Compositional Nested Long Running Transactions

Laura Bocchi

bocchi@cs.unibo.it
Dipartimento di Scienze dell'Informazione, University of Bologna
Mura Anteo Zamboni 7, 40127 Bologna, Italy
Fax:+39 051 2094510

Abstract. Web Services offer a widespread standard for making services available on the Internet. Of particular interest is the possibility of composing existing distributed services to create new complex ones. Existing research has already studied long running transactions within a formal context. In this other research, compensations are just partly compositional: a transaction's failure triggers the compensation of immediately enclosed transactions, but not those of nested transactions. In this paper we formally model a more compositional protocol with the asynchronous pi calculus. The resulting behavior is similar to that of the Business Transaction Protocol of OASIS [1], which also has arbitrary nesting.

1 Introduction

Web Services offer a widespread standard for making services available on the Internet, not just to humans but also to other services. Of particular interest is the possibility of composing existing distributed services, perhaps from different companies, to create a new complex service. In this sense each service is a peer that can behave both as a client and as a service provider. Current work, under the general heading 'Choreography', attempts to standardise this possibility of nested composition. Some examples of proposed standards are BPML by bpmi.org [2], XLANG by Microsoft [3], WSFL by IBM [4] and BPEL4WS by a consortium [5]. The W3C Choreography Group is currently working on the Recommendation for Web Services Choreography (a draft has been public since August 2003 [6]).

Within this context, where the parts are loosely coupled and not always trusted, standard ACID transactions (with properties of Atomicity, Consistency, Isolation, Durability) are too strict. This is especially a problem in business transactions: for instance, some 'pay-employee' transaction must be executed promptly at the start of the month, even if other necessary sub-transactions have not yet finished. The payment might subsequently be undone, 'compensated', if it turned out premature. The cycle of perform-then-maybe-compensate is one characteristic of loosely-coupled long-running transactions. Long running transactions have been introduced in 'data processing applications' [7, 8], where they

were called *Sagas*. Then Web Services led to a renewed interest in long running transactions that are supported, in a mainly local perspective, by already mentioned languages (WSFL, XLANG and BPEL). Other contributions arise in the context of Web transaction protocols, where loosely coupled Web services are coordinated as autonomous entities by means of a defined set of transaction messages. We mention the W3C Tentative Hold Protocol (THP) [9], OASIS BTP [1] and WS-Transactions [10] by BEA, IBM and Microsoft.

There is general agreement on the importance of such weaker transactions, but not yet an agreement on their exact meaning. In this paper we choose one particular form of weak transactions, express it in the pi calculus, and prove formally its correctness. In particular, correctness refers to deadlock absence (*Eventuality*), *Durability* and partial *Atomicity*. There is no global Atomicity in the whole set of transactions, but if a transaction fails then all its sub-transaction fail. We will discuss more later.

1.1 Related Works

Long running transactions have been described within several formal contexts. As regards XLANG (used in the product Microsoft BizTalk), its transactional behavior is informally described in [11], and then implemented in the Join calculus. In [12] the transactional behavior is formally specified at high level, and then implemented in the asynchronous pi calculus. In these other works, compensations are just partly compositional: a transaction's failure triggers the compensation of the enclosed transactions, but not those of nested transactions. It is possible to encode the effect of nested triggers, basically though copying a child's compensation code into the parent. But this is no longer compositional, and is clearly inappropriate in a Web Service context where nested transactions might belong to different (untrusted) companies. The need for compositional nested transactions is stressed in the current W3C draft of standards for Choreography and Coordination [6, 10]. The transactions that we encode are compositional: a failure is able to trigger all the compensations of all its nested transactions.

The issue of the paper is describing Web transaction protocols, which are based upon message exchange in a distributed setting. We do this with the pi calculus – it is a message-based formalism, and seems natural for representing distributed protocols in the sense that it is easy to obtain a straightforward implementation. An alternative would be to use formalisms that express properties as predicates between states, such as TLA [14] or ACTA [13] (a first-order logic-based formalism for describing transactional models). These may possibly lead to a more elegant representation than using the pi calculus, but would probably no longer be as close to an implementation.

1.2 Structure of the Paper

Section 2 provides a minimal background on the pi calculus and introduces basics on the described transaction behavior. Section 3 presents an implementation of transaction managers with the asynchronous pi calculus. Section 4 formally

defines some properties, i.e. Durability, Eventuality and Local Atomicity that are then proved for the given implementation. Section 5 contains some conclusive remarks.

2 Preliminaries

We will specify the BTP in the pi calculus. What follows is a brief introduction on the pi calculus, for a full reference see [15].

The asynchronous pi calculus assumes distributed entities called *processes* which exchange messages over channels, named $u, v, \ldots, z$. The content of a message is also a channel name. A process can send a message z along a channel u with the non-blocking output action $\overline{u}\, z$. A process can also receive a message on channel u with the blocking input action $u(v).P$. The parallel execution of two processes P and Q can be expressed as $P \mid Q$. Parallel processes can communicate by performing an input and an output action on the same channel; for example the process $\overline{u}\, z \mid u(v).P$ will perform an input and an output along channel u. Communication is described by the reaction $\overline{u}\, z \mid u(v).P \xrightarrow{\tau} P\{^{z}/_{v}\}$. Its effects are visible to the receiver as name substitution of the actual parameter z for the formal parameter v. The continuation P of the input process can be executed after the input on u has been received. In the polyadic pi calculus a message is a string of names $\tilde{v}$ instead of a single name v. The process $\nu u.P$ declare a local variable u with scope P. It is also possible to define a process that replicates itself: $!P$ is able to create an arbitrary number of copies of P. The pi calculus is summarized in Table 1: *labelled transitions* define the possible reactions of a process, contexts C are processes with holes filled by other processes, and represent environments. *Simulation* is a relation characterizing when two processes have the same behavior.

The general behavior of the protocol we propose is similar to that of the Business Transaction Protocol of OASIS. The expressed relation between transaction and the arbitrary nesting is also present in *Business Activities* (BA) of WS-Transactions.

A *two phase commit protocol* is used first to assemble the 'votes' of nested transactions (ie. whether or not they succeeded), and second to inform them all of the consensus decision. Additional features are provided for controlling which compensations are to be executed. We illustrate the features in Fig.1. where a holiday booking might succeed even if some sub-transactions (e.g. car rental) have failed; where, moreover, some sub-transactions (e.g. an Alitalia flight) might be cancelled even though the overall booking succeeds. The terminology used in [1] is that a *cohesor* needs only some of its children to succeed, while an *atom* requires them all to succeed. Cohesors are modelled here as entities able to flexibly specify the relation with their children. Atoms are a particular case of cohesor. We in fact consider a partition of all nested transactions into two groups: we require success from all of one *necessary* group, and we do not care about success of the other group. A transaction will report success only if all of its necessary children have succeeded. If a transaction fails then its sub transac-

Table 1. The asynchronous pi calculus

Terms P and contexts C in the asynchronous pi calculus are as follows. In $u(\widetilde{x})$ the names $\widetilde{x}$ are bound, as is x in $\nu x.P$. We identify terms up to alpha-renaming of bound names.

$$\begin{array}{rcl} P & ::= & \mathsf{0} \mid \overline{u}\,\widetilde{x} \mid u(\widetilde{x}).P \mid P|P \mid \nu x.P \mid !P \\ C & ::= & \_ \mid u(\widetilde{x}).C \mid P|C \mid C|P \mid \nu x.C \mid !C \end{array}$$

Labelled transitions are as follows, where labels μ range over $u(\widetilde{x})$, $\nu\tilde{z}.\overline{u}\,\tilde{x}$ and τ.

$$\overline{u}\,\widetilde{x} \xrightarrow{\overline{u}\,\widetilde{x}} \mathsf{0} \quad (\textsc{out}) \qquad u(\widetilde{x}).P \xrightarrow{u(\widetilde{x})} P \quad (\textsc{in}) \qquad \frac{P \mid !P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'} \quad (\textsc{rep})$$

$$\frac{P \xrightarrow{\mu} P' \quad x \notin \mu}{\nu x.P \xrightarrow{\mu} \nu x.P'} \quad (\textsc{res}) \qquad \frac{P \xrightarrow{\nu\tilde{z}.\overline{u}\,\tilde{y}} P' \quad x \neq u, x \in \tilde{y} \backslash \tilde{z}}{\nu x.P \xrightarrow{\nu\tilde{z}x.\overline{u}\,\tilde{y}} P'} \quad (\textsc{open})$$

$$\frac{P \xrightarrow{\mu} P' \quad \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\mu} P' \mid Q} \quad (\textsc{par}) \qquad \frac{P \xrightarrow{\nu\tilde{z}.\overline{u}\,\tilde{y}} P' \quad Q \xrightarrow{u(\tilde{x})} Q' \quad \tilde{z} \cap \mathrm{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\tau} \nu\tilde{z}.(P' \mid Q'\{\tilde{y}/\tilde{x}\})} \quad (\textsc{com})$$

Simulation is as follows. We write $\stackrel{\tau}{\Rightarrow}$ for $\xrightarrow{\tau}{}^*$, and $\stackrel{\mu}{\Rightarrow}$ for $\xrightarrow{\tau}{}^* \xrightarrow{\mu} \xrightarrow{\tau}{}^*$ when $\mu \neq \tau$, and $P \stackrel{\mu}{\Longrightarrow}$ for $\exists P' : P \stackrel{\mu}{\Longrightarrow} P'$. A symmetric relation $\mathcal{S}$ is a *weak ground simulation* if whenever $P\mathcal{S}Q$ then

- $P \xrightarrow{\mu} P'$ implies there exists Q' such that $Q \stackrel{\mu}{\Rightarrow} Q'$ and $P'\mathcal{S}Q'$.

Write $\lesssim$ for the largest ground simulation. S is a *weak ground bisimulation*, if both S and S^{-1} are weak ground simulations. Write $\approx$ for the largest ground bisimulation. We note some standard results:

$$P \approx Q \quad \text{implies} \quad \forall C : C[P] \approx C[Q] \qquad \nu x.x().P \approx \mathsf{0}$$
$$P|\mathsf{0} \approx P \qquad P|Q \approx Q|P \qquad P|(Q|R) \approx (P|Q)|R \qquad !P \approx P|!P$$
$$\nu x.\nu y.P \approx \nu y.\nu x.P \qquad \nu x.(P|Q) \approx P|\nu x.Q \text{ if } x \notin \mathrm{fn}(P)$$
$$\nu x.P \approx \nu x'.P\{^{x'}/_{x}\} \text{ if } x' \notin \mathrm{fn}(P)$$

Notation We write $\widetilde{x}_{\mathcal{C}}$ for an arbitrary sequence $x_1, \ldots, x_n$ of the elements in set $\mathcal{C}$. We also use these syntactic sugars:

$$\begin{array}{rcll} x.P & = & x().P & \text{(empty input)} \\ \widetilde{x}.P & = & x_1.\ldots.x_n.P & \text{(sequence input)} \\ \nu\widetilde{x}.P & = & \nu x_1.\ldots.\nu x_n.P & \text{(sequence restriction)} \\ P \oplus Q & = & \nu c.(\overline{c}\,|c.P|c.Q),\ c \text{ fresh} & \text{(nondeterministic choice)} \\ x[P,Q] & = & \nu u, v.(\overline{x}\,u, v|u.P|v.Q),\ u, v \text{ fresh} & \text{(selection)} \\ \overline{x}\,\mathsf{left} & = & x(u,v).\overline{u} & \\ \overline{x}\,\mathsf{right} & = & x(u,v).\overline{v} & \end{array}$$

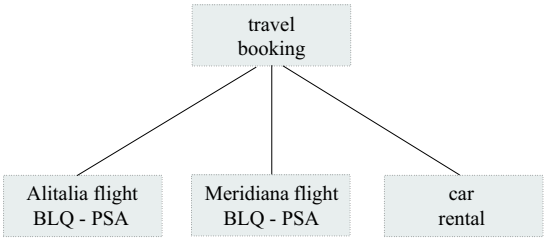


Fig. 1. A prototypical set of nested transactions. Each box represents a transactional web service, provided by different companies. There are several possible modes of failure propagation: *up-propagation*, where if Alitalia and Meridiana fail then we abort the car and the overall booking fails; *non-propagation*, where even if the car fails we can still proceed with the others; *down-specific-propagation*, where if one of Alitalia or Meridiana succeed then the other should be aborted; *down-propagation*, where the booking (the current job) is told to abort by some higher-up agent (not pictured) and so must abort all its children; *spontaneous-failure*, where the booking itself might decide to fail and so must abort its children.

tions fail (partial atomicity). If a transaction succeeds we also consider a second partition of the nested transactions: we will accept the success of the first group, and will abort the others. For instance, Fig.1. uses the following subsets:

If all in this subset succeeded...	*then accept these...*	*and undo these*
{Alitalia}	{Alitalia,car}	{Meridiana}
{Meridiana}	{Meridiana, car}	{Alitalia}

To simplify matters, the work in this paper considers only a single row of the table (ie. one necessary/unnecessary partition and one accept/reject partition), rather than multiple rows in each protocol specification. To handle multiple rows, something like Join patterns [16] might be used.

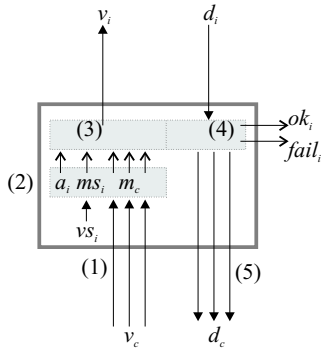
3 Design of Transaction Managers

In this section we implement (nested) transactions and their compensation-triggering. We implement them in the asynchronous pi calculus, using a generalization of the two phase commit implementation given by Berger and Honda [17].

We assume a set $\mathcal{I}$ of transactions ranged over by i. The tree-like hierarchy of these transactions is denoted by a relation $\text{par} : \mathcal{I} \mapsto \mathcal{I}$ which indicates the immediate parent of a transaction; writing $\text{par}^n(i)$ for n applications of the pair function, we assume that if $i = \text{par}^m(j)$ then do not exists n such that $j = \text{par}^n(i)$. Define the set of i's children $C(i) = \{j : \text{par}(j) = i\}$. As discussed in the introduction, we consider only a single 'necessary' partition of $C(i)$ into $N(i), U(i)$ – with the meaning that success of all $N(i)$ is necessary for i to succeed, while $U(i)$ are unnecessary. We therefore consider just a single

consequent partition of $C(i)$ into $A(i), R(i)$ – where all of $A(i)$ are accepted, while all of $R(i)$ are rejected (undone).

We now describe the operation of each transaction block. We illustrate with transaction i, which has children $\widetilde{c}$.



(1) This transaction i itself makes a non-deterministic 'self' vote vs_i. Also, all of the children c make their votes v_c. All these votes are made using left/right notation (Table 1). **(2)** Each vote is transformed into an 'internal message' m. The purpose of this translation is to separate necessary child votes $N(i)$ from unnecessary votes $U(i)$. Each internal message m_c means that the child either voted success (left), or it was unnecessary. But if a child should vote failure (right) and was necessary, it will make an 'abort' signal a_i instead. **(3)** If all internal messages ms_i/m_c arrive, then the transaction i as a whole can succeed, and so indicates success (left) to its parent over the channel v_i. But if even one abort message a_i was received, then the transaction as a whole fails, and so it indicates failure (right). **(4)** Eventually the parent p will know whether to accept i, or to abort/undo it. This decision is communicated to i via the 'decision' channel d_i, and so determines i's final state. The transaction i can indicate its final state via the messages $ok_i/abort_i$. **(5)** Finally, the decision is propagated down to all the children c. The accepted children, those in $A(i)$, will be told the same decision as i received. The rejected children, those in $R(i)$, will be told to abort/undo regardless. The code T_i for transaction i plus all its descendants, is as follows.

$$T_i \quad = \quad \nu a_i, ms_i, vs_i, \widetilde{m}_{C(i)}, \widetilde{v}_{C(i)}, \widetilde{d}_{C(i)}. \qquad \text{(transaction)}$$
$$(T_i.\text{sv} \mid T_i.\text{m} \mid T_i.\text{col} \mid \prod_{c \in C(i)} T_c)$$

$$T_i.\text{sv} \quad = \quad \overline{vs}_i left \oplus \overline{vs}_i right \qquad \text{(self-vote)}$$

$$T_i.\mathrm{m} \quad = \quad vs_i[\overline{ms}_i, \overline{a}_i] \mid \prod_{c \in N(i)} v_c[\overline{m}_c, \overline{a}_i] \mid \prod_{c \in U(i)} v_c[\overline{m}_c, \overline{m}_c] \qquad \text{(internals)}$$

$$\begin{aligned} T_i.\mathrm{col} \quad &= \quad a_i.(\overline{v}_i right \mid T_i.\mathrm{fail}) \qquad \text{(collate votes)} \\ &\quad \mid \widetilde{m}_{C(i)}.ms_i.(\overline{v}_i left \mid d_i[T_i.\mathrm{ok}, T_i.\mathrm{fail}]) \end{aligned}$$

$$T_i.\mathrm{ok} \quad = \quad \overline{ok}_i \mid \prod_{c \in A(i)} \overline{d}_c left \mid \prod_{c \in R(i)} \overline{d}_c right \qquad \text{(ok)}$$

$$T_i.\mathrm{fail} \quad = \quad \overline{abort}_i \mid \prod_{c \in C(i)} \overline{d}_c right \qquad \text{(fail)}$$

To explain the code, the *transaction* T_i consists of three parts: T_i.sv generates its self-vote, T_i.m fulfills step (2) by converting votes into internal messages, and T_i.col collates votes and receives the final decision, for steps (3–5). We have also included all the children transactions T_c, since they refer to the local channels $\widetilde{v}_{C(i)}$ and $\widetilde{d}_{C(i)}$.

The *self-vote* T_i.sv makes a non-deterministic choice (using $\oplus$) to become, at runtime, a vote for success or failure.

The *internals* T_i.m convert all the votes into internal messages according to whether the vote came from a needed component $N(i)$ or an unnecessary one $U(i)$. We count the self-vote as necessary. An internal message m_c is generated if the child c voted success, or if the child c was unnecessary. An internal abort message a_i is generated otherwise (i.e. a necessary child voted for failure).

The *collator* T_i.col will either receive all the internal messages m_c/ms_i, or will receive at least one internal abort a_i. The abort signifies that a necessary part failed. If this happens, then the component i signals a failure to its parent on channel v_i, and proceeds with T_i.fail to tell abort its children. But if all internal messages were received, then it tells its parent about its success, and awaits the parent's final verdict.

The *ok/fail* processes T_i.ok and T_i.fail indicate the final state of this transaction, using the global channels $\overline{ok}_i$ and $\overline{abort}_i$. In the case of OK, the accepted children $A(i)$ are told of the positive verdict, while the rejected children $R(i)$ are told to fail. In the case of FAIL, all children are told to fail.

Let us recall the five modes of propagation identified in Fig.1. and explain how they are reflected in the code. Let us denote the Alitalia transaction with i_a, Meridiana with i_m, car rental with i_c and travel booking with i. As an example we consider the following row:

If all in this subset succeeded...	*then accept these...*	*and undo these*
{Alitalia}	{Alitalia,car}	{Meridiana}

Up-propagation is achieved by enclosing i_a in the set $N(i)$ so that if T_i.m receives the failure vote $\overline{v}_{i_a}$ it eventually fails. It fails by sending a message $\overline{a}_i$ that unblocks the abort branch of T_i.col. *Non-proragation* is achieved by enclosing i_c in the set $U(i)$ so that T_i.m reacts to both success and failure messages $\overline{m}_{i_c}$. *Down-specific proragation* is achieved by enclosing i_m in the set $R(i)$ so that T_i.ok communicates, in any case, a failure decision to i_m. *Down proragation* is implemented by the message $\overline{d}_i$ that in case of local success of i notifies the upper outcome.

Finally, we collect the overall tree of transactions in a test harness H. We suppose the root of the tree is transaction i:

$$H = \nu v_i, d_i.\left(T_i \mid v_i[\overline{d}_i left, \overline{d}_i right]\right).$$

This harness merely executes the root transaction T_i, waits for its overall vote v_i, and immediately sends back the vote as the decision if vote was success. If vote was fail there is no need of decision communication: the child already had its outcome without waiting any signal.

The following lemma describes the observable behavior a of generic transactions. T_i takes a local decision on the basis of the votes of its children (if there are any) and of its non deterministic self-vote. In any case T_i communicates its vote to the parent. If it locally failed it terminates with failure soon. If it did not locally fail it waits for the global decision of the parent and its final outcome depends from it.

Lemma 1. *If* $C(i) = \emptyset$*, then* $T_i \approx (\overline{v}_i left \mid d_i[\overline{ok}_i, \overline{abort}_i]) \oplus (\overline{v}_i right \mid \overline{abort}_i)$.

Proof sketch. When $C(i) = \emptyset$, then also $N(i) = U(i) = A(i) = R(i) = \emptyset$. Hence T_i simplifies to just

$$\begin{aligned} T_i = \nu a_i, ms_i, vs_i.(& (\overline{vs_i} left \oplus \overline{vs_i} right o) \mid vs_i[\overline{ms_i}, \overline{a_i}] \\ & \mid a_i.(\overline{v_i} right \mid \overline{abort}_i) \mid ms_i.(\overline{v_i} left \mid d_i[\overline{ok_i}, \overline{abort}_i])\,). \end{aligned}$$

Observe that the only action T_i can make is a τ move, choosing whether vs_i votes left or right. This is reflected by the right hand side.

4 Transaction Properties

In this section we prove some properties of the protocol: *Durability*, *Eventuality* and *Local Atomicity*. Durability means that each node reaches no more than one outcome and, in general, that the only observable behavior of the protocol is the set of outcome notifications. Eventuality implies the absence of deadlock in the protocol: an outcome is achieved in every node of the tree. Finally we consider Local Atomicity. Normally, atomicity is the property that either every transaction succeeds or every transaction fails. We have seen that this is too strict for business transactions. Instead, local atomicity is just the property that if one transaction fails, then all its children fail. Let us start by defining a transaction's descendants set.

Definition 2 (Descendants). *Define* $D(i) = \{j : \exists n.i = \mathrm{par}^n(j)\}$.

The precise pattern of the 'mountains' (Fig.2.) is determined by the compile-time choice of which failures propagate, ie. by the partitions of $D(i)$, $N(i)/U(i)$ and $A(i)/R(i)$, and also by the run-time non-deterministic self-vote made by each transaction. We start with the proposition that, after the transaction has finished executing, it ends up in a state where every node has made a single choice (either $\overline{ok}_i$ or $\overline{abort}_i$), such that the set of all nodes respects local atomicity.

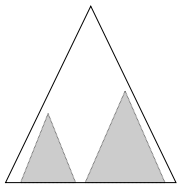


Fig. 2. Local atomicity may be pictured as 'mountains', where the shaded mountains represents those nodes, in the transactions tree, that have failed.

Before starting the lemmas we remark upon conventions. Recall the syntactic sugar for selection (Table 1):

$$\begin{array}{rcll} x[P,Q] & = & \nu u,v.(\overline{x}\,u,v \mid u.P \mid v.Q), \text{with } u,v \text{ fresh} & \text{(selection)} \\ \overline{x}\,\mathsf{left} & = & x(u,v).\overline{u} & \\ \overline{x}\,\mathsf{right} & = & x(u,v).\overline{v} & \end{array}$$

We will use shorthand labels $x(left), x(right), \overline{x}\,left, \overline{x}\,right$, with the following transitions:

$$x[P,Q] \overset{x(left)}{\Longrightarrow} P \qquad x[P,Q] \overset{x(right)}{\Longrightarrow} Q$$

$$\overline{x}\,left \overset{\overline{x}\,left}{\Longrightarrow} \mathsf{0} \qquad \overline{x}\,right \overset{\overline{x}\,right}{\Longrightarrow} \mathsf{0}$$

$$\frac{P \overset{\overline{x}\,left}{\Longrightarrow} P' \quad Q \overset{x(left)}{\Longrightarrow} Q'}{P \mid Q \overset{\tau}{\Longrightarrow} P' \mid Q'} \qquad \frac{P \overset{\overline{x}\,right}{\Longrightarrow} P' \quad Q \overset{x(right)}{\Longrightarrow} Q'}{P \mid Q \overset{\tau}{\Longrightarrow} P' \mid Q'}$$

and also equivalent versions of (RES) and (PAR). These rules are satisfactory abstractions of the actual selection transitions, so long as the process in question only ever uses selection channels appropriately (e.g. there is no $x[P,Q] \mid x(u,v).(\overline{u} \mid \overline{v})$). In some cases we want to refer to generic actions, i.e. votes and decisions, indifferently from their specific types: $\overline{v}_i$ stands for a generic vote from i ($\overline{v}_i left$ or $\overline{v}_i right$ arbitrarily), $\overline{d}_i$ stands for an arbitrary decision ($\overline{d}_i left$ or $\overline{d}_i right$). We will also refer to generic outcome notifications ($\overline{abort}_i$ or $\overline{ok}_i$) with $\overline{outcome}_i$.

Durability

We now prove durability: the observable behavior is never anything other than a single outcome notification ($\overline{ok}_i/\overline{abort}_i$) for each node. The property is proved in Theorem 5; we present some auxiliary lemmas first.

Lemma 3.

1. $\nu a_i, ms_i, \widetilde{m}_{C(i)}.(T_i.m \mid a_i.P \mid \widetilde{m}_{C(i)}.ms_i.Q) \lesssim vs_i[0,0] \mid P \oplus Q \mid \prod_{c\in C(i)} v_c[0,0]$.
2. $P \oplus d_i[Q,P] \lesssim P \oplus (d_i[0,0] \mid P \oplus Q)$.
3. $(P_1 \mid P_2) \oplus (Q_1 \mid Q_2) \lesssim (P_1 \oplus Q_1) \mid (P_2 \oplus Q_2)$.
4. $P_1 \oplus P_2 \approx P_1 \oplus (P_2 \oplus P_1)$.

Proposition 4. $\nu v_i, d_i.T_i \lesssim \prod_{j\in D(i)}(\overline{abort}_j \oplus \overline{ok}_j)$.

Proof. By induction on the depth of the tree.
Base Case. $C(i) = \emptyset$. By Lemma 1, $T_i \approx (\overline{v}_i left | d_i[\overline{ok}_i, \overline{abort}_i]) \oplus (\overline{v}_i right | \overline{abort}_i)$. Applying now Lemma 3 (case 2, 1 and 3) to the right hand term

$$T_i \lesssim (\overline{ok}_i \oplus \overline{abort}_i) \mid (\overline{v}_i left \oplus \overline{v}_i left) \mid d_i[0,0].$$

Trivially $\nu v_i, d_i.T_i \lesssim (\overline{abort}_i \oplus \overline{ok}_i)$.
Inductive Case. let us consider a generic node T_i. We have for inductive hypothesis that $\forall c \in C(i), \nu v_c, d_c.T_c \lesssim \prod_{j\in D(c)}(\overline{abort}_j \oplus \overline{ok}_j)$. It is straightforward that

$$\nu \widetilde{v}_{C(i)}, \widetilde{d}_{C(i)}.\prod_{c\in C(i)} T_c \lesssim A \text{ where } A = \prod_{c\in C(i)}\prod_{j\in D(c)} (\overline{abort}_j \oplus \overline{ok}_j).$$

By the standard results properties of $\approx$ (Table 1) we have the following:

$$\begin{aligned}
T_i \approx\ & \nu vs_i, \widetilde{v}_{C(i)}, \widetilde{d}_{C(i)}.(\prod_{c\in C(i)} T_c \mid T_i.\mathrm{sv} \mid \nu a_i, ms_i, \widetilde{m}_{C(i)}.(T_i.\mathrm{m} \mid T_i.\mathrm{col})) \\
\lesssim\ & \nu vs_i, \widetilde{v}_{C(i)}, \widetilde{d}_{C(i)}.(A \mid T_i.\mathrm{sv} \mid vs_i[0,0] \mid \prod_{c\in C(i)} v_c[0,0] \mid P \oplus Q) && \text{(Lemma 3.1)} \\
\lesssim\ & \nu \widetilde{d}_{C(i)}.((P \oplus Q) \mid \nu \widetilde{v}_{C(i)}.(\prod_{c\in C(i)} v_c[0,0]) \mid \nu vs_i.(vs_i[0,0] \mid T_i.\mathrm{sv}) \mid A. && \text{(structural)}
\end{aligned}$$

Trivially $\nu vs_i.(vs_i[0,0] \mid T_i.\mathrm{sv}) \approx 0$ and $\nu \widetilde{v}_{C(i)}.(\prod_{c\in C(i)} v_c[0,0]) \approx 0$ so

$$T_i \lesssim \nu \widetilde{d}_{C(i)}.((P \oplus Q) \mid A)$$

where $P = \overline{v}_i right \mid T_i.\mathrm{fail}$ and $Q = \overline{v}_i left | d_i[T_i.\mathrm{ok}, T_i.\mathrm{fail}]$. By Lemma 3.2,

$$\begin{aligned}
T_i \lesssim\ & \nu \widetilde{d}_{C(i)}.((T_i.\mathrm{fail} \oplus d_i[T_i.\mathrm{ok}, T_i.\mathrm{fail}]) \mid (\overline{v}_i right \oplus \overline{v}_i left) \mid A) \\
\approx\ & \nu \widetilde{d}_{C(i)}.((T_i.\mathrm{fail} \oplus (d_i[0,0] \mid (T_i.\mathrm{ok} \oplus T_i.\mathrm{fail}))) \mid (\overline{v}_i right \oplus \overline{v}_i left) \mid A) && \text{(Lemma 3.1)} \\
\approx\ & \nu \widetilde{d}_{C(i)}.((\overline{abort}_i \oplus (\overline{ok}_i \oplus \overline{abort}_i)) \mid R \mid d_i[0,0] \mid (\overline{v}_i right \oplus \overline{v}_i left) \mid A) && \text{(Lemma 3.2)}
\end{aligned}$$

where R is obtained extracting the decision propagation from T_i.ok and T_i.fail

$$R = \prod_{c \in C(i)} \overline{d}_c right \oplus ((\prod_{c \in A(i)} \overline{d}_c left \mid \prod_{c \in R(i)} \overline{d}_c right) \oplus \prod_{c \in C(i)} \overline{d}_c right).$$

$$\approx \quad (\overline{abort}_i \oplus (\overline{ok}_i \oplus \overline{abort}_i)) \mid \widetilde{d}_{C(i)}.R \mid d_i[0,0] \mid (\overline{v}_i right \oplus \overline{v}_i left) \mid A \qquad \text{(structural)}$$

$$\approx \quad (\overline{abort}_i \oplus (\overline{ok}_i \oplus \overline{abort}_i)) \mid d_i[0,0] \mid (\overline{v}_i right \oplus \overline{v}_i left) \mid A \qquad \text{(structural)}$$

$$\lesssim \quad (\overline{abort}_i \oplus \overline{ok}_i) \mid d_i[0,0] \mid (\overline{v}_i right \oplus \overline{v}_i left) \mid A. \qquad \text{(Lemma 3.3)}$$

Note that $\nu v_i.(\overline{v}_i right \oplus \overline{v}_i left) \approx 0$ and $\nu d_i.d_i[0,0] \approx 0$, so (structural)

$$\nu v_i, d_i T_i \lesssim \prod_{j \in D(i)} (\overline{abort}_j \oplus \overline{ok}_j).$$

Corollary 5 (Durability). $H \lesssim \prod_{i \in \mathcal{I}} (\overline{abort}_j \oplus \overline{ok}_j)$.

Eventuality

We prove that any node of the tree can always notify an outcome (none of the nodes deadlocks). Lemma 6 proves that each transaction can eventually vote for each possible computation. Lemma 7 provides that, depending on the vote of a node, we can always get a decision from the parent that unblocks one of the final processes T_i.ok or T_i.fail. Then we prove (Lemma 8) that if a node votes and its provided with a decision then it and all the subtree is able to have an outcome. This leads directly to Corollary 9 that deals with the observable behavior of H. The following lemma is that a transaction can always eventually vote, no matter what sequence of internal moves it has already made.

Lemma 6. *If $T_i \stackrel{\tau}{\Longrightarrow} T_i'$ then $T_i' \stackrel{\overline{v}_i left}{\Longrightarrow}$ or $T_i' \stackrel{\overline{v}_i right}{\Longrightarrow}$.*

Henceforth we use the shorthand $\tilde{z} = a_i, ms_i, vs_i, \widetilde{m}_{C(i)}, \widetilde{v}_{C(i)}, \widetilde{d}_{C(i)}$ to refer to the scope of a node T_i.

Lemma 7.

1. *If $T_i \stackrel{\overline{v}_i left}{\Longrightarrow} T_i'$ then $T_i' \stackrel{d_i(left)}{\Longrightarrow} \nu\tilde{z}.(T_i.ok \mid P)$ for some P and $T_i' \stackrel{d_i right}{\Longrightarrow} \nu\tilde{z}.(T_i.fail \mid Q)$ for some Q,*
2. *If $T_i \stackrel{\overline{v}_i left}{\Longrightarrow} T_i'$ then $T_i' \stackrel{\tau}{\Longrightarrow} \nu\tilde{z}.(T_i.fail \mid P)$ for some P.*

Proof sketch. 1. If T_i was able to perform a $\overline{v}_i left$ transition it has previously unblocked the successful branch of T_i.col that is $\overline{v}_i left \mid d_i[T_i.\text{ok}, T_i.\text{fail}]$. After the $\overline{v}_i left$ transition it will become

$$\nu\tilde{z}.(a_i.(\overline{v}_i right \mid T_i.\text{fail}) \mid d_i[T_i.\text{ok}, T_i.\text{fail}] \mid \prod_{c \in C(i)} T_c').$$

The only possible transitions are the following:

$$\xrightarrow{d_i(left)} \nu\tilde{z}.(a_i.(\overline{v}_i right \mid T_i.\text{fail}) \mid T_i.\text{ok} \mid \prod_{c\in C(i)} T'_c), \text{ or}$$

$$\xrightarrow{d_i(right)} \nu\tilde{z}.(a_i.(\overline{v}_i right \mid T_i.\text{fail}) \mid T_i.\text{fail} \mid \prod_{c\in C(i)} T'_c).$$

2. If T_i is able to perform a $\overline{v}_i right$ action it has already triggered the failing action of T_i.com. After sending the output $\overline{v}_i right$ the process of the node i is as follows:

$$\nu\tilde{z}.(T_i.\text{fail} \mid \widetilde{m}_C.ms_i.(\overline{v}_i left \mid d_i[T_i.\text{ok}, T_i.\text{fail}]) \mid \prod_{c\in C(i)} T'_c).$$

Note: by hypothesis, only τ moves have been performed hence T_i.fail has not reacted.

Lemma 8. *If $T'_i : T_i \overset{\overline{v}_i left}{\Longrightarrow} T'_i$ or $T_i \overset{\overline{v}_i right}{\Longrightarrow} T'_i$ then for every $j \in \{i\} \cup D(i)$, we have $T'_i \mid \overline{d}_i \overset{\overline{ok}_j}{\Longrightarrow}$ or $T'_i \mid \overline{d}_i \overset{\overline{abort}_j}{\Longrightarrow}$.*

Proof. Let us reason by induction on the depth of the level of i.
Base Case. $C(i) = \emptyset$. By Lemma 6, $T_i \xrightarrow{\overline{v}_i} T'_i$. Thus by Lemma 7 $T'_i \mid \overline{d}_i \overset{\overline{outcome}_i}{\Longrightarrow}$.
Inductive Case. By Lemma 6 we have $T_i \xrightarrow{\overline{v}_i} T'_i$. By Lemma 7 we have $T'_i \mid \overline{d}_i \overset{\tau}{\Longrightarrow} \nu\tilde{z}.(T_i.\text{ok} \mid P)$ for some P or $T'_i \mid \overline{d}_i \overset{\tau}{\Longrightarrow} \nu\tilde{z}.(T_i.\text{fail} \mid P)$ for some P. Recall the definition of T_i.ok and $T_i.abort$:

$$T_i.\text{ok} \quad = \quad \overline{ok}_i \mid \prod_{c\in A(i)} \overline{d}_c left \mid \prod_{c\in R(i)} \overline{d}_c right$$

$$T_i.\text{fail} \quad = \quad \overline{abort}_i \mid \prod_{c\in C(i)} \overline{d}_c right$$

In both cases it is possible, from $\nu\tilde{z}.(T_i.\text{ok} \mid P)$, to perform the following actions:

- $\xrightarrow{\overline{d}_c}$ such that by inductive hypothesis $\forall j \in D(c) \cup \{c\}$, $T'_c \mid \overline{d}_c \overset{\overline{outcome}_j}{\Longrightarrow}$,
- $\xrightarrow{\overline{outcome}_i}$.

It holds so that for any T'_i such that $T_i \overset{\overline{v}_i}{\Longrightarrow} T'_i$ then for every $j \in \{i\} \cup D(i)$, $T'_i \mid \overline{d}_i \overset{\overline{outcome}_j}{\Longrightarrow}$.

Corollary 9 (Eventuality). *For every H' such that $H \overset{\tau}{\Longrightarrow} H'$ then, for every $j \in \mathcal{I}$, where $H' \overset{\overline{ok}_i}{\Longrightarrow}$ or $H' \overset{\overline{abort}_i}{\Longrightarrow}$.*

Local Atomicity

To prove Local Atomicity we simplify (Lemma 10) the behavior of T_i by considering its state after the it voted. Recall that by Lemma 6, each node eventually votes. Then we show (Lemma 11) that if a node i receives a failure decision or votes failure itself then none of the nodes in the subtree of i will ever notify a successful outcome. Finally we show that any node abort just if it received a failure notification or votes failure itself and generalize the property to the whole protocol H.

Lemma 10.

1. *If* $T_i \stackrel{\overline{v}_i left}{\Longrightarrow} T_i'$ *then* $T_i' \approx \nu\widetilde{d}_{C(i)}.(d_i[T_i.ok, T_i.fail] \mid \prod_{c\in C(i)} T_c')$ *with* $T_c \stackrel{\overline{v}_c}{\Longrightarrow} T_c'$.
2. *If* $T_i \stackrel{\overline{v}_i right}{\Longrightarrow} T_i'$ *then* $T_i' \approx \nu\widetilde{d}_{C(i)}.(T_i.fail \mid \prod_{c\in C(i)} T_c')$ *with* $T_c \stackrel{\overline{v}_c}{\Longrightarrow} T_c'$ *or* $T_c \Rightarrow T_c'$.

Lemma 11.

1. *If* $T_i \stackrel{\overline{v}_i left}{\Longrightarrow} T_i'$ *then* $\nexists j \in \{i\} \cup D(i)$ *such that* $T_i' \mid \overline{d}_i right \stackrel{\overline{ok}_j}{\Longrightarrow}$,
2. *if* $T_i \stackrel{\overline{v}_i right}{\Longrightarrow} T_i'$ *then* $\nexists j \in \{i\} \cup D(i)$ *such that* $T_i' \stackrel{\overline{ok}_j}{\Longrightarrow}$.

Proof. For induction on the depth of the tree.
Base Case. the tree is composed by node T_i with $C(i) = \emptyset$. From Lemma 1 $T_i \approx Ts_i$ with $Ts_i = (\overline{v}_i left | d_i[\overline{ok}_i, \overline{abort}_i]) \oplus (\overline{v}_i right | \overline{abort}_i)$. The only step that Ts_i can perform is a τ action corresponding to the choice of one of the two branches.

1. If the left branch is chose then the only possible sequence of steps is: $Ts_i \stackrel{\tau}{\longrightarrow} \overline{v}_i left | d_i[\overline{ok}_i, \overline{abort}_i] \stackrel{\overline{v}_i left}{\Longrightarrow} d_i[\overline{ok}_i, \overline{abort}_i]$. Hence $d_i[\overline{ok}_i, \overline{abort}_i] \mid \overline{d}_i right \stackrel{\tau}{\Longrightarrow} \overline{abort}_i \stackrel{\overline{abort}_i}{\Longrightarrow}$.
2. If the right branch is chose then the only possible sequence of steps is: $Ts_i \stackrel{\tau}{\longrightarrow} \overline{v}_i right | \overline{abort}_i \stackrel{\overline{v}_i right}{\Longrightarrow} \overline{abort}_i \stackrel{\overline{abort}_i}{\Longrightarrow}$.

Inductive Case. we have that $T_i \stackrel{\overline{v}_i}{\Longrightarrow} T_i'$. Depending on the vote type we have two cases:

1. If $T_i \stackrel{\overline{v}_i left}{\Longrightarrow} T_i'$ then by Lemma 10 $T_i' \approx \nu\widetilde{d}_{C(i)}.(d_i[T_i.\text{ok}, T_i.\text{fail}] \mid \prod_{c\in C(i)} T_c')$. Hence
$$\nu d_i.(T_i' \mid \overline{d}_i right) \approx T_i.\text{fail} \mid \prod_{c\in C(i)} T_c').$$
Here i will surely fail (and just fail for durability (Theorem 5)) and will also provide a $\overline{d}_c right$ decision $\forall c \in C(i)$ that for inductive hypothesis grant that $\nexists j \in \{c\} \cup D(c)$ such that $T_c' \mid \overline{d}_i right \stackrel{\overline{ok}_j}{\Longrightarrow}$.
2. If $T_i \stackrel{\overline{v}_i right}{\Longrightarrow} T_i'$ then for Lemma 10 $T_i' \approx \nu\widetilde{d}_{C(i)}.(T_i.\text{fail} \mid \prod_{c\in C(i)})$. The case is analogue to the previous one here.

Theorem 12 (Local Atomicity). *If* $H \stackrel{\overline{abort}_i}{\Longrightarrow} H'$ *then* $\nexists j \in D(i)$ *such that* $H' \stackrel{\overline{ok}_j}{\Longrightarrow}$.

Proof. If $H \stackrel{\overline{abort}_i}{\Longrightarrow} H'$ then i must have failed for one of the following reasons:

- i voted $\bar{v}_i right$, the result follows from Lemma 11.2,
- i voted $\bar{v}_i left$ and received a failure decision from the parent $\bar{d}_i right$. The result follows from Lemma 11.1.

5 Conclusions

We discussed a possible behavior for long running transactions in a context of hierarchical relations with other transactions, represented by an arbitrarily deep tree. The exercise had the aim to clarify two principal aspects.

The first is the role of *cohesors* and *atoms* in the protocol, their behavior and their relation. We proposed a flexible approach for describing the relation between votes of a sub-transaction and parent outcome type, and again between parent outcome and children outcome. Atoms can be modelled here as particular cases of cohesors. This flexible behavior is present in also in BTP and WS-Transactions. The paper provides an implementation with the pi calculus.

The second aspect discussed in this paper is the mechanism of compensation triggering. Compensations are a straightforward addition to the current work: each failure notification $\overline{abort}_i$ is associated to the execution of the compensation of transaction i. Transactions are thought as independent entities, maybe from different companies, connected by a superior-inferior (caller-provider) links. Those links create a hierarchical structure of arbitrary depth. Our mechanism coordinates the triggering of compensations: when a node i fails the protocol creates the global compensation process by composing the local compensations of all the nodes in the subtree of i.

We proved that each transaction has no more than one outcome (Durability) and that the protocol does not deadlock (Eventuality). We also proved that if one node fails then its entire subtree will also fail (Local Atomicity).

Other aspects have yet to be considered. It would be desirable to allow an explicit representation of the choice of sets $N(i), A(i), U(i), R(i)$ at run time, according to the computation feedback. This aspect is indirectly managed (it could be simulated with Join patterns). Other aspects are the introduction of concepts like localities and unreliability in communication between remote transactions. Managing these would probably lead to the introduction of timers in order to avoid deadlock pathologies.

Acknowledgement. The technical part of this paper owes much to discussions with Lucian Wischik, and the general ideas on transaction behavior also owe much to Greg Meredith. I thank Paolo Ciancarini, Cosimo Laneve, Gianluigi Zavattaro and the anonymous referees for their useful comments.

References

1. S. Dalal, S. Temel, M. Little, M. Potts and J. Webber. Coordinating Business Transactions on the Web. IEEE Internet Computing, January-February 2003.
2. J.J. Dubray. A novel approach for modeling business process definitions. [http://www.ebpml.org/ebpml2.2.doc].
3. S. Thatte. XLANG: Web Services for Business Process Design. [http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm], Microsoft Corporation, 2001.
4. F. Leymann. Web Services Flow Language (WSFL 1.0). [http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf], Member IBM Academy of Technology, IBM Software Group, 2001.
5. F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte and S. Weerawarana. Business Process Execution Language for Web Services (BPEL4WS 1.0). [http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/], 2002.
6. W3C Choreography Group. Web Services Choreography Requirements 1.0. [http://www.w3.org/TR/ws-chor-reqs/], 2003.
7. H. Garcia-Molina and K. Salem. Sagas. In Proc. of SIGMOND International Conference on Management of Data, pp. 249-259, 1987.
8. H. Garcia-Molina, G. Gawlick, J. Klein, K. Kleissner and K. Salem. Salem. Modelling Long Running Activities as Nested Sagas. IEEE Bulletin of Technical Committee on Data Engeneering, 14(1), 1991.
9. J. Roberts and K. Srinivasan. Tentative Hold Protocol Part 1: White paper. W3C Note 28 November 2001. [http://www.w3.org/TR/tenthold-1/]
10. F. Cabrera, G. Copeland, B. Cox, T. Freund, J. Klein, T. Storey and S. Thatte. Web Services Transaction (WS-Transaction). [http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/], 2002.
11. R. Bruni, C. Laneve and U. Montanari. Orchestrating Transactions in Join Calculus. In CONCUR'02, LNCS 2421, pp. 321-337.
12. L. Bocchi, C. Laneve and G. Zavattaro. A Calculus for Long Running Transactions. In FMOODS'03, LNCS 2884, pp. 124-138.
13. P. K. Chrysantys and K. Ramamritham. ACTA: a framework for specifying and reasoning about transaction structure and behavior. In SIGMOND International Conference on Management of Data, pp. 194-203, 1990.
14. L. Lamport. The Temporal Logic of Actions. ACM TOPLAS, 16(3), pp. 872-923, 1994.
15. R. Milner. Communicating and Mobile Systems: the Pi-calculus. Cambridge University Press, 1999.
16. F. Le Fessant and L. Maranget. Compiling Join-Patterns. Electronic Notes in Theoretical Computer Science 16(3), 2000.
17. M. Berger and K. Honda. The Two-Phase Commitment Protocol in an Extended Pi-calculus. Electronic Notes in Theoretical Computer Science 39(1), 2003.

Appendix: Proofs

Note: the present appendix is intended only for completeness of the submission; it is not intended to be part of the final paper.

As a shorthand let us denote with $C \subseteq C(i)$ the set of votes that have not been received from the children; similarly we define $N \subseteq N(i), U \subseteq U(i)$.

Proof of Lemma 3.1

$\nu a_i, ms_i, \widetilde{m}_{C(i)}.(T_i.\mathrm{m} \mid a_i.P \mid \widetilde{m}_{C(i)}.ms_i.Q) \quad \lesssim \quad vs_i[0,0] \mid P \oplus Q \mid \prod_{c \in C(i)} v_c[0,0].$

Proof sketch. Let $A = \nu a_i, ms_i, \widetilde{m}_{C(i)}.(T_i.\mathrm{m} \mid a_i.P \mid \widetilde{m}_{C(i)}.ms_i.Q)$ and $B = vs_i[0,0] \mid P \oplus Q \mid \prod_{c \in C(i)} v_c[0,0]$. Recall that

$$T_i.\mathrm{m} = vs_i[\overline{ms}_i, \overline{a}_i] \mid \prod_{c \in N(i)} v_c[\overline{m}_c, \overline{a}_i] \mid \prod_{c \in U(i)} v_c[\overline{m}_c, \overline{m}_c]$$

We consider informally all possible transitions made by A:

1. A transition $\xrightarrow{vs_i(left)}$ derived from $T_i.\mathrm{m} \xrightarrow{vs_i(left)} \overline{ms}_i \mid \prod_{c \in N(i)} v_c[\overline{m}_c, \overline{a}_i] \mid \prod_{c \in U(i)} v_c[\overline{m}_c, \overline{m}_c]$, or similarly $\xrightarrow{vs_i(right)}$ from $T_i.\mathrm{m} \xrightarrow{vs_i(right)} \overline{a}_i \mid \prod_{c \in N(i)} v_c[\overline{m}_c, \overline{a}_i] \mid \prod_{c \in U(i)} v_c[\overline{m}_c, \overline{m}_c]$. B can also make the same transition.
2. A transition $\xrightarrow{v_c(right)}$ derived from
   - $T_i.\mathrm{m} \xrightarrow{v_c(right)} vs_i[\overline{ms}_i, \overline{a}_i] \mid \prod_{j \in N(i) \setminus c} v_j[\overline{m}_j, \overline{a}_i] \mid \overline{a}_i \mid \prod_{c \in U(i)} v_c[\overline{m}_c, \overline{m}_c]$ if $c \in N(i)$.
   - $T_i.\mathrm{m} \xrightarrow{v_c(right)} vs_i[\overline{ms}_i, \overline{a}_i] \mid \prod_{c \in N(i)} v_c[\overline{m}_c, \overline{a}_i] \mid \prod_{j \in U(i) \setminus c} v_j[\overline{m}_j, \overline{m}_j] \mid \overline{m}_j$ if $c \in U(i)$.

   or similarly a transition $\xrightarrow{v_c(left)}$ that add a $\overline{m}_c$ term to the process in both cases ($c \in N(i)$ or $c \in U(i)$). In these cases B is able to perform the same transition: $B \xrightarrow{v_c(right)} \prod_{j \in C(i) \setminus c} v_j[0,0] \mid \ldots$ or $B \xrightarrow{v_c(left)} \prod_{j \in C(i) \setminus c} v_j[0,0] \mid \ldots$.
3. Further repetitions of steps 1 and 2, producing more $\bar{a}_i$, more $\bar{m}_c$ (but at most one for each transaction $c \in C(i)$), and at most one self vote $\bar{m}s_i$. B is always able to behave in the same way since it has an input process for all the possible votes.
4. Subsequently A can also make τ steps deriving from the reaction of a vote $\overline{m}_c$ with the branch $\widetilde{m}_{C(i)}.ms_i.Q$; B can match this without making any transitions. In every instant, given the set C of votes that A has not collected yet, B is in the form $B = vs_i[0,0] \mid P \oplus Q \mid \prod_{c \in C} v_c[0,0]$ if the self vote has not been collected or $B = P \oplus Q \mid \prod_{c \in C} v_c[0,0]$ otherwise.
5. Finally A can make one of the following transitions:
   - if an aborting vote has been received we have a process $\overline{a}_i$ in parallel with the failure branch $a_i.P$. With a τ action the process P can be unblocked. Also B can make a τ action deriving from the nondeterministic choice $P \oplus Q \xrightarrow{\tau} P$. Notice that if A unblocked P is because it received an output and, since we have exactly one per child then Q will never be

unblocked. Notice also that now, both A and B can continue to make analogues inputs on channels $v_c \in C$ and vs_i.

– If a successful vote has been received and caught from each c in $C(i)$ and from vs_i; then A can unblock Q. In this case B can make a τ step and become Q. We have then A reduced to $\nu a_i, ms_i, \widetilde{m}_{C(i)}.(a_i.P \mid Q)$ and trivially $A \lesssim B$.

Proof of Lemma 6
If $T_i \stackrel{\tau}{\Longrightarrow} T_i'$ then $T_i' \stackrel{\overline{v}_i left}{\Longrightarrow}$ or $T_i' \stackrel{\overline{v}_i right}{\Longrightarrow}$.

Proof sketch. By induction on the depth of the tree. For the *Base Case* we have $C(i) = \emptyset$. It follows that $T_i \approx (\overline{v}_i left | d_i[\overline{ok}_i, \overline{abort}_i]) \oplus (\overline{v}_i right | \overline{abort}_i)$ (Lemma 1). Let $B = (\overline{v}_i left | d_i[\overline{ok}_i, \overline{abort}_i]) \oplus (\overline{v}_i right | \overline{abort}_i)$. If $T_i \stackrel{\tau}{\Longrightarrow} T_i'$ then also $B \stackrel{\tau}{\Longrightarrow} B'$ for some B', and $T_i' \approx B'$. Clearly, for any such B', then $B' \stackrel{\overline{v}_i}{\Longrightarrow}$. Hence also $T_i \stackrel{\overline{v}_i}{\Longrightarrow}$.

For the *Inductive Case* we assume that $T_j \stackrel{\tau}{\Longrightarrow} T_j'$ implies $T_j' \stackrel{\overline{v}_j}{\Longrightarrow}$ for all $j \in C(i)$; we will prove that $T_i \stackrel{\tau}{\Longrightarrow} T_i'$ implies $T_i' \stackrel{\overline{v}_i}{\Longrightarrow}$. We also define the process $T_i'.col$ as achieved from T_i.col with zero or more transitions $\stackrel{m_c}{\longrightarrow}$ with $c \in C$:

$$T_i'.col = a_i.(\overline{v}_i right \mid T_i.\text{fail}) \mid \widetilde{m}_C.ms_i.(\overline{v}_i left | d_i[T_i.\text{ok}, T_i.\text{fail}])$$

T_i can do one of the following τ-steps:

1. a self vote corresponding to the choice $\overline{vs}_i left \oplus \overline{vs}_i right \rightarrow \overline{vs}_i left$ ($\overline{vs}_i left \oplus \overline{vs}_i right \rightarrow \overline{vs}_i right$),
2. an internal move by a descendant ($d \in D(i)$),
3. receiving a vote from a child $c \in U$ corresponding to

$$\prod_{j \in U} v_j[\overline{m}_j, \overline{m}_j] \stackrel{\overline{v}_c}{\longrightarrow} \overline{m}_c \mid \prod_{j \in U \setminus c} v_j[\overline{m}_j, \overline{m}_j]$$

4. receiving a vote from a child $c \in N$ corresponding to one of the following

$$\prod_{j \in N} v_c[\overline{m}_j, \overline{a}_i] \stackrel{\overline{v}_c left}{\longrightarrow} \overline{m}_c \mid \prod_{c \in N \setminus c} v_j[\overline{m}_j, \overline{a}_i]$$

$$\prod_{j \in N} v_c[\overline{m}_j, \overline{a}_i] \stackrel{\overline{v}_c right}{\longrightarrow} \overline{a}_i \mid \prod_{j \in N \setminus c} v_j[\overline{m}_j, \overline{a}_i]$$

5. receiving the self vote: $vs_c[\overline{ms}_i, \overline{a}_i] \stackrel{\overline{vs}_i left}{\longrightarrow} \overline{ms}_i$ for success and $vs_c[\overline{ms}_i, \overline{a}_i] \stackrel{\overline{vs}_i lright}{\longrightarrow} \overline{a}_i$ for failure.
6. catching a successful vote from $c \in C$ descending from an action $T_i'.col \stackrel{m_c}{\Longrightarrow}$:

$$m_c \mid T_i'.col \stackrel{\tau}{\longrightarrow} a_i.(\overline{v}_i right \mid T_i.\text{fail}) \mid \widetilde{m}_{C \setminus \{c\}}.ms_i.(\overline{v}_i left | d_i[T_i.\text{ok}, T_i.\text{fail}])$$

In all the cases, eventually we have one of the following:

– At least one failure vote has been received from a process in $N(i)$:

$$\overline{a}_i \mid T_i'.col \xrightarrow{\tau} \overline{v}_i right \mid T_i.\text{fail} \mid \widetilde{m}_{C'(i)}.ms_i.(\overline{v}_i left | d_i[T_i.\text{ok}, T_i.\text{fail}]).$$

Hence the action $\xrightarrow{\overline{v}_i right}$ is possible, satisfying the lemma.

– No failure votes have been received. In this case, by inductive hypothesis, $\forall j \in N(i)$ we have $T_j \overset{\overline{v}_i left}{\Longrightarrow} \forall j \in N(j)$, $T_j \overset{\overline{v}_i left}{\Longrightarrow}$ or $T_j \overset{\overline{v}_i right}{\Longrightarrow} \forall j \in U(j)$. The set of votes led, via reduction 2 and 3, to a set $\prod_{c \in C(i)} \overline{m}_c$ of output messages. Also the vote $\bar{v}s_i left$ led, via reduction 4, to the output $\overline{ms}_i$.

$$\nu a_i, ms_i, \widetilde{m}_{C(i)}.(\prod_{c \in C(i)} \overline{m}_c \mid \overline{ms}_i \mid T_i.\text{col})$$

$$= \nu a_i, ms_i, \widetilde{m}_{C(i)}.(\prod_{c \in C(i)} \overline{m}_c \mid \overline{ms}_i \mid a_i.(\overline{v}_i right \mid T_i.\text{fail}) \mid \widetilde{m}_C.ms_i.(\overline{v}_i left | d_i[T_i.\text{ok}, T_i.\text{fail}]))$$

$$\overset{\tau}{\Longrightarrow} \approx \nu a_i, ms_i.(\overline{ms}_i \mid a_i.(\overline{v}_i right \mid T_i.\text{fail}) \mid ms_i.(\overline{v}_i left | d_i[T_i.\text{ok}, T_i.\text{fail}]))$$

$$\overset{\tau}{\Longrightarrow} \approx \overline{v}_i left \mid d_i[T_i.\text{ok}, T_i.\text{fail}]$$

$$\xrightarrow{\overline{v}_i left}$$

Hence the lemma is satisfied.

Proof of Lemma 10

1. If $T_i \overset{\overline{v}_i left}{\Longrightarrow} T_i'$ then $T_i' \approx \nu \widetilde{d}_{C(i)}.(d_i[T_i.\text{ok}, T_i.\text{fail}] \mid \prod_{c \in C(i)} T_c')$ with $T_c \overset{\overline{v}_c}{\Longrightarrow} T_c'$.
2. If $T_i \overset{\overline{v}_i right}{\Longrightarrow} T_i'$ then $T_i' \approx \nu \widetilde{d}_{C(i)}.(T_i.\text{fail} \mid \prod_{c \in C(i)} T_c')$ with $T_c \overset{\overline{v}_c}{\Longrightarrow} T_c'$ or $T_c \Rightarrow T_c'$.

Proof sketch. If $T_i \overset{\overline{v}_i left}{\Longrightarrow} T_i'$ then T_i has gone through all the steps descending by what follows (not necessarily in given order)

– $T_i.\text{sv} \xrightarrow{\tau} \overline{vs}_i left$ and $T_c \xrightarrow{\overline{v}_c left} T_c' \quad \forall c \in C(i)$: i needs all the success votes for unblocking its own success vote,
– $T_i.\text{m}$ performed a transition $\xrightarrow{\overline{m}_c}$ for each $c \in C(i)$ and the transition $\xrightarrow{\overline{ms}_i}$. Eventually $T_i.\text{m}$ is reduced to $\mathsf{0}$.
– $T_i.\text{col}$ interacted with $T_i.\text{m}$ performing a sequence of $\xrightarrow{m_c}$ actions for each $c \in C(i)$ and a step $\xrightarrow{ms_i}$. The process $T_i.\text{col}$ is eventually reduced to $a_i.(\overline{v}_i right | T_i.\text{fail}) \mid \overline{v}_i left \mid d_i[T_i.\text{ok}, T_i.\text{fail}]$

After all these steps (that represent the only possible set of steps) we have

$$T_i' = \quad \nu\tilde{z}.(a_i.(\overline{v}_i right | T_i.\text{fail}) \mid d_i[T_i.\text{ok}, T_i.\text{fail}] \mid \prod_{c \in C(i)} T_c')$$

$$\approx \quad \nu\widetilde{d}_{C(i)}.(d_i[T_i.\text{ok}, T_i.\text{fail}] \mid \prod_{c \in C(i)} T_c').$$

If $T_i \stackrel{\overline{v}_i right}{\Longrightarrow} T_i'$ then T_i has gone through the following steps descending by what follows (not necessarily in given order)

- at least one between $T_i.\text{sv} \stackrel{\tau}{\longrightarrow} \overline{vs}_i right$ and $T_c \stackrel{\overline{v}_c right}{\longrightarrow} T_c'$ since i needs at least one failure vote to unblock the failure branch $a_i.(\overline{v}_i \mid T_i.\text{fail})$,
- optionally one of the $T_i.\text{sv} \stackrel{\tau}{\longrightarrow} \overline{vs}_i left$ and $T_c \stackrel{\overline{v}_c left}{\longrightarrow}$
- $T_i.\text{m}$ performed at least one step $\stackrel{\overline{a}_i}{\longrightarrow}$ for the (one or more) failure votes received and optionally some steps $\stackrel{\overline{m}_c}{\longrightarrow}$ and the self vote $\stackrel{\overline{ms}_i}{\longrightarrow}$.
- $T_i.\text{col}$ interacted with $T_i.\text{m}$ at least once with a $\stackrel{a_i}{\longrightarrow}$ action that unblocks the failure branch. Optionally is could have received some messages along m_c. $T_i.\text{col}$ is eventually reduced to $\overline{v}_i right \mid T_i.\text{fail} \mid \widetilde{m}_C.ms_i.(\overline{v}_i left | d_i[T_i.\text{ok}, T_i.\text{fail}])$.

After all these steps (that represent the only possible set of steps) we have

$$T_i' = \qquad \nu\tilde{z}.(T_i.\text{fail} \mid \widetilde{m}_C.ms_i.(\overline{v}_i left | d_i[T_i.\text{ok}, T_i.\text{fail}]) \mid G \mid \prod_{c \in C(i)} T_c').$$

where G encloses the possibly remaining parts of $T_i.\text{sv}$ and $T_i.\text{m}$. It is evident that $\nu ms_i, vs_i, \widetilde{m}_{C(i)}, \widetilde{v}_{C(i)}.(\widetilde{m}_C.ms_i.(\overline{v}_i left | d_i[T_i.\text{ok}, T_i.\text{fail}]) \mid G) \approx \mathsf{0}$ since it can just perform τ actions and at least a missing success prevents $\overline{v}_i left | d_i[T_i.\text{ok}, T_i.\text{fail}]$ to be unblocked. It follows:

$$T_i' \approx \qquad \nu\widetilde{d}_{C(i)}.(T_i.\text{fail} \mid \prod_{c \in C(i)} T_c').$$