the ad-hoc collection of correctness properties that is normally given for atomic commit.

All this gives rise to an appealing possibility. (1) Synchronous rendezvous is a high level semantics for atomic commit, with a concise notation. (2) Web service languages already borrow *asynchronous* rendezvous from process calculi. (Asynchronous rendezvous is when the sender does not wait for its message to be consumed). (3) Web service languages also talk about atomicity of transactions. Putting these together, maybe we could create (4) a new web service language which uses synchronous as well as asynchronous rendezvous. Such a language would not need to mention atomicity (since atomicity would be implicit in its rendezvous). It would also enjoy strong formal links with process calculi.

As mentioned, traditional synchronous rendezvous from the pi calculus is a special case of atomic commit. We will see that general atomic commit induces a richer rendezvous mechanism, similar to the 'join' of the join calculus [10]. Moreover cohesive commitment – a general case of atomic commitment – induces the standard choice operator in process calculi.

It is commonly said that synchronous rendezvous amounts to solving distributed consensus. This is not true: rendezvous is strictly easier. In particular, distributed consensus blocks if any of the participating parties have crashed; a binary cohesion need not block, so long as two parties remain active.

The plan of the paper is as follows. Sections 1 and 2 present traditional rendezvous, and then traditional atomic commit. The latter is presented in a non-standard manner that reveals the correspondence. Section 3 gives a process calculus with a richer form rendezvous, which corresponds to arbitrary atomic commit. Section 4 revisits the two phase commit protocol of [1], recasting its correctness statement as a bisimulation with this richer process calculus. In doing this we provide a formal link between the protocol and the traditional correctness properties of atomic commit.

The chief original contribution of this paper is to reveal formally the link between atomic commit and rendezvous.

2 Rendezvous

We give a brief summary of rendezvous, as used in the pi calculus. It assumes the following setting. There are N distributed parties that execute their code. A party may eventually arrive at a send command $\overline{u}\,x$ (indicating a wish to send data x over channel u) whereupon it blocks. Or a party may arrive at a receive command $u(y)$ (a wish to receive formal parameter y over channel u) whereupon it also blocks. If there is a sender and a receiver blocked on a channel, they may eventually *rendezvous* – exchange data and unblock. The rendezvous is strictly between two parties. Even if there were several parties willing to receive on the channel, only one of them does so.

Definition 2.1 (Pi calculus) *Assume a set $\mathcal{N}$ of names ranged over by $x, y, \ldots$. Write $\widetilde{u}$ for a sequence $u_1 \ldots u_n$. Pi calculus terms P are*

$$P ::= \mathbf{0} \quad | \quad \overline{u}\,x.P \quad | \quad u(x).P \quad | \quad \nu x.P \quad | \quad P|P.$$

The operators $\nu x.P$ and $u(x).P$ bind x. We identify programs up to alpha-renaming of bound names and up to structural congruence $\equiv$ as below, closed under contexts:

$$P|\mathbf{0} \equiv P \qquad P|Q \equiv Q|P \qquad P|(Q|R) \equiv (P|Q)|R \qquad \text{(structure)}$$
$$\nu x.\nu y.P \equiv \nu y.\nu x.P \qquad \nu x.(P|Q) \equiv P|\nu x.Q \text{ if } x \notin \text{fn}\,P \qquad \text{(scope)}$$

The reaction relation $\rightarrow$, also called 'rendezvous', is the smallest relation satisfying $\overline{u}\,\widetilde{x}.P \mid u(\widetilde{y}).Q \rightarrow P|Q\{\widetilde{x}/\widetilde{y}\}$ and closed with respect to $\equiv$ and contexts. Observations $P \downarrow u$ are given by

$$\overline{u}\,\widetilde{x}.P \downarrow u \qquad u(\widetilde{x}).P \downarrow u \qquad P|Q \downarrow u \text{ if } P \downarrow u \text{ or } Q \downarrow u$$
$$\nu x.P \downarrow u \text{ if } P \downarrow u \text{ and } u \neq x$$

A pi calculus rendezvous can be represented diagrammatically. In the following, write $\overline{u}\,x.P|u(y).Q|R$ to refer generically to any state which has some complementary output $\overline{u}\,x.P$ and input $u(y).Q$, and R' for a state with none.

$$\begin{array}{ll} \overline{u}\,x.P|u(y).Q|R & R' \\ \Downarrow & \\ P|Q\{x/y\}|R & \end{array} \qquad (1)$$

There is a standard technique in process calculi, *bisimulation*, for judging whether two transition systems have the same interactive behaviour:

Definition 2.2 (Bisimulation) *Let $(\mathcal{P}, \rightarrow_p, \downarrow_p)$ and $(\mathcal{Q}, \rightarrow_q, \downarrow_q)$ be two transition systems, each equipped with an observation relation. Let $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{Q}$ relate their states. Write $\Rightarrow$ for $\rightarrow^*$ and $\Downarrow$ for $\Rightarrow\downarrow$. Then $\mathcal{S}$ is a bisimulation if and only if whenever $P\ \mathcal{S}\ Q$ then*

- *$P \Downarrow_p u$ if and only if $Q \Downarrow_q u$;*

- $P \Rightarrow_p P'$ *implies* $Q \Rightarrow_q Q'$ *with* $P' \mathcal{S} Q'$*;*
- $Q \Rightarrow_q Q'$ *implies* $P \Rightarrow_p P'$ *with* $P' \mathcal{S} Q'$*.*

Two terms $P \in \mathcal{P}$ *and* $Q \in \mathcal{Q}$ *are called* bisimilar *when there exists a bisimulation between them. Write* $\dot{\approx}$ *for the largest bisimulation.*

For instance, taking $\mathcal{P}$ and $\mathcal{Q}$ both from the pi calculus, then $A = \overline{u}\,x \mid u(y).P$ and $B = \overline{u}\,x \mid \nu u'.(u(y).\overline{u}\,'y|u'(y).P)$ are bisimilar, via $\mathcal{S}= \{(A, B)\} \cup \mathbf{I}$.

3 Atomic Commit

We give a brief summary of atomic commitment. This summary is unconventional: it has been structured to reveal the similarity with rendezvous.

A typical setting of atomic commit is in a distributed database system. When a transaction is proposed, some distributed parts might be able or unable to accept the transaction. The transaction can only be committed only if everyone was able. Let us suppose N independent parties. Each of them

- is either able or unable;
- when asked to vote on whether the transaction should go ahead, votes either *yes* (if the party is able to accept the transaction) or *no* (if it is unable);
- reaches a decision that is either *commit* or *abort*, and terminates.

It is required that all distributed parties reach the same decision (commit or abort). We call this the 'collective decision.' In order to achieve this unanimity the parties engage in a protocol between themselves, sending their votes and exchanging messages. It is necessary for the protocol to take into account the possibility that some some messages are lost (communication failure) or some parties crash (site failure).

Formally, for a protocol to be called an 'atomic commit' protocol it must satisfy the following standard properties [2]:

AC1. All parties that reach a decision reach the same one.

AC2. A party cannot reverse its decision after it has reached one.

AC3. Commit can only be reached if all parties voted *yes* (ie. were able).

AC4. If all parties were able and there were no failures, the collective decision will be to commit.

AC5. At any stage, if all failures are repaired, then all parties will eventually reach a decision. (This is also referred to as weak-termination or liveness).

There are other properties which are implied by existing literature, and satisfied by existing protocols:

AC6. A party cannot change from able to unable, or vice versa.

AC7. If there are enough failures, the collective decision will be to abort.

Consider a protocol as a set of states with transitions $\rightarrow$ between them. A transition might be the an emission of a message, or a simultaneous broadcast and receipt (in the case of instantaneous communication), or a message-loss event or site failure. We will partition all the possible states into the following disjoint and exhaustive partitions. Write (x, y) for a state where x parties have decided to commit and y have decided to abort. Write $(x, y)a$ if all parties are able, $(x, y)u$ if some are unable. Let i, j range over $1 \ldots N-1$. The partitions are:

- $(0, 0)a$ – all able
- $(0, 0)u$ – some unable
- $(i, 0)$
- $(0, j)a$ – all able
- $(0, j)u$ – some unable
- $(N, 0)$
- $(0, N)a$ – all able
- $(0, N)u$ – some unable
- (i, j), where $i + j<N$
- (i, j), where $i + j=N$.

Each of the properties imposes constraints on which inter-state transitions are disallowed, or which must exist. For instance, *AC4* means there exists a sequence $(0, 0)a \Rightarrow (N, 0)$. *AC2* means there is no sequence of transitions $(0, N) \Rightarrow (N, 0)$. Table 1 summarises all the transitions. (By *AC1*, the two states $(i, j) : i+j<N$ and $(i, j) : i+j=N$ are never reached; the have therefore been omitted from the table.)

To explain, the table shows whether some sequence of transitions $\Rightarrow$ *from* a state (row) *to* a state (column) is possible or not. For instance, a sequence from $(0, 0)a \Rightarrow (N, 0)$ is possible due to *AC4*. This does not mean that the sequence will necessarily be taken; it merely means that it is possible, supposing lucky nondeterministic choices and no failure. The table marks these possibilities in bold, with the name of the rule which indicates that possibility. By definition, every state admits the empty sequence of transitions back to itself, which is denoted **id**. Where one of the properties rules out a transition, this is indicated by a cross. We remark on the use of *AC5* (weak-termination). One instance of this, which we denote (a), is that from $(0, 0)u$ it is possible to reach either $(N, 0)$ or $(0, N)a$ or $(0, N)u$. It happens that $(N, 0)$ is ruled out due to *AC3* and $(0, N)a$ is ruled out due to *AC6*. The only possibility left is $(0, N)u$. The table has been annotated with superscripts $a \ldots d$ to show four separate applications of *AC5*.

Some spaces in the table are left unspecified, such as for $(0, 0)a \Rightarrow (i, 0)$. Hence a protocol is allowed to include the transition, or allowed to omit it.

Definition 3.1 (Protocol) *An atomic commit protocol is any transition sys-*

from\to	$(0,0)a$	$(0,0)u$	$(i,0)$	$(0,j)a$	$(0,j)u$	$(N,0)$	$(0,N)a$	$(0,N)u$
$(0,0)a$	**id**	A$\not$C6			A$\not$C6	**AC4**	**AC7**	A$\not$C6
$(0,0)u$	A$\not$C6	**id**	A$\not$C3	A$\not$C6		A$\not$C3^a	A$\not$C6^a	**AC5**^{a}
$(i,0)$	A$\not$C2	A$\not$C3	**id**	A$\not$C2	A$\not$C2	**AC5**^{b}	A$\not$C2^b	A$\not$C2^b
$(0,j)a$	A$\not$C2	A$\not$C6	A$\not$C2	**id**	A$\not$C6	A$\not$C2^c	**AC5**^{c}	A$\not$C6^c
$(0,j)u$	A$\not$C6	A$\not$C2	A$\not$C2	A$\not$C6	**id**	A$\not$C2^d	A$\not$C6^d	**AC5**^{d}
$(N,0)$	A$\not$C2	A$\not$C2	A$\not$C2	A$\not$C2	A$\not$C2	**id**	A$\not$C2	A$\not$C2
$(0,N)a$	A$\not$C2	A$\not$C6	A$\not$C2	A$\not$C2	A$\not$C2	A$\not$C2	**id**	A$\not$C6
$(0,N)u$	A$\not$C6	A$\not$C2	A$\not$C2	A$\not$C2	A$\not$C6	A$\not$C2	A$\not$C6	**id**

Table 1
Possible transitions for an atomic commit protocol

tem $(\mathcal{P}, \to, \downarrow)$ *where* $\mathcal{P}$ *can be partitioned into the eight states listed above, and where transition-sequences* $P \Rightarrow P'$ *are allowed or disallowed as per the table above.*

Write $P \downarrow a$ *in a state where all parties are able,* $P \downarrow u$ *in a state where some are unable,* $P \downarrow cmt$ *in a state where one or more parties have decided to commit, and* $P \downarrow abt$ *in a state where one or more have decided to abort.*

In particular, the following is one possible atomic-commit protocol.

$$\begin{array}{ccc} (0,0)a & & (0,0)u \\ \Downarrow \quad \searrow & & \Downarrow \\ (N,0) \quad (0,N)a & & (0,N)u \end{array} \tag{2}$$

This is an obvious diagram that corresponds to our intuition of what an atomic commitment protocol is: it either succeeds, or it fails! One might think of it as a high-level semantics for atomic commit. The following theorem justifies this:

Theorem 3.2 (Correctness) *A transition system* $(P, \to, \downarrow)$ *is an atomic-commit protocol if and only if it is bisimilar to Diagram 2.*

Proof. Straightforward table lookup.

There is one further refinement to make in Diagram 2. The intended meaning of 'abort' is to return a party to the state it was before the attempted commitment. It is thus able to entertain further commitments. This can be represented by transitions $(0,N)a \to (0,0)a$ and $(0,N)u \to (0,0)u$. With

these transitions the diagram degenerates, and $AC7$ is redundant:

$$\begin{array}{ccc} (0,0)a & & (0,0)u \\ \Downarrow & & \\ (N,0) & & \end{array} \tag{3}$$

Remark that Diagrams 1 and 3 are basically the same.

4 General Rendezvous

The preceding section has taken bisimulation from the process calculi arena, and applied it to atomic commit. In this section instead we take general commitment and use it to induce a general rendezvous mechanism.

Use the following analogy. An atomic commitment is like a rendezvous. When parties are able to commit, it is like having $\overline{u}\,x$ and $u(y)$ in the calculus. When a party decides to commit, it is like unblocking itself due to having performed a rendezvous.

Atomic commit allows multiple parties to participate, whereas pi rendezvous allows only two. This suggests to consider a different form of rendezvous which allows multiple parties, for instance

$$\overline{u}\,a \mid \overline{v}\,b \mid \overline{w}\,c \mid u(x)_\wedge v(y)_\wedge w(z).P \quad \to \quad P\{^{abc}/_{xyz}\}.$$

Write J for a *join-pattern* $u_1(\widetilde{x}_1).P_1{}_\wedge \ldots {}_\wedge u_n(\widetilde{x}_n).P_n$.

There is a further generalization to make, based on the commit inside a *cohesor* transaction. Cohesors are presented in BTP (see [9] for an overview) and a similar concept is also in WS-Transactions [8]. Normally, atomic commit requires that either all parties abort or they all commit. But a cohesor instead requires that either all abort, or some pre-defined subset commit while the rest abort. A cohesor has a list of acceptable subsets. As an example, consider a holiday booking which involves flying to Venice and hiring a water-taxi or a gondola. There are two acceptable subsets: (Plane $\wedge$ Water-taxi) and (Plane $\wedge$ Gondola). Cohesors induce a *choice* operator in the calculus. For example, writing $p()$ for the plane, $w()$ for the water-taxi and $g()$ for the gondola,

$$\begin{array}{rcl} \overline{p}\,.P \mid \overline{t}\,.Q \mid p()_\wedge w().T + p()_\wedge g().G & \to & P \mid Q \mid T. \\ \overline{p}\,.P \mid \overline{g}\,.Q \mid p()_\wedge w().T + p()_\wedge g().G & \to & P \mid Q \mid G. \end{array}$$

Definition 4.1 *The general rendezvous calculus has terms P as for the pi calculus (Definition 2.1) but with inputs $u(\widetilde{x}).P$ replaced by*

$$P ::= \ldots \quad \mid \quad J_1 + \ldots + J_m.$$

Again we identify terms up to $\equiv$. The reaction relation $\to$ is the smallest

satisfying the following and closed under contexts:

$$\begin{aligned}\overline{u}_1 x_1.P_1 \mid \ldots \mid \overline{u}_n x_n.P_n \mid J_1 + \ldots + J_m \mid R \;\; \rightarrow \\ P_1 \mid \ldots \mid P_n \mid Q_1\{x_1/y_1\} \mid \ldots \mid Q_n\{x_n/y_n\} \mid R \\ \text{if some } J_j = u_1(y_1).Q_1 \wedge \ldots \wedge u_n(y_n).Q_n.\end{aligned}$$

Observations are as for the pi calculus but with

$$J_1 + \ldots + J_m \downarrow u \text{ if some } J_i \text{ contains } u(\widetilde{x}).P.$$

Any protocol for atomic or cohesive commitment must be bisimilar to a simple rendezvous in this calculus.

Join patterns, and summation of join patterns, were invented in the join calculus [10] of Fournet and Gonthier. But actually, there is an essential difference between the join calculus and our work. In the join calculus, senders have no continuations (so there is just $\overline{p}$ instead of $\overline{p}.P$). Moreover, all receivers for a given name must be at the same physical location. These two constraints mean that the rendezvous no longer needs any protocol; this allows a very lightweight implementation of rendezvous. By contrast, our project is to use rendezvous as a high-level model of commitment protocols.

5 To Express 2PCP

We illustrate our general rendezvous calculus by revisiting an earlier correctness result due to Berger and Honda [1]. The intention of this section is to show a practical example of our rendezvous calculus, by applying it to an existing problem. This section therefore demands some familiarity with the cited work. However, this section is not needed for the rest of the paper.

Berger and Honda implemented a two-phase commit protocol over their formal model of lossy network (based on a modified asynchronous pi calculus). Writing $2PCP$ for their protocol, they proved that $2PCP \mathrel{\dot{\approx}} !\overline{abort} \oplus !\overline{commit}$ in all contexts: to an outside observer, the protocol either commits or aborts. Here the notation $P \oplus Q$ is non-deterministic choice; it is shorthand for $\nu c.(\overline{c} \mid c().P \mid c().Q)$. Note: this section does not stand along; it assumes the reader is familiar with [1].

We now reformulate their correctness result. They considered a setting of n parties, where each party non-deterministically choose to be able or unable. Write AC for the high-level representation of this:

$$AC \;\; = \;\; u_1() \wedge \ldots \wedge u_n().K_C \mid \overline{u}_1.K_1 \oplus !\overline{una} \mid \ldots \mid \overline{u}_n.K_n \oplus !\overline{una}.$$

Write $[\![AC]\!]$ for Berger and Honda's implementation of AC into their formal network model and protocol. Then the required correctness result is basically $AC \mathrel{\dot{\approx}} [\![AC]\!]$.

Even if all parties were able to react (through the right non-deterministic choices), it is still possible that an instance of the protocol might lead to failure (through message-loss). Applying Lemma 5.3 of [1] to Diagram 3 above, we let each party attempt to recover from this kind of failure by retrying.

There is one subtlety. Two phase commit is a non-compositional protocol: it has a *fixed topology*, in the sense that all the parties involved in the protocol have already fixed to use each other as partners in their interaction, and no one else. In the term AC, their fixed topology was identified by the names $\widetilde{u}$. We reflect the assumption of fixed topology by asserting that $\widetilde{u} \cap \mathrm{fn}\, \widetilde{K} = \emptyset$, and proving $\nu\widetilde{u}.AC \mathrel{\dot{\approx}} [\![\nu\widetilde{u}.AC]\!]$.

The proof is largely similar to that given by Berger and Honda. So as not to duplicate too much of their work, we fix upon a simpler setting. Following Lemma 5.2 of [1] we replace all timeouts with input-guarded choices: $\mathrm{timer}(u.P, Q) = \nu t.(\overline{t} \mid u.P + t.Q)$. We ignore site failure (ie. crash/recovery). Let the coordinator itself not vote. Given these simplifications, Berger and Honda's protocol is as follows. This is written in the low-level network calculus of [1]; we do not repeat its definition here. In the following, we use $[\![\overline{u}_i.K_i]\!] = P_i$ and $[\![u_1()_\wedge \ldots {}_\wedge u_n().K_C]\!] = C$, and we write P_N for $\prod_{n\in N} P_n$.

$$
\begin{array}{rcl}
[\![AC]\!] & = & \nu\widetilde{d}\widetilde{e}.\big([C]_{\widetilde{u}\widetilde{e}} \mid [P_1 \oplus !\overline{una}]_{d_1} \mid \ldots \mid [P_n \oplus !\overline{una}]_{d_n} \big) \\
P_i & = & \overline{u}_i \mid P_i' \\
P_i' & = & \mathrm{timer}(d_i[K_i, (d_i e_i)P_i],\ \overline{e}_i | P_i') \\
 & & \\
C & = & \nu\widetilde{c}a.\big(C_N^{wait} \mid C^{true} \mid C^{false} \big) \\
C_i^{wait} & = & \mathrm{timer}(u_i.\overline{c}_i,\ \overline{a}\,) \\
C^{true} & = & c_1(). \ \ldots \ .c_n().(S_N^{true} \mid K_C) \\
C^{false} & = & a().\nu\widetilde{d'}\widetilde{e'}.(\ S_N^{false} \mid C\{d'e'/de\}\ \big) \\
S_i^{true} & = & \overline{d}_i\mathsf{left} \mid e_i.S_i^{true} \\
S_i^{false} & = & \overline{d}_i\mathsf{right}\langle d_i' e_i' \rangle \mid e_i().S_i^{false}
\end{array}
$$

Theorem 5.1 $\nu\widetilde{u}.AC \mathrel{\dot{\approx}} [\![\nu\widetilde{u}.AC]\!]$.

Proof. Start with smaller lemmas, using the congruence of [1] Section 4.1. (Write P^n for n parallel copies of P.) The first lemma allows for a simplification when success is inevitable; it corresponds to [1] Lemma 5.3. For any n and m, we have

$$\nu de.\big([\overline{d}\,\mathsf{left}^m \mid e.S^{true}] \mid [\overline{e}^n \mid \mathrm{timer}(d[K, (de)P],\ \overline{e}|P')]_d \big) \approx K. \qquad (4)$$

Proof: Denote the left hand side by (m, n). Construct $\mathcal{S} = \{((m, n), K)\}$ for all m, n. The left hand side admits these internal transitions: either a timeout yielding $(m, n{+}1)$; or an interaction on e yielding $(m{+}1, n{-}1)$; or a message-loss on e giving $(m, n{-}1)$, or a message-loss on d giving $(m{-}1, n)$, or an interaction on d yielding $K \mid \nu de.(\overline{d}\mathsf{left}^{m-1} \mid e.S^{true} \mid \overline{e}^n)$. It admits no

external transitions except for those in K. Hence it is $\approx K$.

For the next lemma, recall that the process proceeds in rounds. This lemma allows the simplification that, when a next round has started, then the previous round can be discarded.

$$\nu de.\nu d'e'.\big(\,\overline{d}\mathsf{right}\langle d'e'\rangle^m \mid e.S^{false} \mid C\{^{d'e'}/_{de}\} \mid \overline{e}^n \mid \mathrm{timer}(d[K,(de)P],\ \overline{e}|P')\,\big) \approx \nu de.(C|P). \tag{5}$$

Proof: denote the left hand side by (m,n). This admits the same transitions as for 4, but this time the final interaction on d yields $\nu d'e'.(C\{^{d'e'}/_{de}\}|P\{^{d'e'}/_{de}\} \mid \nu de.(\overline{d}\mathsf{right}\langle d'e'\rangle \mid e.S^{false} \mid \overline{e}^n))$. This is $\approx \nu d'e'.(C\{^{d'e'}/_{de}\}|P\{^{d'e'}/_{de}\} \equiv C|P$.

The next lemma allows a convenient simplification: whether an internal channel c_i has reacted with C^{true} or not, is immaterial. It is like [1] Lemma 5.6.v.

$$\nu c_i.(\overline{c}_i \mid c_i.P) \approx P. \tag{6}$$

Proof: trivial.

We now prove the case where every party has made a non-deterministic choice to be able:

$$\nu\widetilde{de}.\big(\,[C]_{\widetilde{ue}} \mid [P_1]_{d_1} \mid \ldots \mid [P_n]_{d_n}\,\big) \approx K_C|K_1|\ldots|K_n. \tag{7}$$

This case corresponds to [1] Theorem 5.1.ii. After all the non-deterministic choices have been made, we identify three phases:

(i) Vote-gathering. Characterise states in this phase by a set $V \subseteq N$ of parties where the vote $\overline{u}_i : i \in V$ has not yet been sent, and a function $E : N \mapsto \mathrm{int}$ such that there are $E(i)$ copies of $\overline{e}_i$. Also a partition of N into W, C, A such that there is still $C_i^{wait} : i \in W$ and $\overline{c}_i : i \in C$ and an $\overline{a}$ for each element in A. States in this phase are

$$\nu\widetilde{de}\widetilde{ca}.\big(\,[C_W^{wait} \mid C^{true} \mid C^{false} \mid \overline{c}_C \mid \overline{a}^{|A|}]_{\widetilde{ue}} \mid [P_1'']_{d_1} \mid \ldots \mid [P_n'']_{d_n}\,\big).$$

where $P_i'' = \overline{u}_i^n \mid \overline{e}_i^{E(i)} \mid P_i'$ and $n =$ if $i \in V$ then 1 else 0. By Equation 6, any subsequent state where a $\overline{c}_i$ has reacted with C^{true} is bisimilar to a state in this phase.

(ii) All votes in favour. States in this phase are

$$\nu\widetilde{de}.\big(\,[S_N^{true} \mid K_c \mid \nu a.C^{false}]_{\widetilde{ue}} \mid [P_1'']_{d_1} \mid \ldots \mid [P_n'']_{d_n}\,\big).$$

where $V = N$ and P'' is as above. By Equation 4, this is $\approx K_C|K_1|\ldots|K_n$.

(iii) Some votes against. States in this phase are

$$\nu\widetilde{de}\widetilde{d'}\widetilde{e'}\widetilde{ca}.\big(\,[C_W^{wait} \mid C^{true} \mid S^{false} \mid C\{^{d'e'}/_{de}\} \mid \overline{c}_C \mid \overline{a}^n]_{\widetilde{ue}} \mid [P_1'']_{d_1} \mid \ldots \mid [P_n'']_{d_n}\,\big).$$

where $W \subseteq N$ and for any n. By Equation 5, all these states are $\approx$ to Equation 7.

To complete the proof of Equation 7, construct $\mathcal{S}=\dot{\approx} \cup \{(Q,\ K_C|K_1|\ldots|K_n)\}$ for all Q in any of the phases above. The only external transitions made by any Q are those from phase 2, matched by some K. As for internal transitions, the three phases are closed (up to $\dot{\approx}$) under internal transitions.

The unable case (like 7 above, but where at least one of the parties made a non-deterministic choice for $!\overline{una}$) is similar; it corresponds to [1] Theorem 5.1.i.

The result follows from combining the able and unable cases.

6 Conclusions

We remark on compositionality. Traditional analysis of atomic commit protocols (eg. [1,5]) is not compositional: the analysis starts with a fixed set of parties and no other parties are considered.

Our calculus from Section 4 is partly compositional in the sense that a term $u()_\wedge v().P$ may be placed ('composed') in any context $\_ \mid \overline{u}.Q \mid \overline{u}.R \mid \overline{v}.S$, and go on to rendezvous with whatever it finds. In effect, a program chooses to make a transaction involving one u service and one v service, but it does not specify which *particular* providers of the services.

It would also be interesting to consider a *fully compositional* rendezvous, where the full set of participants in the protocol are not known by the initiator. This corresponds to the web-service situation of nested transactions: eg. I make a transaction with the ticket agent, and as part of this the ticket agent makes a sub-transaction with the airline and its bank. We conjecture that such full compositionality might be achieved through allowing joins also of *outputs*, rather than just the joins of inputs that we find in join patterns:

$$agent()_\wedge gondola().P \mid \overline{gondola}.Q \mid \overline{agent}_\wedge airline()_\wedge bank().R \mid \overline{airline}.S \mid \overline{bank}.T.$$

This term should admit a rendezvous of all parties simultaneously, giving $P|Q|R|S|T$. We leave this for further work. (We remark that summation of input and output together has traditionally been considered unimplementable [14] on the grounds that certain rendezvous cannot be settled in bounded time. It seems important to find how this unimplementability applies to nested transactions.)

Further work

We are interested in applying the techniques of this paper to compensations. Compensations are used for long-running transactions, where it is inappropriate to keep all parties locked while the atomic-commit protocol proceeds. The idea is that a party can optimistically assume that the transaction succeeded, but then roll-back ('compensate') if this assumption proves false. A program with compensations will typically yield intermediate states in which some parties disagree about whether a decision should be taken; but any observer who

observes such disagreement will itself be rolled-back, so the disagreements should not count as observable.

We start from the compensational pi calculus of [4]. This defines a compensation-triggering mechanism, and also a sequencing operator (;). However it does not provide primitives with which to program the rolling-back of events. Such programmer roll-backs are the subject of ongoing research. In the meantime, we give a simpler example of atomic commit in the compensational pi calculus – one which leverages both compensation-triggering and sequencing. It uses *contexts* $\mathtt{t}(P, F, B, C)$. This context behaves as process P, but if it reduces to an abort command then the compensation F is triggered; F stands for *failure-handler*. The two terms B and C are used for nesting of transactions, but are not needed here. (Again, we leave definition of the formalism to [4].)

The low-level atomic commit implementation is

$$[\![AC_2]\!] = \mathtt{t}\,\big(\,(\textstyle\prod_{i\in N}(\text{abort} \oplus \text{done}));\overline{x},\ !\overline{una},\ B,\ C\,\big)$$

where $B = C = \text{done}$. In the program, all N parties make a non-deterministic choice to abort or proceed. These decisions are consolidated by the structural rules $\text{abort}|\text{abort} \equiv \text{abort}$ and $\text{done}|P \equiv P$ into just a single $\text{abort};\overline{x}$ or $\text{done};\overline{x}$. From the first,

$$\mathtt{t}(\text{abort};\overline{x},\ !\overline{una}, \text{done}, \text{done}) \to\ !\overline{una}.$$

From the second,

$$\mathtt{t}(\text{done};\overline{x}, !\overline{una}, B, C) \to\dot{\approx}\ \overline{x}.$$

Hence a match with a high-level semantics: let $AC_2 = \nu\widetilde{u}.(u_1()_\wedge \ldots {}_\wedge u_n().\overline{x} \mid \overline{u}_1 \oplus !\overline{una} \mid \ldots \mid \overline{u}_n \oplus !\overline{una})$. Then again

$$AC_2 \,\dot{\approx}\, [\![AC_2]\!].$$

References

[1] Berger, M. and K. Honda, *The two-phase commitment protocol in an extended pi-calculus*, in: *EXPRESS '00*, Electronic Notes in Theoretical Computer Science **39** (2000).
URL ftp://ftp.dcs.qmw.ac.uk/lfp/martinb/express00.ps.gz

[2] Bernstein, P. A., V. Hadzilacos and N. Goodman, "Concurrency Control and Recovery in Database Systems," Addison-Wesley, 1987.
URL http://research.microsoft.com/pubs/ccontrol/

[3] Black, A., V. Cremet, R. Guerraoui and M. Odersky, *An equational theory for transactions*, in: G. Goose, J. Hartmanis and J. van Leeuwen, editors, *FSTTCS 2003*, Lecture Notes in Computer Science **2914** (2003), pp. 38–49.
URL http://www.cse.ogi.edu/~black/publications/fsttcs221.pdf

[4] Bocchi, L., C. Laneve and G. Zavattaro, *A calculus for long running transactions*, in: E. Najm, U. Nestmann and P. Stevens, editors, *FMOODS 2003*, Lecture Notes in Computer Science **2884** (2003), pp. 124–138.
URL http://www.cs.unibo.it/~laneve/papers/biztalk.pdf

[5] Bocchi, L., *Compositional nested long running transactions*, in: *FASE 2004*, 2004, to appear.
URL http://www.cs.unibo.it/~bocchi/pubs.html

[6] Bruni, R., C. Laneve and U. Montanari, *Orchestrating transactions in join calculus*, in: L. Brim, P. Jančar, M. Křetinský and A. Kučera, editors, *CONCUR 2002*, Lecture Notes in Computer Science **2421** (2002), pp. 321–337.
URL http://www.cs.unibo.it/~laneve/papers/zsimpl.ps

[7] Business Process Management Initiative, *Business process modelling notation (BPML)*, website.
URL http://www.bpmi.org/

[8] Cabrera, F., G. Copeland, B. Cox, T. Freund, J. Klein, T. Storey and S. Thatte, *Web services transactions*, website.
URL http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/

[9] Dalal, S., S. Temel, M. Little, M. Potts and J. Webber, *Coordinating business transactions on the web*, IEEE Internet Computing **7** (2003), pp. 30–39.
URL http://csdl.computer.org/comp/mags/ic/2003/01/w1030abs.htm

[10] Fournet, C. and G. Gonthier, *The reflexive chemical abstract machine and the join-calculus*, in: *Proceedings of POPL '96*, ACM (1996), pp. 372–385.
URL http://research.microsoft.com/~fournet/papers/reflexive-cham-join-calculus.ps

[11] Microsoft, *Biztalk server*, website.
URL http://www.microsoft.com/biztalk/

[12] Milner, R., "Communicating and mobile systems: the Pi-calculus," Cambridge University Press, 1999.

[13] Nestmann, U., R. Fuzzati and M. Merro, *Modeling consensus in process calculus*, in: R. Amadio and D. Lugiez, editors, *CONCUR 2003*, Lecture Notes in Computer Science **2761** (2003), pp. 400–414.
URL
http://lamp.epfl.ch/~uwe/doc/nestmann.fuzzati.merro-concur03.pdf

[14] Palamidessi, C., *Comparing the expressive power of the synchronous and the asynchronous pi-calculus*, in: *POPL'97*, ACM SIGPLAN/SIGACT (1997), pp. 256–265.
URL http://www.cse.psu.edu/~catuscia/papers/pi_calc/popl.ps