Design and Verification of Long–Running Transactions in a Timed Framework

Ruggero Lanotte [a], Andrea Maggiolo–Schettini [b], Paolo Milazzo [b], Angelo Troina [c,d,*]

[a] *Dipartimento di Scienze della Cultura, Politiche e dell'Informazione - Università dell'Insubria*
Via Carloni 78, 22100 - Como, Italy.

[b] *Dipartimento di Informatica - Università di Pisa*
Largo Pontecorvo 3, 56127 - Pisa, Italy.

[c] *LIX - École Polytechnique*
Rue de Saclay, 91128 - Palaiseau, France.

[d] *LSV - ENS Cachan*
61 Avenue du Président Wilson, 94235 - Cachan, France.

Abstract

Long–running transactions consist of tasks which may be executed sequentially and in parallel, may contain sub–tasks, and may require to be completed before a deadline. These transactions are not atomic and, in case of executions which cannot be completed, a compensation mechanism must be provided.

In this paper we develop a model of Communicating Hierarchical Timed Automata suitable to describe the mentioned aspects in a framework where also time is taken into account. We develop the patterns for composing long–running transactions sequentially, in parallel or by nesting. The correct compensation of a composed long–running transaction is preserved by these composition patterns.

The automaton-theoretic approach allows the verification of properties by model checking. As a case study, we model and analyse an example of e–commerce application described in terms of long–running transactions.

Key words: Hierarchical Timed Automata; Long–running Transactions; Compensations; Model checking.

* Corresponding author: Phone: +33 (0)1 69 33 89 57 - Fax: +33 (0)1 69 33 30 14.
Email addresses: ruggero.lanotte@uninsubria.it (Ruggero Lanotte), maggiolo@di.unipi.it (Andrea Maggiolo–Schettini), milazzo@di.unipi.it (Paolo Milazzo), troina@lix.polytechnique.fr (Angelo Troina).

1 Introduction

The term *transaction* is commonly used in database systems to denote a logical unit of work designed for short–lived activities, usually lasting under a few seconds. These transactions are performed either completely or not at all: this means that if something goes wrong during the execution of the transaction, a roll–back activity is performed, which re–establishes the state of the system exactly as it was before the beginning of the transaction.

In order to permit the system to perform the roll–back activity, locks are acquired on the necessary resources at the beginning of a transaction and are released only at its end (in both the cases of completion and roll–back). The use of locks, which forbids others to access the resources, is justified by the short duration of the transaction. These transactions are called *ACID transactions*, because they satisfy the properties of Atomicity, Consistency, Isolation and Durability. Recent developments in distributed systems have created the need of a new notion of transaction in which remote entities (possibly of different companies) may interact by performing complex activities (which may require also a human–interaction) that may take minutes, days or weeks. This increased length of time with respect to ACID transactions, forbids the use of locks on resources, and hence makes roll–back activities impossible. In this kind of transactions, the alternative to roll-back activities is the use of compensations, which are activities explicitly programmed to remove the effects of the performed actions, and may require, for instance, the payment of some kind of penalty. This new kind of transactions are usually called *long–running transactions*, but they are also known as sagas [19], web transactions [22], and extended transactions [20]. Although there is an interest for their support in distributed object–based middlewares [20], they are studied in particular in the context of orchestration languages and notations for Web Services (such as BPEL [11], WSCI [30] and BPMN [12]).

Web Services are technologies that allow distribution and interoperability of heterogeneous software components providing services over the Internet. Orchestration languages permit the definition of complex services in terms of interactions among simpler services. Most orchestration languages offer several primitives for composing and handling services. Since the specifications of these languages mainly consist in informal textual description of their constructors, there is a strong interest in the formalisation of their semantics, see [5,8–10,6,7,14,15,26,22,28,29,31]. Among these papers, [10,6,7,14,15,22,28] give theoretical foundations to orchestration languages fragments describing long–running transactions. In particular, [10] identifies three main composition patterns for transactional activities with compensations, namely sequential composition, parallel composition, and nesting, and provides a formal semantics for them.

Communicating Hierarchical Machines (CHMs) [3], which are finite state machines endowed with the ability of refining states and of composing machines in parallel, seem to be a formalism suitable to describe transactional activities and their composition patterns. Actually, they allow describing concurrent and interacting components (such as transactional activities) and facilitate the definition of composition patterns providing hierarchical composition by means of state refinement. Moreover, time is an important factor in the functioning of distributed systems, where communication may take time and deadlines may be used to counteract failure of remote components. Besides, transactions may have deadlines imposed by the requested QoS. Hence, to describe transactions, a formalism is needed that also allows the representation of time constraints.

After the seminal paper by Alur and Dill [2], many models of Timed Automata have been proposed and used to describe systems in which time cannot be abstracted. A model of Hierarchical Timed Automata (HTAs) has been proposed in [16]. An important advantage of automata based formalisms is that they are amenable to formal analysis, such as model checking.

In this paper we define the model of Communicating Hierarchical Transaction–based Timed Automata (CHTTAs). As HTAs, CHTTAs have a notion of explicit time and take from CHMs the ability of composing machines in parallel and hierarchically. However, CHTTAs differ from CHMs and HTAs insofar as they have two different terminal states (to describe the commit or abort of transactions) and provide different communication mechanisms. We give a flattening procedure in order to obtain a timed automaton from a CHTTA, and hence the reachability problem for CHTTAs is decidable and properties of CHTTAs may be verified by model checkers defined for Timed Automata (e.g. Kronos [32] and UPPAAL [4]). For instance, one can verify whether a long–running transaction, or a part of it, terminates correctly or not, by checking the reachability of the commit state. Moreover, since the analysis is performed in a timed framework, one may also study how the reachability of a certain state is affected by the time constraints within the transaction, or check the upper and lower bounds of its duration.

We propose CHTTAs to describe and analyse transactional activities in a timed framework and define operations for composing CHTTAs which correspond to composition patterns of transactional activities. We give formal representations of these patterns in terms of CHTTAs and prove their correctness. While the design of long–running transactions requires a hierarchical description, we may flatten the resulting CHTTA by obtaining a timed automaton on which model checking can be applied. As a case study, we model with CHTTAs a typical long–running transaction and verify some properties with the UPPAAL model checker [4].

The remainder of the paper is organised as follows. In Section 2 we introduce

the syntax and the semantics of CHTTAs. In Section 3 we give a flattening procedure for translating CHTTAs into Timed Automata. As a consequence, we prove the decidability of the reachability problem for CHTTAs. In Section 4 we show how the patterns of sequential and parallel composition and nested transactions can be modelled with CHTTAs; long–running transactions are obtained by combining these patterns. This allows model checking long–running transactions. As a case study, in Section 5, we model a long–running transaction describing a client–server double request and we analyse it with the UPPAAL model checker. Finally, in Section 6 we discuss some related works, and in Section 7 we draw our conclusions.

2 Communicating Hierarchical Transaction–based Timed Automata

Let us assume a finite set of communication channels $\mathcal{C}$ with a subset $C_{Pub} \subseteq \mathcal{C}$ of *public channels*. As usual, we denote with $a!$ the action of sending a signal on channel a and with $a?$ the action of receiving a signal on a.

Let us assume a finite set X of positive real variables called *clocks*. A *valuation* over X is a mapping $v : X \to \mathbb{R}^{\geq 0}$ assigning real values to clocks. Let V_X denote the set of all valuations over X. For a valuation v and a time value $t \in \mathbb{R}^{\geq 0}$, let $v + t$ denote the valuation such that $(v + t)(x) = v(x) + t$, for each clock $x \in X$.

The set of *constraints* over X, denoted $\Phi(X)$, is defined by the following grammar:

$$\phi ::= x \sim c \,|\, \phi \wedge \phi \,|\, \neg\phi \,|\, \phi \vee \phi \,|\, true$$

where ϕ ranges over $\Phi(X)$, $x \in X$, $c \in \mathbb{Q}$ and $\sim \in \{<, \leq, =, \neq, >, \geq\}$.

We write $v \models \phi$ when *valuation v satisfies constraint ϕ*. More formally:

- $v \models x \sim c \quad \Leftrightarrow \quad v(x) \sim c$;
- $v \models \phi_1 \wedge \phi_2 \quad \Leftrightarrow \quad v \models \phi_1$ and $v \models \phi_2$;
- $v \models \neg\phi \quad \Leftrightarrow \quad v \not\models \phi$;
- $v \models \phi_1 \vee \phi_2 \quad \Leftrightarrow \quad v \models \phi_1$ or $v \models \phi_2$;
- $v \models true$.

Let $B \subseteq X$; with $v[B]$ we denote the valuation resulting after resetting all clocks in B. More precisely, $v[B](x) = 0$ if $x \in B$, $v[B](x) = v(x)$, otherwise. Finally, with $\mathbf{0}$ we denote the valuation with all clocks reset to 0, namely $\mathbf{0}(x) = 0$ for all $x \in X$.

Definition 2.1 *A Transaction–based Timed Automaton (TTA) is a tuple* $A = (\Sigma, X, S, Q, q_0, Inv, \delta)$, *where:*

- $\Sigma \subseteq \{a!, a? \mid a \in \mathcal{C}\}$ *is a finite set of labels;*
- X *is a finite set of clocks;*
- S *is a finite set of superstates;*
- $Q = L \cup S \cup \{\odot, \otimes\}$*, where* L *is a finite set of basic states and* $\odot$ *and* $\otimes$ *represent the special states commit and abort, respectively;*
- $q_0 \in L$ *is the initial state;*
- $Inv : L \cup S \to \Phi(X)$ *is the invariant function assigning to each basic state and superstate a formula that must hold in any instant in which the state is enabled;*
- $\delta \subseteq (L \times \Sigma \cup \{\tau\} \times \Phi(X) \times 2^X \times Q) \cup (S \times \{\boxdot, \boxtimes\} \times Q)$ *is the set of transitions, where* $\boxdot$ *and* $\boxtimes$ *are special labels for transitions of commit and abort, respectively.*

Superstates are states that can be refined to automata (*hierarchical composition*). Note that from superstates in S only transitions with labels in $\{\boxdot, \boxtimes\}$ can be taken. We assume that $\odot$ and $\otimes$ are the final states of a TTA.

A TTA is said to be flat when it has no refinable states.

Definition 2.2 (Flat TTAs) *A TTA* $A = (\Sigma, X, S, Q, q_0, \delta)$ *is* flat *if* $S = \emptyset$.

Inspired by the definition of CHMs (see [3]) we now introduce CHTTAs as an extension of TTAs allowing superstate refinement and parallelism.

Definition 2.3 *Let* $\Sigma_{Pub} = \{a!, a? \mid a \in C_{Pub}\}$ *and* $\mathcal{A} = \{A^1, \ldots, A^n\}$ *be a finite set of TTAs, with* $A^i = (\Sigma^i, X^i, S^i, Q^i, q_0^i, Inv^i, \delta^i)$ *and such that there exists* m *(*$m < n$*) such that* A^j *is flat if and only if* $j \geq m$*. A* Communicating Hierarchical Transaction–based Timed Automaton *(*$CHTTA_{\mathcal{A}}^{\Sigma_{Pub}}$*) is given by the following grammar:*

$$CHTTA_{\mathcal{A}}^{\Sigma_{Pub}} \ ::= \ \langle A^i, \mu \rangle \quad \Big| \quad CHTTA_{\mathcal{A}}^{\Sigma_{Pub}} || CHTTA_{\mathcal{A}}^{\Sigma_{Pub}}$$

where μ *is a* hierarchical composition function $\mu : S^i \to CHTTA_{\{A^{i+1}, \ldots, A^n\}}^{\Sigma_{Pub}}$.

Parallelism allows concurrent execution of automata. Hierarchical composition allows refining superstates. Automata executed in parallel may communicate by synchronizing transitions labelled with a sending and a receiving action on the same channel. Communication performed using non public channels are only allowed between components inside the same superstate or at top–level. Communication performed by using public channels have no restrictions.

Note that, by definition of $\mathcal{A}$ and μ, cyclic nesting is avoided. In the following, if it does not give rise to ambiguity, we may write CHTTA instead of $\text{CHTTA}_{\mathcal{A}}^{\Sigma_{Pub}}$. Finally, if A is a flat TTA, in $\langle A, \mu \rangle$ μ is an empty function; in this case, we denote the whole CHTTA just with A.

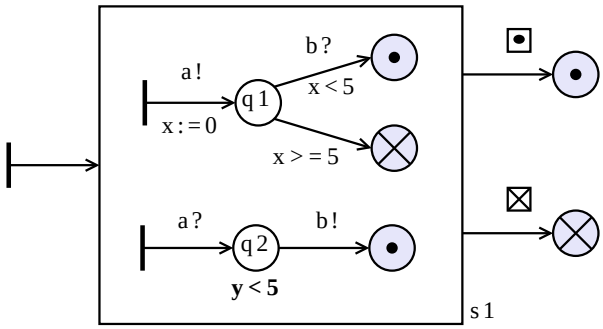

Fig. 1. Example of CHTTA.

Example 2.4 *In Figure 1 we show an example of CHTTA. Superstates of the CHTTA are depicted as boxes and basic states as circles; initial states are represented as vertical segments. Invariants are written in boldface and juxtaposed under the state to which they refer. We omit them when they are equal to the constraint true. Transitions are labelled arrows in which labels* τ *and constraints true are omitted. Containment in boxes represents hierarchical composition, while parallel composition is represented by juxtapositions. With* Inv_{true} *we denote the invariant which assigns true to all states. The CHTTA in the figure is formally defined as*

$$A = \langle (\emptyset, \emptyset, \{s_1\}, \{q_0, s_1, \odot, \otimes\}, q_0, Inv_{true}, \delta), \mu \rangle$$

where:

$$\delta = \{(q_0, \tau, true, \emptyset, s_1), (s_1, \boxdot, \odot), (s_1, \boxtimes, \otimes)\}$$

and $\mu(s_1) = A_1 || A_2$*.* A_1 *and* A_2 *are defined as*

$$A_1 = (\{a!, b?\}, \{x\}, \emptyset, \{q_0, q_1, \odot, \otimes\}, q_0, Inv_{true}, \delta_1)$$

where $\delta_1 = \{(q_0, a!, true, \{x\}, q_1), (q_1, b?, x < 5, \emptyset, \odot), (q_1, \tau, x \geq 5, \emptyset, \otimes)\}$ *and*

$$A_2 = (\{a?, b!\}, \{y\}, \emptyset, \{q_0, q_2, \odot, \otimes\}, q_0, Inv, \delta_2)$$

where $Inv(q_0) = true$ *and* $Inv(q_2) = y < 5$*. The set of transitions is given by* $\delta_2 = \{(q_0, a?, true, \emptyset, q_2), (q_2, b!, true, \emptyset, \odot)\}$*.*

2.1 Semantics of CHTTAs

Configurations of CHTTAs are pairs $tc = (c, \nu)$ where c, the *untimed configuration*, represents the currently active states, and ν, the *composed valuation*, represents the current clock valuations.

The configuration of a CHTTA without parallel components, when the currently active state is a basic state, is a pair (q, v) with q the currently active

state and v the automaton clock valuation. We represent with $q.c$ the configuration where q is a superstate and c is the untimed configuration of $\mu(q)$, and with $v.\nu$ the composed valuation where v is the clock valuation of the automaton having q as superstate and ν is the composed valuation of the clocks of $\mu(q)$. We denote with $c_1; c_2$ the untimed configuration of the parallel composition of two CHTTAs having c_1 and c_2 as untimed configurations. Analogously, we denote with $\nu_1; \nu_2$ the composed valuation of the parallel composition of two CHTTAs having ν_1 and ν_2 as composed valuations.

Formally, the set of configurations $Conf(A)$ of a CHTTA A is inductively defined as follows:

- if $A = \langle(\Sigma, X, S, Q, q_0, Inv, \delta), \mu\rangle$, then $Conf(A) = \{(Q \setminus S) \times V_X\} \cup \{(q.c, v.\nu) \mid q \in S \wedge v \in V_X \wedge (c, \nu) \in Conf(\mu(q))\}$;
- if $A = A_1||A_2$ then $Conf(A) = \{(c_1; c_2, \nu_1; \nu_2) \mid (c_1, \nu_1) \in Conf(A_1) \wedge (c_2, \nu_2) \in Conf(A_2)\}$.

For a composed valuation ν and a time value $t \in \mathbb{R}^{\geq 0}$, let $\nu + t$ denote the composed valuation such that $(v + t)(x) = v(x) + t$, for each valuation v occurring in ν.

Given a CHTTA A and a configuration $(c, \nu) \in Conf(A)$, $Inv_A(c, \nu)$ holds in the following cases:

- if $A = \langle(\Sigma, X, S, Q, Inv, q_0, \delta), \mu\rangle$ and $c \notin S \cup \{\odot, \otimes\}$, then $\nu \models Inv(c)$ must hold;
- if $A = \langle(\Sigma, X, S, Q, Inv, q_0, \delta), \mu\rangle$ and $(c, \nu) = (q.c', v.\nu')$ with $q \in S$, $v \models Inv(q)$ and $Inv_{\mu(q)}(c', \nu')$ must hold;
- if $A = A_1||A_2$ and $(c, \nu) = (c_1; c_2, \nu_1; \nu_2)$, then $Inv_{A_1}(c_1, \nu_1)$ and $Inv_{A_2}(c_2, \nu_2)$ must hold.

The initial configuration of A, denoted $Init(A) \in Conf(A)$, is the configuration (c, ν) such that each state occurring in c is an initial state and each valuation occurring in ν is $\mathbf{0}$.

We give a semantics of CHTTAs in SOS style [27] as a labelled transition system where states are pairs (A, tc) with $A \in \text{CHTTA}_{\mathcal{A}}^{\Sigma_{Pub}}$ and $tc \in Conf(A)$, and labels are in $\mathbb{R}^{>0} \cup \bigcup_i \Sigma^i \cup \{\tau\}$.

In order to simplify the SOS semantics for CHTTAs we introduce a notion of structural equivalence for pairs (A, tc), accounting for commutativity and associativity of parallelism. The relation $\approx$ is the least equivalence relation satisfying $(A_1||A_2, tc_1; tc_2) \approx (A_2||A_1, tc_2; tc_1)$ and $(A_1||(A_2||A_3), tc_1; (tc_2; tc_3)) \approx ((A_1||A_2)||A_3, (tc_1; tc_2); tc_3)$. Moreover, given an untimed parallel configuration $c = c_1; \ldots; c_n$ we use the following notations: $c \approx \odot$ if $c_i = \odot$ for all i; and $c \approx \otimes$ if $c_i \in \{\odot, \otimes\}$ for all i and there exists some j such that $c_j = \otimes$.

Definition 2.5 (Semantics of CHTTAs) *Given $A \in CHTTA_{\mathcal{A}}^{\Sigma_{Pub}}$, the semantics of a A is the least labelled transition relation $\xrightarrow{\alpha}$ over $\{A\} \times Conf(A)$ closed with respect to structural equivalence and satisfying the rules in Figure 2.*

$$\frac{t \in \mathbb{R}^{>0} \quad \forall t' \in [0,t].Inv_A(c, \nu + t')}{(A,(c,\nu)) \xrightarrow{t} (A,(c,\nu+t))} \tag{T}$$

$$\frac{(q,\alpha,\phi,B,q') \in \delta \quad v \models \phi \quad q' \notin S \quad Inv_A(q', v[B])}{(\langle A,\mu\rangle,(q,v)) \xrightarrow{\alpha} (\langle A,\mu\rangle,(q',v[B]))} \tag{C1}$$

$$\frac{(q,\alpha,\phi,B,q') \in \delta \quad v \models \phi \quad q' \in S \quad Init(\mu(q')) = (c,\nu) \quad Inv_{\mu(q')}(q'.c, v[B].\nu)}{(\langle A,\mu\rangle,(q,v)) \xrightarrow{\alpha} (\langle A,\mu\rangle,(q'.c, v[B].\nu))} \tag{C2}$$

$$\frac{(\mu(q),(c,\nu)) \xrightarrow{\alpha} (\mu(q),(c',\nu')) \quad \alpha \in \Sigma_{Pub} \cup \{\tau\}}{(\langle A,\mu\rangle,(q.c,v.\nu)) \xrightarrow{\alpha} (\langle A,\mu\rangle,(q.c',v.\nu'))} \tag{C3}$$

$$\frac{(A_1,(c_1,v)) \xrightarrow{\alpha} (A_1,(c_1',v')) \quad \alpha \in \Sigma_{Pub} \cup \{\tau\}}{(A_1||A_2,(c_1;c_2,v)) \xrightarrow{\alpha} (A_1||A_2,(c_1';c_2,v'))} \tag{P1}$$

$$\frac{(A_1,(c_1,v)) \xrightarrow{a!} (A_1,(c_1',v')) \quad (A_2,(c_2,v')) \xrightarrow{a?} (A_2,(c_2',v''))}{(A_1||A_2,(c_1;c_2,v)) \xrightarrow{\tau} (A_1||A_2,(c_1';c_2',v''))} \tag{P2}$$

$$\frac{c \approx \odot \quad (q,\boxdot,q') \in \delta \quad q' \notin S}{(\langle A,\mu\rangle,(q.c,v.\nu)) \xrightarrow{\tau} (\langle A,\mu\rangle,(q',v))} \tag{Com1}$$

$$\frac{c \approx \odot \quad (q,\boxdot,q') \in \delta \quad q' \in S \quad Init(\mu(q')) = (c',\nu')}{(\langle A,\mu\rangle,(q.c,v.\nu)) \xrightarrow{\tau} (\langle A,\mu\rangle,(q'.c',v.\nu'))} \tag{Com2}$$

$$\frac{c \approx \otimes \quad (q,\boxtimes,q') \in \delta \quad q' \notin S}{(\langle A,\mu\rangle,(q.c,v.\nu)) \xrightarrow{\tau} (\langle A,\mu\rangle,(q',v))} \tag{Ab1}$$

$$\frac{c \approx \otimes \quad (q,\boxtimes,q') \in \delta \quad q' \in S \quad Init(\mu(q')) = (c',\nu')}{(\langle A,\mu\rangle,(q.c,v.\nu)) \xrightarrow{\tau} (\langle A,\mu\rangle,(q'.c',v.\nu'))} \tag{Ab2}$$

where $A = (\Sigma, X, S, Q, Inv, q_0, \delta)$ except for rule (T) where A is any CHTTA.

Fig. 2. SOS semantics for CHTTAs.

Rule (T) allows the elapsing of time for a generic CHTTA A. We note that the time t is the same for any TTA composing A. The invariant condition should hold in order to allow time to elapse.

Rules (C1) and (C2) describe the behavior of a flat TTA. From a configuration (q, v), the step is performed due to a transition (q, α, ϕ, B, q') such that the condition ϕ is satisfied by v. After the step, the flat TTA is in the configuration composed by state q' and a new valuation where clocks in B are reset. The invariant condition should hold, in the reached state, for the new valuation. If

q' is a superstate (rule (C2)), then the CHTTA $\mu(q')$ becomes active inside q'.

The synchronisation step is described by rule (P2). By definition of the relation $\approx$ also CHTTAs that are not neighborhood in the parallel composition can communicate.

Rules (C3) and (P1) allow expanding the step of a TTA which is a component of a CHTTA. Rule (C3) deals with the hierarchical composition and rule (P1) deals with the parallel composition. The label of the step is either τ or a public channel. Hence, thanks to rule (P2), communication between TTAs in parallel is allowed both for private and public channels, while for TTAs in different superstates the communication is allowed only if the channel is public. Moreover, we note that the step we are expanding cannot be a time step. Hence, since time steps can be performed only by the root, the time elapsed is the same for each TTA composing the CHTTA we are considering.

Each execution of a superstate terminates with either a commit or an abort state. Rules (Com1) and (Com2) deal with the case in which the commit of the superstate takes the TTA to a basic state or to a superstate, respectively, and rules (Ab1) and (Ab2) deal with the case in which the abort of the superstate takes the TTA to a basic state or to a superstate, respectively.

Given a string $w = \alpha_1 \dots \alpha_m$, we will write $(A,(c,\nu)) \stackrel{w}{\Longrightarrow} (A,(c',\nu'))$ to denote the existence of a sequence of steps $(A,(c,\nu)) \stackrel{\alpha_1}{\longrightarrow} \dots \stackrel{\alpha_m}{\longrightarrow} (A,(c',\nu'))$. We denote with $|w| = m$ the length of w and with $w[i] = \alpha_i$ the $i-$th label.

Definition 2.6 (Accepted Language) *We denote with* $\mathcal{L}(A,\Sigma_V)$ the language accepted by a CHTTA A w.r.t. a set of visible actions $\Sigma_V \subseteq \Sigma_{Pub}$. *Namely,* $\mathcal{L}(A,\Sigma_V) = \{w \in (\{\tau\} \cup \Sigma_V \cup \mathbb{R}^{>0})^* \mid (A, Init(A)) \stackrel{w}{\Longrightarrow} (A,(\odot,\nu'))$ *or* $(A, Init(A)) \stackrel{w}{\Longrightarrow} (A,(\otimes,\nu'))\}$.

3 Deciding Reachability for CHTTAs

Reachability is interesting for proving properties. For Timed Automata the reachability problem is PSPACE–COMPLETE. In our case the problem is still decidable, but becomes EXPSPACE–COMPLETE.

Firstly, we give an algorithm for flattening a generic CHTTA, hence the reachability problem can be checked on the Timed Automaton resulting by the flattening. Due to the complexity of the flattening, the reachability problem for CHTTAs is EXPSPACE–COMPLETE. The increase of complexity is caused by the communication between different superstates.

Let $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$ and ϕ be a formula in $\Phi(X)$. With $\phi[Y := X]$ we denote the formula obtained by replacing each clock y_i appearing in ϕ with the clock x_i. Moreover, with $X_{i,j}$ we denote the set of clocks obtained by renaming each clock x in X with the clock $x^{i,j}$, more precisely $X_{i,j} = \{x_1^{i,j}, \ldots, x_n^{i,j}\}$.

Given a $CHTTA$ A, with $w(A)$ we denote the maximum width of the CHTTAs composing A. Namely:

$$w(\langle A_1, \mu_1\rangle \| \ldots \| \langle A_m, \mu_m\rangle) = max\{m, w(\langle A_1, \mu_1\rangle), \ldots, w(\langle A_m, \mu_m\rangle)\},$$

where $w(\langle A, \mu\rangle) = max\{w(\mu(q)) \mid q \in S\}$.

Moreover, $d(A)$ denotes the maximum depth of A. Namely:

$$d(\langle A_1, \mu_1\rangle \| \ldots \| \langle A_m, \mu_m\rangle) = max\{d(\langle A_1, \mu_1\rangle), \ldots, d(\langle A_m, \mu_m\rangle)\},$$

where $d(\langle A, \mu\rangle) = 1 + max\{d(\mu(q)) \mid q \in S\}$.

Definition 3.1 *Let $\mathcal{A} = \{A^1, \ldots, A^n\}$, with $A^i = (\Sigma^i, X^i, S^i, Q^i, q_0^i, Inv^i, \delta^i)$, be a set of TTAs, and $A \in CHTTA_{\mathcal{A}}^{\Sigma_{Pub}}$. Given $\Sigma_V \subseteq \Sigma_{Pub}$, with $Flat(A, \Sigma_V)$ we denote the flat TTA $(\Sigma, X, \emptyset, Q, q_0, Inv, \delta)$ such that:*

- $\Sigma = \Sigma_V$;
- $X = \bigcup_{i \in [1, d(A)]} \bigcup_{j \in [1, w(A)]} X_{i,j}$;
- $Q = \{c \mid (c, \nu) \in Conf(A)\}$;
- $q_0 = c_0$ *where* $(c_0, \nu) = Init(A)$ *is the initial configuration of* A;
- $Inv(c) = \phi_1 \wedge \cdots \wedge \phi_m$ *such that:*
  - *for any* $q \in Q^h$ *appearing in* c *at position* i, j, $Inv^h(q)[X^h := X^h_{i,j}]$ *is equal to* ϕ_l, *for some* l.
  - *for any* l, ϕ_l *is equal to* $Inv^h(q)[X^h := X^h_{i,j}]$, *for some* q, h, i, j.
- δ *is such that:*
  - $(c, \tau, true, \emptyset, c')$ *is in* δ *if there exists a step* $(A, (c, \nu)) \xrightarrow{\tau} (A, (c', \nu'))$ *triggered by either a commit or an abort transition;*
  - (c, α, ϕ, B, c') *is in* δ *if there exists a step* $(A, (c, \nu)) \xrightarrow{\alpha} (A, (c', \nu'))$, *with* $\alpha \in \Sigma_V$ *triggered by the transition* (q, α, ϕ, B, q') *of a TTA* A^i;
  - (c, τ, ϕ, B, c') *is in* δ *if there exists a step* $(A, (c, \nu)) \xrightarrow{\tau} (A, (c', \nu'))$ *triggered by the transition* $(q^1, a!, \phi^1, B^1, p^1)$ *of the TTA* A^i *at position* i_1, j_1 *and by the transition* $(q^2, a?, \phi^2, B^2, p^2)$ *of the TTA* A^j *at position* i_2, j_2 *such that* $\phi = (\phi_1[X^i := X^i_{i_1,j_1}]) \wedge (\phi_2[X^j := X^j_{i_2,j_2}])$ *and* $B = B^1_{i_1,j_1} \cup B^2_{i_2,j_2}$.

By induction on the length of the sequence of steps we have the following proposition.

Proposition 3.2 *Let A be a CHTTA; it holds that $(A,(c_0,v_0)) \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} (A,(c_n,v_n))$ is a sequence of steps of A iff $(A',(c_0,v'_0)) \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} (A',(c_n,v'_n))$ is a sequence of steps of A' where $A' = Flat(A,\Sigma_V)$.*

As a consequence we have that the class of CHTTAs is equivalent to the class of Timed automata.

Proposition 3.3 *Let $\mathcal{A} = \{A^1,\ldots,A^n\}$ and $A \in CHTTA_{\mathcal{A}}^{\Sigma_{Pub}}$ where each A^i has at most h states and k clocks. The reachability problem for A can be computed in $O(h^{w(A)^{d(A)}} \cdot 2^{k\cdot d(A)\cdot w(A)})$.*

PROOF *$Flat(A,\Sigma_V)$ has at most $h^{w(A)^{d(A)}}$ states. Actually, the root is a parallel composition of at most $w(A)$ CHTTAs. Each of them has depth at most $d(A)-1$ and, by induction, has $h^{w(A)^{d(A)-1}}$ state configurations, thus implying that the number of state configurations is $h^{w(A)^{d(A)}}$.*

Given a Timed Automaton with d states and l clocks, the reachability problem can be solved in $d\cdot 2^l$ (see [2]). Hence, since $Flat(A,\Sigma_V)$ has $h^{w(A)^{d(A)}}$ states and at most $k\cdot d(A)\cdot w(A)$ clocks, the reachability problem for A can be computed in $h^{w(A)^{d(A)}} \cdot 2^{k\cdot d(A)\cdot w(A)}$. □

Thus, the reachability problem for a CHTTA A is EXPSPACE–COMPLETE w.r.t. m, $w(A)$ and $d(A)$. Moreover, as it happens for Timed Automata (see [2]), the reachability problem for a CHTTA A is PSPACE–COMPLETE w.r.t. the number of clocks of A.

Proposition 3.4 *Let $\mathcal{A} = \{A^1,\ldots,A^n\}$ and $A \in CHTTA_{\mathcal{A}}^{\Sigma_{Pub}}$, where each A^i has at most m states. The reachability problem for A is EXPSPACE–COMPLETE w.r.t. m, $w(A)$ and $d(A)$.*

PROOF *The reachability problem for CHMs is EXPSPACE–COMPLETE w.r.t. m, $w(A)$ and $d(A)$ (see [3]). The same holds for* untimed *CHTTAs. As a consequence, the reachability problem for CHTTAs is at least EXPSPACE–COMPLETE w.r.t. m, $w(A)$ and $d(A)$. Therefore, if the reachability problem is PSPACE–COMPLETE for $Flat(A,\Sigma_V)$, then the thesis holds. But $Flat(A,\Sigma_V)$ has at most $k\cdot w(A)\cdot d(A)$ clocks, that is a polynomial number of clocks, and hence the thesis holds since the reachability problem for Timed Automata is PSPACE–COMPLETE (see [2]).* □

4 Compositional Patterns for Long–Running Transactions

A *long–running transaction* is composed by atomic activities (called *subtransactions* or simply *activities*) that should be executed completely. Atomicity

for activities means that they are either successfully executed (*committed*) or no effect is observed if their execution fails (*aborted*).

Partial executions of a long–running transaction are not desirable, and, if they occur, they must be compensated for. Therefore, all the activities A_i in a long–running transaction have a compensating activity B_i that can be invoked to repair from the effects of a successful execution of A_i if some failure occurs later. In order to guarantee that after possible failures a state is reached in which the effects of failing activities are repaired, compensations are assumed to be transactions that always complete their execution successfully. This assumption is usual in formalisations of long–running transactions [7,22,23]. In many real cases, we can definitively consider compensations to be atomic activities executed in a short time and satisfying the ACID properties. However, it is not excluded that a compensation may be a complex activity with subactivities which abort. In this case, the compensation should be built in such a way to guarantee a final successful commit. We denote with $A⇱B$ the association of compensation B with activity A. To simplify the analysis, we do not allow compensations to be long–running transactions themselves. In any case, an extension taking this feature into account could be provided quite easily.

Following the approach introduced in [10], we consider composition patterns for transactional activities with compensations, namely sequential composition, parallel composition and nesting. Given two transactional activities with compensations $A_1⇱B_1$ and $A_2⇱B_2$, we denote with $A_1⇱B_1 \cdot A_2⇱B_2$ their sequential composition and with $A_1⇱B_1 || A_2⇱B_2$ their parallel composition. Moreover, we denote with $\{A⇱B\}_i$ the nesting of transaction $A⇱B$, where i is a unique index labelling the nested transaction. To model these patterns we consider the semantics proposed in [10], we describe both activities and compensations as CHTTAs, and we formulate the composition patterns of transactional activities as compositions of CHTTAs. For each composition pattern we prove *correct completion* and *correct compensation*, namely we prove that the composed CHTTA representing a pattern always reaches the commit or the abort state, and activates compensations accordingly to the semantics of the pattern.

4.1 Sequential Transactions

Activities $A_1, \ldots, A_n$ composing a *sequential transaction* are assumed to be executed sequentially, namely, when activity A_i is committed, activity A_{i+1} starts its execution. Compensation activities $B_1, \ldots, B_n$ are associated with each activity A_i. Following the semantics in [10], transactions of this kind must be guaranteed that either the entire sequence $A_1, \ldots, A_n$ is executed or the compensated sequence $A_1, \ldots, A_i, B_i, \ldots, B_1$ is executed for some $i < n$. The first case means that all activities in the sequence completed successfully,

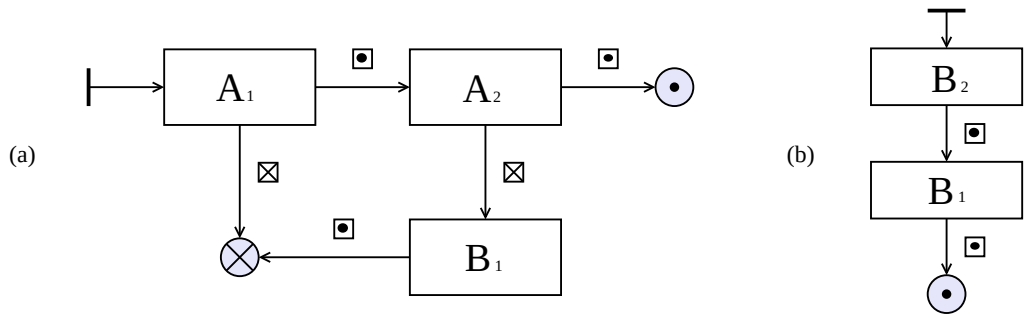

Fig. 3. Pattern for Sequential Transactions.

thus representing a successful commit of the whole transaction. The second case stands for the abort of activity A_{i+1}; hence, all the activities already completed $(A_1, \ldots, A_i)$ are recovered by executing the compensating activities $(B_i, \ldots, B_1)$.

In Figure 3 (a) we show the CHTTA $A = [\![A_1 \Cap B_1 \cdot A_2 \Cap B_2]\!]^S$ modelling the pattern of sequential transactions. We consider just two activities A_1, A_2 and compensations B_1, B_2. Note that, since the transaction is composed by only two activities, the compensation B_2 is not executed. This is because compensations are invoked only for activities that complete successfully, however, if activity A_2 commits, then the whole transaction successfully commits, and no compensation needs to be invoked. The compensation $B = [\![A_1 \Cap B_1 \cdot A_2 \Cap B_2]\!]^S_C$ of the whole transactional activity A is defined as the sequential execution of the compensations B_2 and B_1 (see Figure 3 (b)).

Definition 4.1 (Sequential Pattern) *Given $A_1, A_2, B_1, B_2 \in CHTTA^{\Sigma_{Pub}}_{\mathcal{A}}$, the* sequential composition *of activities A_1, A_2 with compensations B_1, B_2 is the $CHTTA^{\Sigma_{Pub}}_{\mathcal{A}}$ $A = [\![A_1 \Cap B_1 \cdot A_2 \Cap B_2]\!]^S$ depicted in Figure 3 (a). The* compound compensation *of A is defined as the $CHTTA^{\Sigma_{Pub}}_{\mathcal{A}}$ $B = [\![A_1 \Cap B_1 \cdot A_2 \Cap B_2]\!]^S_C$ depicted in Figure 3 (b).*

Formally, the sequential composition $A = [\![A_1 \Cap B_1 \cdot A_2 \Cap B_2]\!]^S$ *is defined as:*

$$A = \langle (\emptyset, \emptyset, \{s_1, s_2, s_3\}, \{s_1, s_2, s_3, q_0, \odot, \otimes\}, q_0, Inv_{true}, \delta), \mu \rangle$$

where

$$\delta = \{(q_0, \tau, true, \emptyset, s_1), (s_1, \boxdot, s_2), (s_1, \boxtimes, \otimes), (s_2, \boxdot, \odot), (s_2, \boxtimes, s_3), (s_3, \boxdot, \otimes)\}$$

and $\mu = \{(s_1, A_1), (s_2, A_2), (s_3, B_1)\}$. *The* compound compensation $B = [\![A_1 \Cap B_1 \cdot A_2 \Cap B_2]\!]^S_C$ *is:*

$$B = \langle (\emptyset, \emptyset, \{s_1, s_2\}, \{s_1, s_2, q_0, \odot, \otimes\}, q_0, Inv_{true}, \delta'), \mu' \rangle$$

with

$$\delta' = \{(q_0, \tau, true, \emptyset, s_2), (s_2, \boxdot, s_1), (s_1, \boxdot, \odot)\}$$

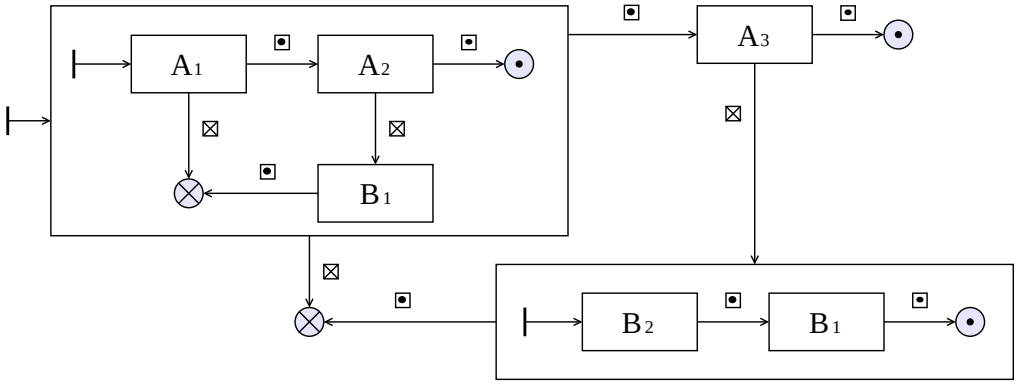

Fig. 4. Composing Sequential Transactions.

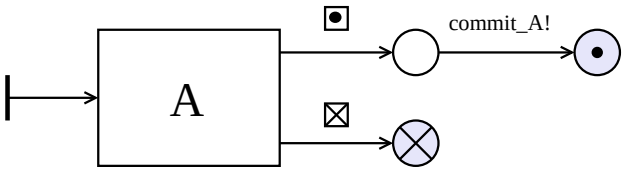

Fig. 5. The wrapped CHTTA A^M.

and $\mu' = \{(s_1, B_1), (s_2, B_2)\}$.

Considering only two activities in the sequential pattern is not a real limitation, since the case of n activities may be reduced by iteratively grouping the activities in pairs. Intuitively, $A = [\![A_1 \triangleright B_1 \cdot A_2 \triangleright B_2 \cdot A_3 \triangleright B_3]\!]^S = [\![A' \triangleright B' \cdot A_3 \triangleright B_3]\!]^S$, where $A' = [\![A_1 \triangleright B_1 \cdot A_2 \triangleright B_2]\!]^S$ and $B' = [\![A_1 \triangleright B_1 \cdot A_2 \triangleright B_2]\!]^S_C$ is the compensation for the whole sequential subtransaction A' (see Figure 4).

In order to prove the correctness of our definitions of composition patterns, we introduce the notion of *wrapped* CHTTAs. Intuitively, given a CHTTA A, we call A-*wrapped* the automaton A^M which performs the special action $commit_A!$ before reaching the final commit state.

Definition 4.2 (Wrapping) *Given a CHTTA* A*, with* A^M *we denote the* A–wrapped *CHTTA depicted in Figure 5. More Formally:*

$$A^M = \langle (\{commit_A!\}, \emptyset, \{s\}, \{s, q_0, q_1, \odot, \otimes\}, q_0, Inv_{true}, \delta), \mu \rangle$$

where $\delta = \{(q_0, \tau, true, \emptyset, s), (s, \boxdot, q_1), (s, \boxtimes, \otimes), (q_1, commit_A!, true, \emptyset, \odot)\}$ *and* $\mu(s) = A$.

As the reader has noticed, the previous definitions are given in both a pictorial and a more formal way. For simplicity, we will give the definitions of the patterns in the next sections only in the pictorial way. Taking Definitions 4.1 and 4.2 as a model, we are confident that the reader can easily deduce how the formal definitions could be extracted from their graphical representations.

The next lemma derives immediately from the definition of wrapping, stating

that given a CHTTA A, the wrapped CHTTA A^M either commits or aborts, respectively, whenever the original automaton A commits or aborts.

Lemma 4.3 *Given a CHTTA A, $(A,(c,\nu)) \stackrel{w}{\Longrightarrow} (A,(c',\nu'))$, with $c \not\approx \odot$ and $c \not\approx \otimes$ and either $c' \approx \odot$ or $c' \approx \otimes$ if and only if $(A^M,(s\cdot c,\epsilon\cdot\nu) \stackrel{w'}{\Longrightarrow} (A^M,(s\cdot\hat{c},\epsilon\cdot\hat{\nu}))$, where (given $\tilde{z}\in\{\mathbb{R}^{>0}\}^*$):*

$$\begin{cases} w' = \tilde{z}\cdot\tau\cdot w\cdot\tau\cdot commit_A! \;\; and \;\; \hat{c}=\odot & \text{if } c'\approx\odot \\ w' = \tilde{z}\cdot\tau\cdot w\cdot\tau \;\; and \;\; \hat{c}=\otimes & \text{if } c'\approx\otimes \end{cases}$$

To prove the correct completion of the sequential pattern let us consider the set of special actions $\{commit_{A_1}!, commit_{B_1}!, \ldots, commit_{A_n}!, commit_{B_n}!\} \subseteq \Sigma_V$. A sequential composition of activities correctly commits if and only if all the activities composing the pattern correctly commit.

Theorem 4.4 (Correct Completion) *Given the sequential composition pattern $A = [\![A_1^M \triangleright B_1^M \cdot \ldots \cdot A_n^M \triangleright B_n^M]\!]^S$, it holds that $(A, Init(A)) \stackrel{w}{\Longrightarrow} (A,(\odot,\nu))$ if and only if $w\in\mathcal{L}(A,\Sigma_V)$ and $w = \tilde{x}_1\cdot commit_{A_1}!\cdot\ldots\cdot\tilde{x}_n\cdot commit_{A_n}!\cdot\tilde{x}_{n+1}$ where $\tilde{x}_i\in(\{\tau\}\cup\mathbb{R}^{>0})^*$.*

PROOF *By Definition 2.6 we have that if $w\in\mathcal{L}(A,\Sigma_V)$, then $(A,Init(A)) \stackrel{w}{\Longrightarrow} (A,(\odot,\nu))$. Hence we need to prove only the $\Rightarrow$ implication.*

Given $c = c_1;\ldots;c_n$ with $c_i \not\approx \odot$, we prove by induction on n that whenever $(A,Init(A)) \stackrel{w}{\Longrightarrow} (A,(\odot,\nu))$, then $w = \tilde{x}_1\cdot commit_{A_1}!\cdot\ldots\cdot\tilde{x}_n\cdot commit_{A_n}!\cdot\tilde{x}_{n+1}$ where $\tilde{x}_i\in(\{\tau\}\cup\mathbb{R}^{>0})^*$.

If $n=1$ then the thesis holds by Lemma 4.3.

If $n>1$, then $A_1^M \triangleright B_1^M \cdot \ldots \cdot A_n^M \triangleright B_n^M$ is synthesised as $A' \triangleright B' \cdot A_n^M \triangleright B_n^M$ where $A' = A_1^M \triangleright B_1^M \cdot \ldots \cdot A_{n-1}^M \triangleright B_{n-1}^M$ and B is the sequence of compensations $B_1,\ldots,B_{n-1}$. By induction, if $(A',Init(A)) \stackrel{w'}{\Longrightarrow} (A',(\odot,\nu))$, then we have $w' = \tilde{y}_1\cdot commit_{A_1}!\cdot\ldots\cdot\tilde{y}_{n-1}\cdot commit_{A_{n-1}}!\cdot\tilde{y}_n$ where $\tilde{y}_i\in(\{\tau\}\cup\mathbb{R}^{>0})^*$. Now, again by Lemma 4.3, if $(A_n^M,(c,\nu)) \stackrel{w''}{\Longrightarrow} (A_n^M,(\odot,\nu'))$, then we get the string $w'' = \tilde{z}\cdot commit_{A_n}!\cdot\tilde{z}'$. Hence, for a fixed $w = w'\cdot w''$ where $\tilde{x}_1=\tilde{y}_1,\ldots,\tilde{x}_{n-1}=\tilde{y}_{n-1}$ and $\tilde{x}_n = \tilde{y}\cdot\tilde{z}$ and $\tilde{x}_{n+1}=\tilde{z}'$ we have that $(A,Init(A)) \stackrel{w}{\Longrightarrow} (A,(\odot,\nu'))$ with $w = \tilde{x}_1\cdot commit_{A_1}!\cdot\ldots\cdot\tilde{x}_n\cdot commit_{A_n}!\cdot\tilde{x}_{n+1}$ where $\tilde{x}_i\in(\{\tau\}\cup\mathbb{R}^{>0})^*$. □

When some activity in the sequential pattern aborts all the activities that already completed are recovered by executing their compensating activities. The result for the correct compensation of the sequential composition pattern can be formalised as follows.

Theorem 4.5 (Correct Compensation) *Given the sequential composition*

pattern $A = [\![A_1^M \Rsh B_1^M \cdot \ldots \cdot A_n^M \Rsh B_n^M]\!]^S$, $(A, Init(A)) \stackrel{w}{\Longrightarrow} (A, (\otimes, \nu))$ *if and only if,* $w \in \mathcal{L}(A, \Sigma_V)$ *and, for some* $k \in [1, n]$, $w = \tilde{x}_1 \cdot commit_{A_1}! \cdot \ldots \cdot \tilde{x}_{k-1} \cdot commit_{A_{k-1}}! \cdot \tilde{x}'_{k-1} \cdot commit_{B_{k-1}}! \cdot \ldots \cdot \tilde{x}'_1 \cdot commit_{B_1}! \cdot \tilde{x}'$ *where* $\tilde{x}_i, \tilde{x}'_i \in (\{\tau\} \cup \mathbb{R}^{>0})^*$.

PROOF *By Definition 2.6 we have that if* $w \in \mathcal{L}(A, \Sigma_V)$, *then* $(A, Init(A)) \stackrel{w}{\Longrightarrow} (A, (\otimes, \nu))$. *Hence we need to prove only the* $\Rightarrow$ *implication.*

We prove by induction on n that, if$(A, Init(A)) \stackrel{w}{\Longrightarrow} (A, (\otimes, \nu))$, *then we have* $w \in \mathcal{L}(A)$ *and, for some* $k \in [1, n]$, $w = \tilde{x}_1 \cdot commit_{A_1}! \cdot \ldots \cdot \tilde{x}_{k-1} \cdot commit_{A_{k-1}}! \cdot \tilde{x}'_{k-1} \cdot commit_{B_{k-1}}! \cdot \ldots \cdot \tilde{x}'_1 \cdot commit_{B_1}! \cdot \tilde{x}'$ *where* $\tilde{x}_i, \tilde{x}'_i \in \{\{\tau\} \cup \mathbb{R}^{>0}\}^*$.

If $n = 1$ *the thesis holds by Lemma 4.3.*

If $n > 1$, *then* $A_1^M \Rsh B_1^M \cdot \ldots \cdot A_n^M \Rsh B_n^M$ *is synthesised as* $A' \Rsh B' \cdot A_n^M \Rsh B_n^M$ *where* $A' = A_1^M \Rsh B_1^M \cdot \ldots \cdot A_{n-1}^M \Rsh B_{n-1}^M$ *and* B' *is the sequence of compensations* $B_{n-1}^M, \ldots, B_1^M$. *We have two cases. If* $(A', Init(A')) \stackrel{w'}{\Longrightarrow} (A', (\otimes, \nu))$, *then the thesis holds by induction. Otherwise, if* $(A', Init(A')) \stackrel{w'}{\Longrightarrow} (A', (\odot, \nu))$, *then by Theorem 4.4,* $w' = \tilde{y}_1 \cdot commit_{A_1}! \cdot \ldots \cdot \tilde{y}_{n-1} \cdot commit_{A_{n-1}}! \cdot \tilde{y}_n$ *where* $\tilde{y}_i \in (\{\tau\} \cup \mathbb{R}^{>0})^*$. *Now, by Lemma 4.3, if* $(A_n^M, (c, \nu)) \stackrel{w''}{\Longrightarrow} (A_n^M, (\otimes, \nu'))$, *then* $w' = \tilde{z}$, *and,* $(B', (Init(B')) \stackrel{w'''}{\Longrightarrow} (B', (\odot, \nu''))$ *with* $w''' = \tilde{y}'_{n-1} \cdot commit_{B_{n-1}}! \cdot \ldots \cdot \tilde{y}'_1 \cdot commit_{B_1}! \cdot \tilde{y}'$. *Therefore, for a fixed* $w = w \cdot w' \cdot w'' \cdot w'''$; $\tilde{x}_1 = \tilde{y}_1, \ldots, \tilde{x}_{n-1} = \tilde{y}_{n-1}$; $\tilde{x}'_1 = \tilde{y}'_1, \ldots, \tilde{x}'_{n-2} = \tilde{y}'_{n-2}$ *and* $\tilde{x}'_{n-1} = \tilde{z} \cdot \tilde{y}'_{n-1}$ *we have that* $(A, Init(A)) \stackrel{w}{\Longrightarrow} (A, (\otimes, \nu'))$ *and* $w = \tilde{x}_1 \cdot commit_{A_1}! \cdot \ldots \cdot \tilde{x}_{n-1} \cdot commit_{A_{n-1}}! \cdot \tilde{x}'_{n-1} \cdot commit_{B_{n-1}}! \cdot \ldots \cdot \tilde{x}'_1 \cdot commit_{B_1}! \cdot \tilde{x}'$ *where* $\tilde{x}_i, \tilde{x}'_i \in \{\{\tau\} \cup \mathbb{R}^{>0}\}^*$. □

4.2 Parallel Transactions

If activities $A_1, \ldots, A_n$ composing a *parallel transaction* are executed concurrently, the whole transaction terminates when all the activities A_i complete their execution. Again, we assume compensation activities $B_1, \ldots B_n$. Again, from the semantics in [10], if all the activities terminate successfully then the whole transaction reaches a commit state. If some A_i aborts, then compensation activities should be invoked for the activities that completed successfully. In this latter case, the final result of the whole transaction is "abort".

The pattern for parallel transactions is shown in Figure 6. As for sequential transactions, we consider only two activities A_1, A_2 with compensations B_1, B_2 composed in parallel, thus resulting in the CHTTA $A = [\![A_1 \Rsh B_1 || A_2 \Rsh B_2]\!]^P$ of Figure 6 (a). We remark that, by the semantics of CHTTAs, the parallel operator $||$ is assumed to be commutative and associative. In such a pattern, activities A_1 and A_2 are executed concurrently together with a *controller* that invokes compensations when one of the two activities commits and the other

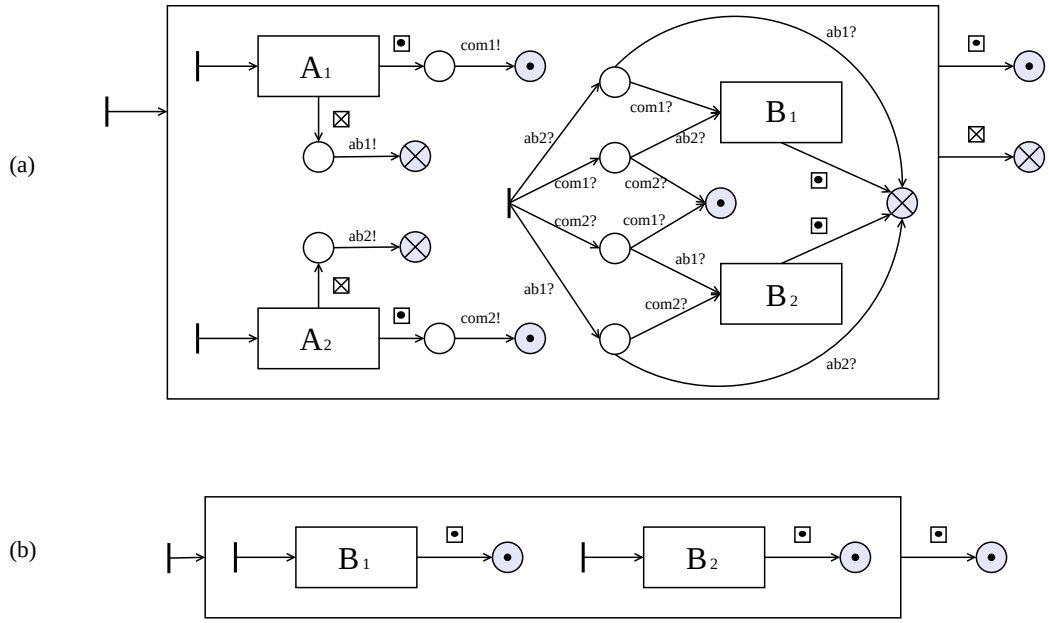

Fig. 6. Pattern for Parallel Transactions.

aborts (see the synchronisation on channels *abi* and *comi*). We define the compensation $B = [\![A_1 \Cap B_1 || A_2 \Cap B_2]\!]^P_C$ of A as the concurrent execution of the compensations B_1 and B_2 (see Figure 6 (b)).

Definition 4.6 (Parallel Composition) *Let $A_1, A_2, B_1, B_2 \in CHTTA^{\Sigma_{Pub}}_{\mathcal{A}}$, we define the* parallel composition *of activities A_1 and A_2 with compensations B_1 and B_2 as the $CHTTA^{\Sigma_{Pub}}_{\mathcal{A}}$ $A = [\![A_1 \Cap B_1 || A_2 \Cap B_2]\!]^P$ depicted in Figure 6 (a). The* compound compensation *of A is the $CHTTA^{\Sigma_{Pub}}_{\mathcal{A}}$ $B = [\![A_1 \Cap B_1 || A_2 \Cap B_2]\!]^P_C$ depicted in Figure 6 (b).*

As for sequential transactions, considering only two activities in the parallel pattern is not a limitation, since the case of n activities may be reduced by iteratively grouping the activities in pairs. For instance, we have that $A = [\![A_1 \Cap B_1 || A_2 \Cap B_2 || A_3 \Cap B_3]\!]^P = [\![A' \Cap B' || A_3 \Cap B_3]\!]^P$, where the automaton A' is equal to $[\![A_1 \Cap B_1 || A_2 \Cap B_2]\!]^P$ and $B' = [\![A_1 \Cap B_1 || A_2 \Cap B_2]\!]^P_C$ is the compensation for the whole parallel subtransaction A'.

The parallel composition pattern reaches a commit state if and only if all the activities composing it terminate successfully.

Theorem 4.7 (Correct Completion) *Given the parallel composition $A = [\![A^M_1 \Cap B^M_1 || \ldots || A^M_n \Cap B^M_n]\!]^P$, $(A, Init(A)) \stackrel{w}{\Longrightarrow} (A, (\odot, \nu))$ if and only if, $w \in \mathcal{L}(A, \Sigma_V)$ and $\forall i \in [1,n].\exists!\ j \in [1,|w|].\ w[j] = commit_{A_i}!$.*

PROOF *Since executions of activities A_i and compensations B_i do not interfere, we can assume that all compensations B_i are performed after the commit of activities A_i that terminate successfully.*

Hence, the theorem can be reformulated as follows.

Given $A = [\![A_1^M \Rsh B_1^M || \ldots || A_n^M \Rsh B_n^M]\!]^P$, $(A, Init(A)) \stackrel{w}{\Longrightarrow} (A, (\odot, \nu))$ *if and only if* $w \in \mathcal{L}(A, \Sigma_V)$ *and there exists a permutation* $(i_1, \ldots, i_n)$ *of* $(1, \ldots, n)$ *such that* $w = \tilde{x}_1 \cdot commit_{A_{i_1}}! \cdot \ldots \cdot \tilde{x}_n \cdot commit_{A_{i_n}}! \cdot \tilde{x}_{n+1}$ *where* $\tilde{x}_i \in (\{\tau\} \cup \mathbb{R}^{>0})^*$.

By Definition 2.6, $w \in \mathcal{L}(A, \Sigma_V)$ *implies* $(A, Init(A)) \stackrel{w}{\Longrightarrow} (A, (\odot, \nu))$. *Hence, we need to prove only the* $\Rightarrow$ *implication.*

Given $c = c_1; \ldots; c_n$ *with* $c_i \not\approx \odot$, *we can prove by induction on* n *that whenever* $(A, (c, \nu)) \stackrel{w}{\Longrightarrow} (A, (\odot, \nu'))$, *then* $w \in \mathcal{L}(A, \Sigma_V)$ *and there exists a permutation* $(i_1, \ldots, i_n)$ *of* $(1, \ldots, n)$ *such that* $w = \tilde{x}_1 \cdot commit_{A_{i_1}}! \cdot \ldots \cdot \tilde{x}_n \cdot commit_{A_{i_n}}! \cdot \tilde{x}_{n+1}$ *where* $\tilde{x}_i \in \{\{\tau\} \cup \mathbb{R}^{>0}\}^*$.

If $n = 1$ *then the thesis holds by Lemma 4.3.*

If $n > 1$, *then, given a sequence* $(A, (c, \nu)) \stackrel{w'}{\Longrightarrow} (A, (c'_1; \ldots; c'_n, \nu'))$ *such that* $c'_k = \odot$ *for some* $1 \leq k \leq n$, *by Lemma 4.3 we have* $w' = \tilde{z} \cdot commit_{A_k} \cdot \tilde{z}'$. *Now, since* A_{k+1}^M *has committed and hence it does not participate in communications, we have that* $(A, (c'_1; \ldots; c'_n, \nu')) \stackrel{w''}{\Longrightarrow} (A, (\odot, \nu''))$ *iff* $(A', (c', \nu')) \stackrel{w''}{\Longrightarrow} (A', (\odot, \nu''))$, *where* $A' = [\![A_1^M \Rsh B_1^M || \ldots || A_{k-1}^M \Rsh B_{k-1}^M || A_{k+1}^M \Rsh B_{k+1}^M || \ldots || A_n^M \Rsh B_n^M]\!]^P$ *and* $c' = c'_1; \ldots; c'_{k-1}; c'_{k+1}; \ldots; c'_n$. *By induction, if* $(A', (c', \nu')) \stackrel{w''}{\Longrightarrow} (A', (\odot, \nu''))$, *then there exists a permutation* $(j_1, \ldots, j_{n-1})$ *of* $(1, \ldots, n) \setminus \{k\}$ *such that* $w'' = \tilde{y}_1 \cdot commit_{A_{j_1}}! \cdot \ldots \cdot \tilde{y}_{n-1} \cdot commit_{A_{j_{n-1}}}! \cdot \tilde{y}_n$ *where* $\tilde{y}_i \in \{\{\tau\} \cup \mathbb{R}^{>0}\}^*$. *Hence, for a fixed* $w = w' \cdot w''$ *and* $(i_1, \ldots, i_n) = (k, j_1, \ldots, j_{n-1})$ *and* $\tilde{x}_1 = \tilde{z}$, $\tilde{x}_2 = \tilde{z}' \cdot \tilde{y}_1$, $\tilde{x}_3 = \tilde{y}_2, \ldots, \tilde{x}_{n+1} = \tilde{y}_n$ *and* $\tilde{x}_n = \tilde{y} \cdot \tilde{z}$ *and* $\tilde{x}_{n+1} = \tilde{z}'$ *we have that* $(A, (c, \nu)) \stackrel{w}{\Longrightarrow} (A, (\odot, \nu'))$ *and* $w = \tilde{x}_1 \cdot commit_{A_{i_1}}! \cdot \ldots \cdot \tilde{x}_n \cdot commit_{A_{i_n}}! \cdot \tilde{x}_{n+1}$, *where* $\tilde{x}_i \in (\{\tau\} \cup \mathbb{R}^{>0})^*$. *Since* $Init(A) = (c_1; \ldots; c_n)$ *satisfies the condition* $c_i \not\approx \odot$, *the thesis holds.* $\square$

If some activity of the parallel composition aborts, then compensation activities are invoked for the activities that completed successfully.

Theorem 4.8 (Correct Compensation) *Given the parallel composition* $A = [\![A_1^M \Rsh B_1^M || \ldots || A_n^M \Rsh B_n^M]\!]^P$, $(A, Init(A)) \stackrel{w}{\Longrightarrow} (A, (\otimes, \nu))$ *if and only if* $w \in \mathcal{L}(A)$ *and, there exists* $Committed \subset \{A_1, \ldots, A_n\}$ *such that* $\forall A_i \notin Committed$ $w[j] \neq commit_{A_i}!$ *and* $\forall A_i \in Committed$ $\exists!\ j \in [1, |w|[$ *such that* $w[j] = commit_{A_i}! \wedge \exists!\ k \in]j, |w|]$ *such that* $w[k] = commit_{B_i}!$.

PROOF *The theorem can be reformulated as follows.*

Given $A = [\![A_1^M \Rsh B_1^M || \ldots || A_n^M \Rsh B_n^M]\!]^P$, $(A, Init(A)) \stackrel{w}{\Longrightarrow} (A, (\otimes, v))$ *if and only if* $w \in \mathcal{L}(A)$ *and there exists a proper subset* D *of* $\{1, \ldots, n\}$ *such that* $w = \tilde{x}_1 \cdot commit_{X_1}! \cdot \ldots \cdot \tilde{x}_{|D|} \cdot commit_{X_{|D|}}! \cdot \tilde{x}_{|D|+1} \cdot commit_{X_{|D|+1}}! \cdot \ldots \cdot \tilde{x}_{2|D|} \cdot commit_{X_{2|D|}}! \cdot \tilde{x}_{2|D|+1}$ *where:*

- *if* $X_i = A_i$*, then* $i \in D$*;*
- *for any* $i \in D$ *there exist* k *and* h *in* $[1, 2 \cdot |D|]$ *with* $k < h$ *and such that* $A_i = X_k$ *and* $B_i = X_h$*;*
- *for any* i*,* $\tilde{x}_1, \ldots, \tilde{x}_{2\cdot|D|+1} \in (\{\tau\} \cup \mathbb{R}^{>0})^*$*.*

By Definition 2.6, $w \in \mathcal{L}(A, \Sigma_V)$ *implies that* $(A, Init(A)) \stackrel{w}{\Longrightarrow} (A, (\odot, \nu))$*. Hence, we need to prove only the* $\Rightarrow$ *implication.*

Let c *be a configuration of* A*; we say that* c is well defined *if it holds that* A_i *is in the state* $\odot$ *iff* B_i *is in a state reachable after reading* $comm_{A_i}$*. With* $commit(c)$ *we denote the sum of the* A_i *and* B_j *that have committed.*

Given two well defined configurations c_1 *and* c_2*, we write that* $c_1 < c_2$ *if* $commit(c_1) > commit(c_2)$*.*

Given a well defined configuration c*, we prove by induction on the relation* $<$ *that if* $(A, (c, \nu)) \stackrel{w}{\Longrightarrow} (A, (\odot, \nu'))$*, then there exists a proper subset* D *of* $\{i \mid A_i$ *in* c *is not in the state* $\odot\}$ *such that* $w = \tilde{x}_1 \cdot commit_{X_1}! \cdot \ldots \cdot \tilde{x}_m \cdot commit_{X_m}! \cdot \tilde{x}_{m+1}$ *where:*

- *if* $X_i = A_i$*, then* $i \in D$*;*
- *if* $X_i = B_i$ *and* $i \notin D$*, then* A_i *in* c *is in the state* $\odot$*;*
- *for any* $i \in D$ *there exist* k *and* h *in* $[1, m]$ *with* $k < h$ *and such that* $A_i = X_k$ *and* $B_i = X_h$*;*
- *for any* i*,* $\tilde{x}_1, \ldots, \tilde{x}_{m+1} \in (\{\tau\} \cup \mathbb{R}^{>0})^*$*.*

The base case $c = \odot$ *is trivial.*

We now consider the induction step. Let c' *be a configuration reachable from* A *such that* $(A, (c, \nu)) \stackrel{w}{\Longrightarrow} (A, (c', \nu'))$ *and* $w = \tilde{z} \cdot commit_X! \cdot \tilde{z}'$ *with* $\tilde{z}, \tilde{z}' \in (\{\tau\} \cup \mathbb{R}^{>0})^*$*. By Lemma 4.3 and since* c *is well defined, there exists* j *such that one of the following two cases holds:*

- $X = A_j$ *and* A_j *in* c *is not in the state* $\odot$*;*
- $X = B_j$ *and* A_j *in* c *is in the state* $\odot$*.*

In both cases we have that c' *is well defined and* $c' < c$*, hence, by induction, if* $(A, (c', \nu')) \stackrel{w}{\Longrightarrow} (A, (\odot, \nu''))$*, then there exists a proper subset* D' *of* $\{i \mid A_i$ *in* c' *is not in the state* $\odot\}$ *such that* $w = \tilde{y}_1 \cdot commit_{X_1}! \cdot \ldots \cdot \tilde{y}_m \cdot commit_{X_{m'}}! \cdot \tilde{y}_{m'+1}$ *where:*

- *if* $X_i = A_i$*, then* $i \in D'$*;*
- *if* $X_i = B_i$ *and* $i \notin D'$*, then* A_i *in* c' *is in the state* $\odot$*;*
- *for any* $i \in D$ *there exist* k *and* h *in* $[1, m']$ *with* $k < h$ *and such that* $A_i = X_k$ *and* $B_i = X_h$*;*
- *for any* i*,* $\tilde{y}_1, \ldots, \tilde{y}_{m'+1} \in (\{\tau\} \cup \mathbb{R}^{>0})^*$*.*

Therefore, for a fixed $D = D' \cup \{j\}$, and $\tilde{x}_1 = \tilde{z}$ and $\tilde{x}_2 = \tilde{z'} \cdot \tilde{y}_1$ and $\tilde{x}_3 = \tilde{y}_2, \ldots, \tilde{x}_{m+1} = \tilde{y}_{m'+1}$ we have that $(A, (c, \nu)) \stackrel{w}{\Longrightarrow} (A, (\otimes, \nu'))$. Now, since $Init(A)$ is well defined, the thesis holds. □

4.3 Nested Transactions

A nested transaction is composed by a hierarchy of subtransactions. In such a scheme, each subtransaction is executed independently and concurrently with respect to its parent and siblings, and decides autonomously whether to commit or abort. When a transaction aborts, all its subtransactions which already committed must be compensated for. On the other hand, a transaction can commit even if some of its subtransactions have aborted. As we shall see in Section 5, nested transactions can be used, for instance, to represent subactivities which are optional and whose failure may be tolerated.

The pattern for nested transactions is shown in Figure 7. We consider a single activity A with compensation B and with $\{A \Rsh B\}_i$, where i is used to index the nested transaction, we denote the fact that activity A is nested within a transaction. Since the commit or abort of activity A should not affect the global result of the parent transaction, activity A is encapsulated within a schema that reaches a commit state even if activity A aborts.

Actually, in order to correctly compensate a nested transaction (either because the parent transaction aborts, or because the parent transaction, after committing correctly, needs to be compensated), we must keep track of the nested transactions that really committed. Thus, for each nested transaction (with index i), we define a controller automaton that checks whether the transaction aborts or commits, and stores the final state of the transaction by synchronizing on channels *an_i* or *cn_i*, respectively (see Figure 7 (a) and (c)). The compensation of the nested activity is encapsulated in the automaton in Figure 7 (b). If, at some point, the compensation of the nested transaction is invoked, such an automaton is activated and synchronises on channels *con_i* or *abn_i* with the controller automaton in order to decide whether to execute compensation B or not. In the controller automaton, transitions labelled with *stop_i*? are used to stop the activity of the controller allowing it to reach its final commit state at the end of the transaction. Notice that channels *an_i*, *cn_i*, *con_i*, *abn_i* and *stop_i* are assumed to be public (for each nested transaction with index i, $\{a!, a? \mid a \in \{an\_i, cn\_i, con\_i, abn\_i, stop\_i\}\} \subseteq \Sigma_V$).

Definition 4.9 (Nesting) *Given $A \in CHTTA_{\mathcal{A}}^{\Sigma_{Pub}}$ we define the* nesting *of activity A with compensation B as the $CHTTA_{\mathcal{A}}^{\Sigma_{Pub}} A' = [\![\{A \Rsh B\}_i]\!]^N$ depicted in Figure 7 (a). The compound compensation of A' is given by the pair $(B', B^T) = [\![\{A \Rsh B\}_i]\!]_C^N \in (CHTTA_{\mathcal{A}}^{\Sigma_{Pub}})^2$, where B' and B^T are the two CHTTAs in*

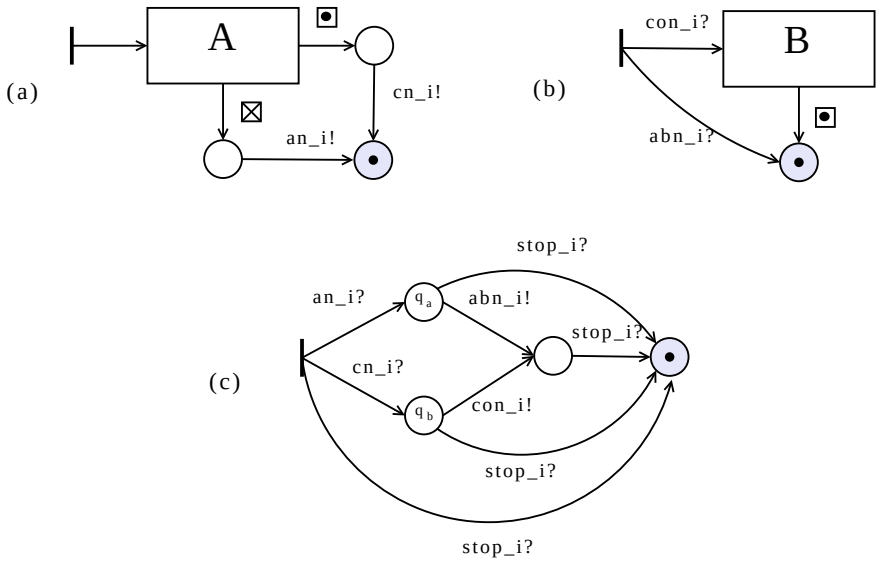

Fig. 7. Pattern for Nested Transactions.

Figures 7 (b) and 7 (c), respectively.

The independent execution of a nested activity implies that activity A may abort or commit without affecting the future behaviour of its parent and siblings. Hence, the pattern of a nested transaction is assumed to end always in a commit state. The next theorem states, trivially, the correct completion of the pattern for nested transactions (a nested transaction always reaches a commit state).

Theorem 4.10 (Correct Completion) *Given the nesting* $A' = [\![\{A \triangleright B\}_i]\!]^N$, *it holds that* $(A', Init(A')) \stackrel{w}{\Longrightarrow} (A', (\odot, \nu))$ *for all* $w \in \mathcal{L}(A', \Sigma_V)$.

PROOF *If activity A reaches the final state* $\otimes$, *then the nested transaction reaches the state* $\odot$ *after a* $\boxtimes$ *step followed by an an_i! step. Similarly, if activity A reaches the final state* $\odot$, *then the nested transaction reaches the state* $\odot$ *after a* $\boxdot$ *step followed by a cn_i! step.* $\square$

The correct compensation for nested transactions is guaranteed by enclosing the compensation activity B within the CHTTA B' and by executing the controller B^T in parallel with the nested transaction $A' = [\![\{A \triangleright B\}_i]\!]^N$. While in Theorem 4.10 we did not need to use the wrapped automata A^M and B^M, in the next theorem we again resort to such a construction. Intuitively, the theorem states that the controller B^T correctly drives the execution of the compensation B if, at some point, the compensation B' is invoked.

Theorem 4.11 (Correct Compensation) *Let* $A' = [\![\{A^M \triangleright B^M\}_i]\!]^N || B^T$, *where* $(B', B^T) = [\![\{A \triangleright B\}_i]\!]^N_C$, *we have that:*

- $(A', Init(A')) \stackrel{w}{\Longrightarrow} (A', (\odot; q_a, \nu))$ *if and only if* $commit_A! \notin w$;

- $(A', Init(A')) \stackrel{w}{\Longrightarrow} (A', (\odot; q_c, \nu))$ *if and only if* $commit_A! \in w$.

Moreover, it holds that:

- $(A'||B', ((\odot; q_a, \nu); Init(B'))) \stackrel{w'}{\Longrightarrow} (A'||B', (\odot, \nu'))$ *if and only if* $commit_B! \notin w'$;
- $(A'||B', ((\odot; q_c, \nu); Init(B'))) \stackrel{w'}{\Longrightarrow} (A'||B', (\odot, \nu'))$ *if and only if* $commit_B! \in w$.

PROOF *By Lemma 4.3,* A^M *reaches the state* $\odot$ *if and only if* $commit_A! \in w$. *Moreover, if* A^M *reaches the final state* $\odot$, *then the nested transaction reaches the state* $\odot$ *after a* $\boxdot$ *step and a* $cn\_i!$ *step. Hence,* B^T *synchronises on channel* cn_i *and reaches the state* q_c *if and only if* $commit_A! \in w$. *Similarly, if* A^M *reaches the final state* $\otimes$ *(in this case* $commit_A! \notin w$*), then the nested transaction reaches the state* $\odot$ *after a* $\boxtimes$ *step and an* $an\_i!$ *step. Hence,* B^T *synchronises on channel* an_i *and reaches the state* q_a *if and only if* $commit_A! \notin w$.

Now, if B^T *is in state* q_c, *then* $A'||B'$ *can synchronise only on channel* con_i *and, as a consequence, the compensation* B *is activated (implying* $commit_B! \in w'$*). Therefore, if* B^T *passes through state* q_c, *then* $commit_B! \in w'$. *Similarly, if* B^T *is in state* q_a, *then* $A'||B'$ *can synchronise only on channel* abn_i *and, as a consequence, the compensation* B *is not activated (thus* $commit_B! \notin w'$*). Therefore,if* B^T *passes through state* q_a *then* $commit_B! \notin w'$. $\square$

4.4 Long–Running Transactions

Sequential, parallel and nested transactions may be composed in order to define complex transactions. Hence, resorting to the patterns defined in the previous sections, we give the definition of long–running transactions.

Definition 4.12 (Long–Running Transaction) *Let* $A_1, \ldots, A_n$ *be activities in* $CHTTA_{\mathcal{A}}^{\Sigma_{Pub}}$ *with compensations* $B_1, \ldots, B_n \in CHTTA_{\mathcal{A}}^{\Sigma_{Pub}}$; *a* long–running transaction *is defined by the following grammar:*

$$T \ ::= \ A_i \vec{r} B_i \quad | \quad T \cdot T \quad | \quad T||T \quad | \quad \{T\}_i.$$

Again, we assume that nested transactions are labelled with an index $i \in \mathbb{N}$. Moreover, given a transaction T, we assume that for any pair of nested subtransactions $\{T'\}_i$, $\{T''\}_j$ of T it holds that $i \neq j$.

Now, we need to introduce an *encoding* function $[\![\cdot]\!] : T \rightarrow \text{CHTTA}_{\mathcal{A}}^{\Sigma_{Pub}} \times \text{CHTTA}_{\mathcal{A}}^{\Sigma_{Pub}} \times \mathcal{P}(\text{CHTTA}_{\mathcal{A}}^{\Sigma_{Pub}})$, such that $[\![T]\!] = (A, B, M)$, where the CHTTAs A is the compound CHTTA modelling the transaction T, B its compen-

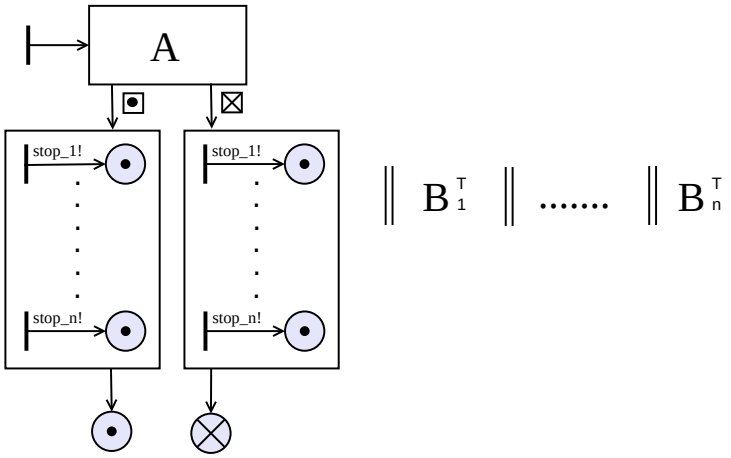

Fig. 8. Top–level CHTTA.

sation and M is the set of CHTTAs modelling the controllers B_i^T of the nested transactions contained within T.

Definition 4.13 (Encoding) *We recursively define the* encoding *function* $[\![\cdot]\!] : T \to CHTTA_{\mathcal{A}}^{\Sigma_{Pub}} \times CHTTA_{\mathcal{A}}^{\Sigma_{Pub}} \times \mathcal{P}(CHTTA_{\mathcal{A}}^{\Sigma_{Pub}})$ *as follows:*

- $[\![A_i \curvearrowright B_i]\!] = (A_i, B_i, \emptyset)$,
- $[\![T_1 \cdot T_2]\!] = ([\![A_1 \curvearrowright B_1 \cdot A_2 \curvearrowright B_2]\!]^S, [\![A_1 \curvearrowright B_1 \cdot A_2 \curvearrowright B_2]\!]_C^S, M_1 \cup M_2)$,
  where $(A_i, B_i, M_i) = [\![T_i]\!]$ *for* $i \in [1, 2]$,
- $[\![T_1 || T_2]\!] = ([\![A_1 \curvearrowright B_1 || A_2 \curvearrowright B_2]\!]^P, [\![A_1 \curvearrowright B_1 || A_2 \curvearrowright B_2]\!]_C^P, M_1 \cup M_2)$,
  where $(A_i, B_i, M_i) = [\![T_i]\!]$ *for* $i \in [1, 2]$.
- $[\![\{T\}_i]\!] = ([\![\{A \curvearrowright B\}_i]\!]^N, B', B^T \cup M)$,
  where $(A, B, M) = [\![T]\!]$ *and* $[\![\{A \curvearrowright B\}_i]\!]_C^N = (B', B^T)$.

The root of the hierarchy of a transaction is usually referred to as *top–level*. Here, given the encoding of a transaction T as $[\![T]\!] = (A, B, M)$, we should pay attention to the controllers $B_i^T \in M$ of nested subtransactions. In particular, each controller B_i^T must be put in parallel with the CHTTA A modelling the whole transaction. Moreover, at the end of the execution of A, we need to send the stop signals to all these controllers in order to let them reach their final commit states (see Figure 7 (c)).

Definition 4.14 (Top–Level) *Given a long–running transaction* T *and its encoding* $[\![T]\!] = (A, B, M)$ *with* $M = \{B_1^T, B_2^T, \ldots, B_n^T\}$, *we define the* top–level *of* T *(denoted top*(T)*) as the CHTTA in Figure 8.*

Given a transaction T, since the building blocks of the encoding function are the patterns of sequential, parallel and nested transactions, the correct completion and correct compensation of the top–level CHTTA *top*(T) is given by induction on the structure of T and by Theorems 4.4, 4.5, 4.7, 4.8, 4.10, 4.11.

Example 4.15 *Given activities* $A_1, A_2, A_3 \in CHTTA_{\mathcal{A}}^{\Sigma_{Pub}}$ *and compensations*

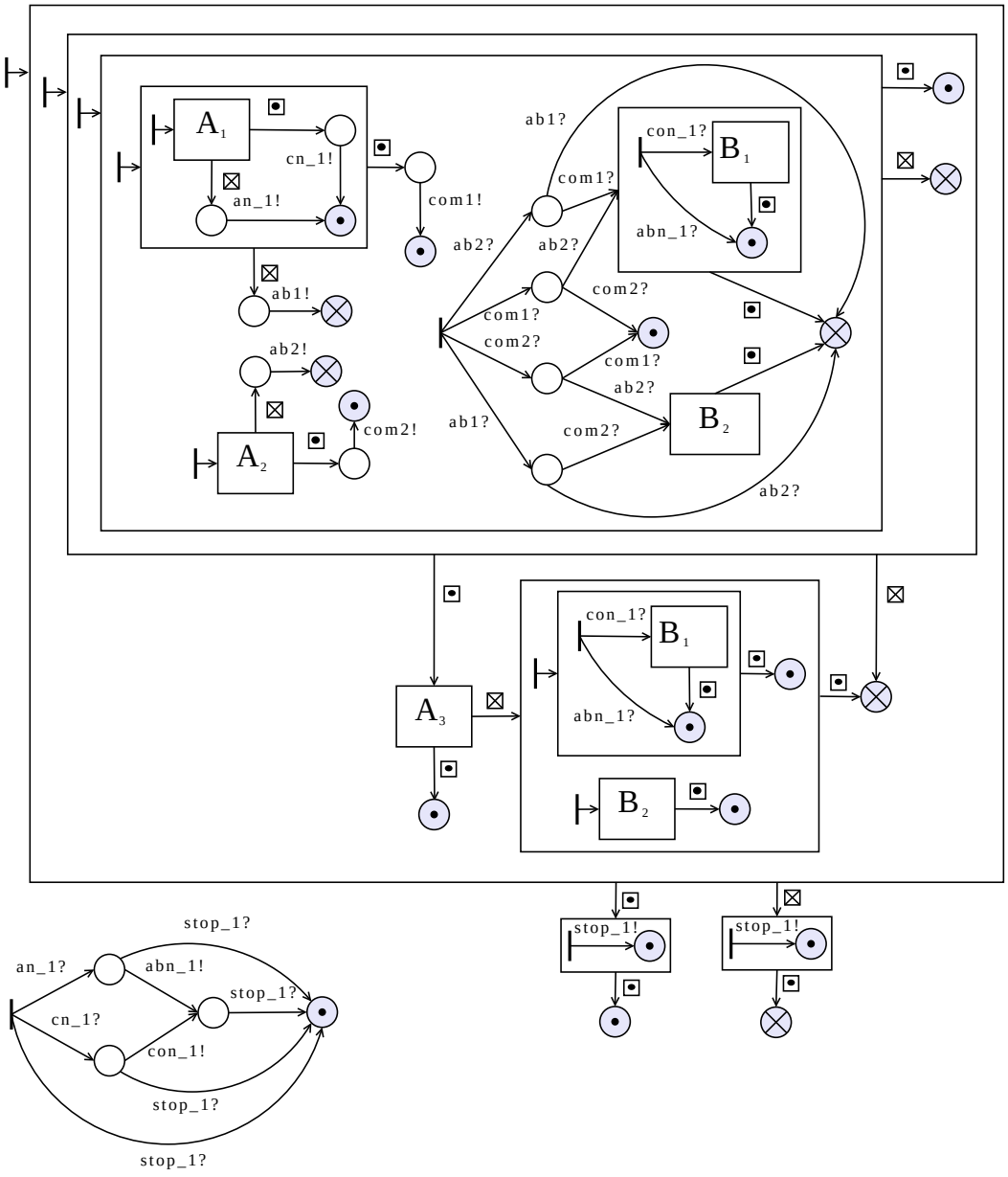

Fig. 9. Example of a Top–level CHTTA.

$B_1, B_2, B_3 \in CHTTA_{\mathcal{A}}^{\Sigma_{Pub}}$, *an example of* long–running transaction *is:*

$$T = (\{A_1 \triangleright B_1\}_1 || A_2 \triangleright B_2) \cdot A_3 \triangleright B_3.$$

The CHTTA $top(T)$ *modelling the top–level of* T *is shown in Figure 9.*

Modelling long–running transactions with CHTTAs allows verifying properties by model checking.

In fact, given a long–running transaction T obtained as in Definition 4.12, and a set of visible actions Σ_V, we may flatten the CHTTA $top(T)$ according to

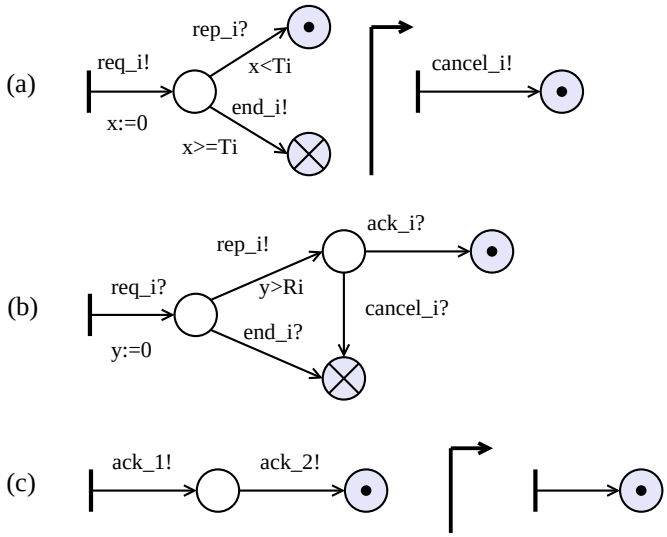

Fig. 10. A Double Request.

Definition 3.1, and then verify properties of the transaction by model checking the timed automaton $Flat(top(T), \Sigma_V)$.

5 Case Study: A Double Request

We model a typical all–or–nothing scenario in which a client performs two concurrent requests to two different servers, waits for replies, and sends back acknowledgements either to both servers (if it receives both replies) or to none of them (if it receives at most one reply). A similar scenario in a realistic context is given in [25], where a typical e–commerce application is described in which a customer of an on–line shop orders two products which are provided by two different stores. In that case, acknowledgements are sent (and products are bought) only if both products are available, instead, in our case, acknowledgements are sent only if replies are received before given times.

A single request/reply activity performed by the client is described by the transaction given in Figure 10 (a). We denote the transaction as $A_i \Gamma B_i$. The client sends the request to the server by synchronizing on channel req_i and waits for the reply to be received as a synchronisation on channel rep_i. The time deadline for the reply is T_i. This is expressed as a constraint on the value of clock x_i which is set to zero when the request is sent. If the reply is received in time, the transaction commits, otherwise a stop message is sent to the server as a synchronisation on channel end_i, and the transaction is aborted. The compensation of this transaction consists in a synchronisation on channel $cancel\_i$, which corresponds to sending an undo message to the server.

A server is modelled by the automaton given in Figure 10 (b). We denote such an automaton with S_i. The server receives a request and sends the reply by synchronizing on the proper channels, and it spends a time between these two synchronisations which is greater than R_i. This amount of time models the time spent by the server to satisfy the request of the client. Then, the server reaches a state in which it waits for either an acknowledge or an undo message from the client. These two communications are modelled as synchronisations on channels ack_i and $cancel\_i$, and lead to the commit or the abort of the server activity, respectively.

The activity of sending acknowledgments to two servers S_1 and S_2 is modelled by the transaction given in Figure 10 (c). We denote this transaction with $A_{ack} \Rsh B_{ack}$. Finally, the whole client transaction in which two requests are sent to two different servers and the corresponding acknowledgments are sent if both requests are satisfied, is modelled by the long–running transaction $T = (A_1 \Rsh B_1 || A_2 \Rsh B_2) \cdot A_{ack} \Rsh B_{ack}$ and the whole system in which both the client and the two servers are modelled is $SYSTEM = T||S_1||S_2$.

To verify properties of this system, we consider the CHTTA $top(T)$, and we manually compute the flat TTA $T' = Flat(top(T), \Sigma_V)$, where $\Sigma_V = \{a!, a? \mid a \in \{req_i, rep_i, stop_i, cancel_i, ack_i\}\}$ by following the procedure given in Section 3.1. Now, since S_1 and S_2 are both flat, we have that $T'||S_1||S_2$ can be used as an input for the UPPAAL model checker, which is able to manage parallel composition of flat timed automata. In Figure 11 we show the UPPAAL model for the two server transactions. In Figure 12 we show the UPPAAL timed automaton for the client transaction which is obtained from the flat TTA T' and where, in order to reduce the size of the model, we have removed unnecessary τ transitions. We have also added to this model the invariant $x_i \leq T_i$ in the state where the client is waiting for the server reply. This implies that the client will stop its activity as soon as the deadline T_i is reached. Moreover, we exploited the UPPAAL feature to define some states as *urgent*. Intuitively, time is not allowed to pass when the system is in an urgent state. Note that using urgent states is semantically equivalent to adding an extra clock z, that is reset on all incoming edges, and having an invariant $z \leq 0$ on the state. Resorting to urgent states allows us to avoid the execution of paths containing an infinite sequence of timed transitions, and to deal only with maximal paths ending in the commit or in the abort state. These maximal paths are exactly the ones which correspond to the strings in $\mathcal{L}(top(T), \Sigma_V)$.

In Figure 13 we show the results of the model checking. We have verified eight properties, and each property has been verified three times: once by setting both timeouts T_1 and T_2 greater than R_1 and R_2, respectively, once by setting $T_1 < R_1$ and $T_2 > R_2$, and once by setting both T_1 and T_2 smaller than R_1 and R_2, respectively.

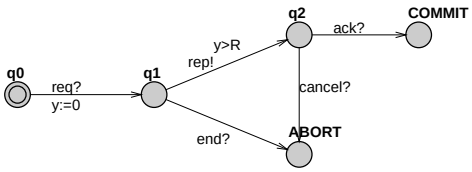

Fig. 11. Timed Automaton Modelling a Server.

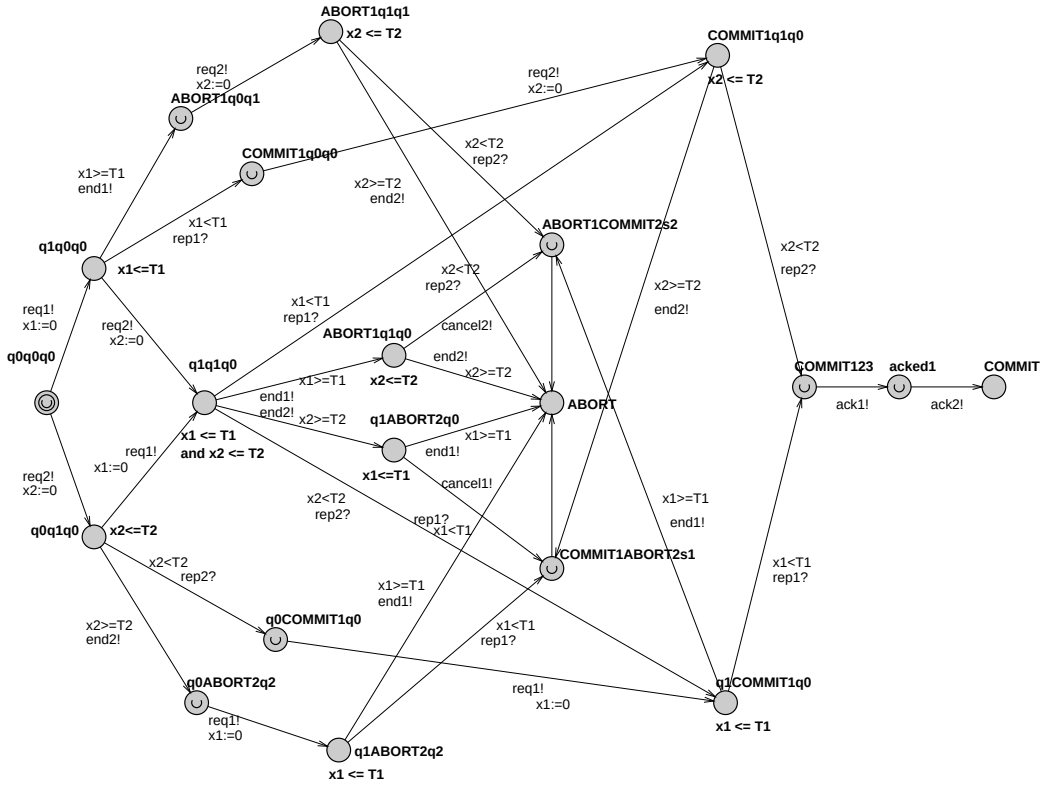

Fig. 12. Timed Automaton Modelling the Client Transaction.

Properties are expressed as logical formulas using the operators accepted by the UPPAAL model checker. A logical formula may have one of the following forms: $E\Diamond\phi$, $E\Box\phi$, $A\Diamond\phi$, $A\Box\phi$ and $\phi \rightsquigarrow \psi$, where ϕ and ψ are state formulas, namely conditions which could be satisfied by a state. In particular: $E\Diamond\phi$ represents reachability: it asks whether ϕ is satisfied by some reachable state; $E\Box\phi$ says that there should exists a maximal path such that ϕ is always true; $A\Diamond\phi$ says that ϕ is eventually satisfied in all paths; $A\Box\phi$ expresses that ϕ should be true in all reachable states; finally, $\phi \rightsquigarrow \psi$ means that whenever ϕ is satisfied, then eventually (in the continuation of the path) ψ will be satisfied.

Properties 1–3 express the correctness of the encoding of long–running transactions into automata. These properties must be satisfied for any setting of the parameters. In particular, property 1 says that either the commit or the abort states of the transaction (denoted $T.\odot$ and $T.\otimes$, respectively) must be eventually reached. Property 2 requires that if at least one of the abort states

	$T_1 > R_1$ $T_2 > R_2$	$T_1 < R_1$ $T_2 > R_2$	$T_1 < R_1$ $T_2 < R_2$
1. $A\Diamond(T.\odot \vee T.\otimes)$	true	true	true
2. $(A_1.\otimes \vee A_2.\otimes) \rightsquigarrow T.\otimes$	true	true	true
3. $(A_1.\odot \wedge A_2.\odot) \rightsquigarrow T.\odot$	true	true	true
4. $T.\odot \rightsquigarrow (S_1.\odot \wedge S_1.\odot)$	true	true	true
5. $x_1 \geq T_1 \rightsquigarrow T.\otimes$	true	true	true
6. $x_2 \geq T_2 \rightsquigarrow T.\otimes$	true	true	true
7. $E\Diamond T.\odot$	true	false	false
8. $E\Diamond T.\otimes$	true	true	true

Fig. 13. Results of the model checking.

of the parallel activities A_1 and A_2 is reached, then the whole transaction must reach its abort state, and property 3 requires that if both parallel activities A_1 and A_2 reach their commit states, then the whole transaction must reach its commit state.

Properties 4–6 express the correctness of the scenario we are modelling. As before, these properties must be satisfied for any setting of the parameters. Property 4 says that if the transaction reaches a commit state, then eventually both servers must reach their commit states. Properties 5 and 6 say that if one of the two clocks of the parallel activities A_1 and A_2 becomes greater than its deadline, then the whole transaction must reach its abort state.

Finally, properties 7 and 8 express that the commit and abort states of the transaction can be reached, for different settings of the parameters. In particular, the commit state can be reached only if both the timeouts T_1 and T_2 are greater than the times R_1 and R_2 spent by the two servers. The abort state, instead, can be reached with any setting of the parameters. This is true because R_1 and R_2 are lower bounds, hence a server may spend more time than its minimum time, and may exceed the corresponding deadline in the transaction.

A variant of this scenario is the one in which one of the two concurrent client requests should not necessarily be satisfied for the commitment of the whole transaction. Consider, for instance, a double request in which a client orders two products, one of the products is fundamental, and the other is optional (for instance a laptop computer and a mouse). In this case, the client might accept to buy the former product even if the latter is not available.

Such a scenario can be modelled by nesting the activity requesting the optional product. Thus, if $A_F \Rsh B_F$ is the activity modelling the request for the fundamental product and $A_O \Rsh B_O$ the activity modelling the request for the optional product, the client double request may be represented as $T' = (A_F \Rsh B_F || \{A_O \Rsh B_O\}_1) \cdot A'_{Ack} \Rsh B'_{Ack}$. Note that the top–level scheme of such

a transaction is the same as the one in Figure 9 (where $A_1 \Gamma B_1$ corresponds to $A_O \Gamma B_O$, $A_2 \Gamma B_2$ corresponds to $A_F \Gamma B_F$ and $A_3 \Gamma B_3$ corresponds to $A'_{Ack} \Gamma B'_{Ack}$). Again, our flattening procedure may be used to obtain the timed automaton $Flat(top(T'), \Sigma_V)$ and to verify properties of the transaction in this different scenario.

6 Related Work

The model of CHTTAs is obtained by extending existing models of Communicating Hierarchical Timed Automata in order to deal with the commit and abort of transactions, and with a communication mechanism which makes use of public channels. Among the papers dealing with hierarchical composition and time we cite the following.

In [16–18] David et al. present a framework, called Hierarchical Timed Automata (HTAs), for the formal verification of a real-time extension of UML statecharts. In particular, the authors extend a reasonable subset of the rich UML statechart model with real–time constructs such as clocks, timed guards, and invariants. A translation of HTAs into networks of flat timed automata is given and used to produce the input to the real–time model checking tool UPPAAL.

Among the papers for the analysis of compositional patterns we cite [10]. In this paper, Bruni et al. present a family of transactional process calculi with increasing expressiveness. Starting from a very small language in which activities are composed only sequentially, the authors progressively introduce a parallel composition operator, a nesting operator, programmable compensations and an exception handling mechanism. We borrow from [10] the patterns for composing transactions and we describe them by means of CHTTAs.

In [28] a kernel of the orchestration language BPEL containing the constructs for the description of long–running transactions is considered and endowed with an operational semantics. In [7] an extension of the π–calculus is introduced with a mechanism to describe long–running transactions which is inspired by BPEL. In [6] the asynchronous π–calculus is used to formally model a compositional protocol in charge of activating compensations of nested long–running transactions. In [13] the language StAC for describing long–running transactions is introduced with a notation inspired by the formalisms CSP and CCS. A formal semantics for StAC is given in [14]. In [15] a model of long–running transactions in the framework of the CSP process algebra is proposed. With respect to all the above mentioned papers our formalism allows time to be described and models that can be translated into Timed Automata on which automatic verification can be performed.

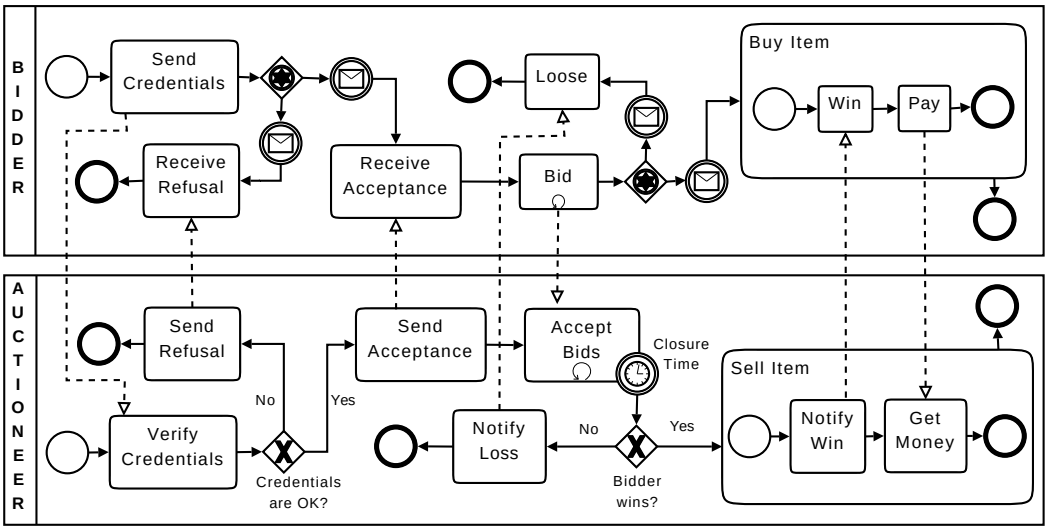

Fig. 14. Example of a business process described in BPMN.

In [22] and [23] the π–calculus is extended with a mechanism to handle compensations and with a notion of discrete time. Timers can be associated with processes and compensations are activated either after abort or after a time–out. On the contrary, in our model we have continuous time and any activity may depend on time.

YAWL [1] is a workflow language with a well defined formal semantics that implements the most common workflow patterns. In [9] a formal semantics for BPEL is given through a methodology for translating BPEL processes into YAWL workflows, paving the way for the formal analysis, aggregation and adaptation of BPEL processes. The approach defines a YAWL pattern for each BPEL activity and gives suitable instantiations and interconnections among the patterns.

As regards verification of service orchestration we cite [29] and [21] where BPEL workflows are expressed by means of process calculi and Timed Automata, respectively, and are verified by model checking. The papers mentioned do not consider the BPEL fragment dealing with long running transactions.

We conclude by showing how an example of business process described by means of the graphical notation BPMN [12] can be modeled with CHTTAs. The example in Figure 14 consists of two processes representing an auctioneer and a bidder. Each process is a flowchart where solid arrows represent sequence flows and dashed arrows represent message flows. Boxes represent activities, diamonds represent gateways, and circles represent events. The bidder sends its credentials to the auctioneer, which verifies them and may send either an acceptance or a refusal message back to the bidder. Note that the gateway in the auctioneer process is *data–based*, whereas the gateway in the bidder

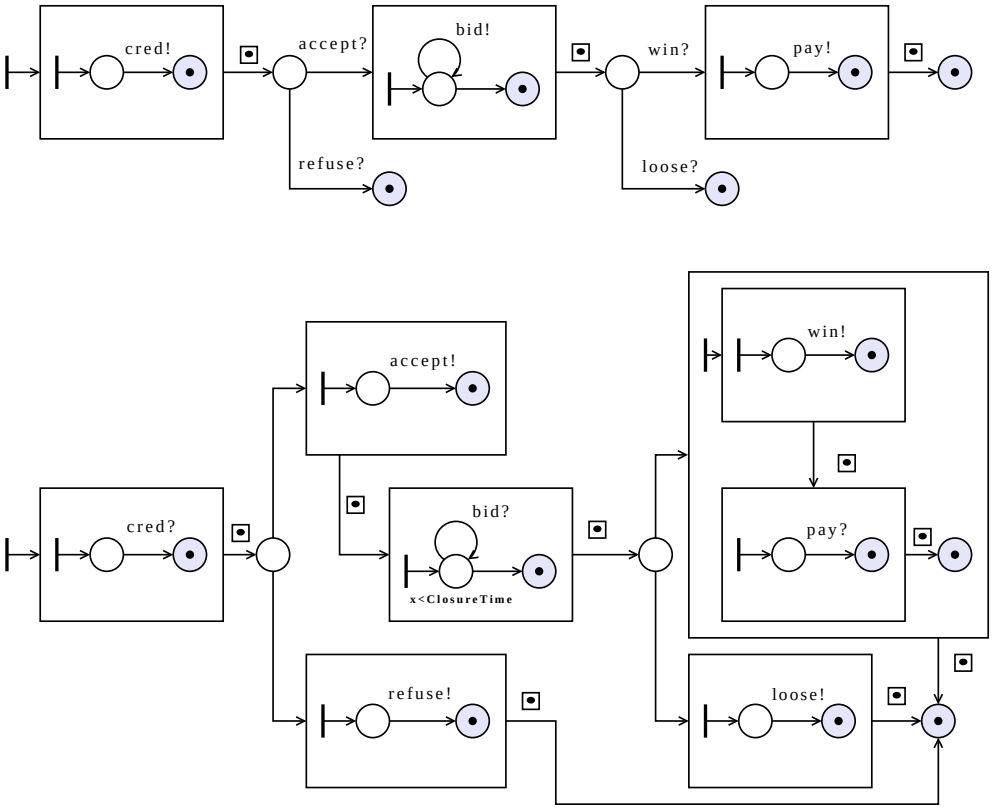

Fig. 15. Translation into CHTTAs of the business process in Figure 14.

process is *event–based*, namely in this case the choice depends on the received message. In case of refusal of the bidder's credentials both processes terminate. If the bidder's credentials are accepted the bidder starts bidding repeatedly (represented by the symbol ↻ in the activity) and the auctioneer accepts bids until a timer event, representing the auction closure, occurs. If a bidder has won, the auctioneer notifies the win and waits for the payment, otherwise, it notifies the loss to the bidder. Note that win notification and payment reception are subactivities of a compound activity; the same happens for the corresponding activities performed by the bidder.

In Figure 15 we show a translation of the BPMN processes in Figure 14 into the parallel composition of two CHTTAs. Every BPMN activity is translated into a superstate, every gateway becomes a state, every sequence flow arrow becomes a transition, and every message flow arrow becomes a pair of communication actions. Note that event–based gateways are translated into states whose outgoing transitions are labeled with the first action of each of the activities under choice. Activities which can be repeated give rise to loops in the automata. Timer events are translated into constraints on clocks. The example should intuitively suggest how BPMN diagrams could be translated into CHTTAs also in a more general case.

7 Conclusions

In this paper we have developed the model of CHTTAs suitable to describe long–running transactions in a timed setting.

Following the approach in [10], we have identified some composition patterns for transactional activities with compensations. In [10], a big–step semantics is given which, assuming that basic activities either commit or abort, gives the outcome of a long–running transaction. In the present paper, basic activities are described as CHTTAs, and therefore their behaviour can be modelled in detail.

The present paper has been particularly focused on the design of compensation handling mechanisms. There are, however, other aspects which could be considered in the design of long–running transactions. For example, transactions usually support fault tolerance for recovery from both internal faults (i.e., machine failures or software faults) and external faults (i.e., termination messages). As already stressed, partial updates within a long–running transaction could not be rolled back automatically, as they are in ACID transactions, when a failure occurs. However, an exception code block for the long–running transaction could be invoked when a fault occurs. The exception code block may contain a set of fault handlers to deal with any of the faults that can arise during the execution of the transaction. Following an approach similar to the one used to deal with compensations, we believe that our framework could be extended with a fault termination handling mechanism. Assuming that the execution of a long–running transaction is interrupted when a fault occurs, we may resort to some special transitions (taking a role similar to the one of $\boxdot$ and $\boxtimes$) invoking an exception handler at the time an exception occurs. The exception handler could then activate the activities necessary to manage the fault. Note that this extension might influence the proven decidability of reachability for CHTTAs. Actually, the fault handling may require the atomic interruption of several activities running in parallel, and this may cause the increase of expressiveness of the formalism, with all possible consequences. We leave this problem for further investigation.

Our main goal was to lay the foundations for the formal verification of long-running transactions in a timed setting. This might be eventually done by constructing a translator from BPMN or BPEL to CHTTAs, and implementing the flattening procedure to obtain the input expected by a model checker for timed automata, such as UPPAAL.

References

[1] W. M. P. van der Aalst and A. H. M. ter Hofstede, YAWL: Yet Another Workflow Language, Information Systems 30 (2005) 245–275.

[2] R. Alur and D. L. Dill, A Theory of Timed Automata, Theoretical Computer Science 126 (1994) 183–235.

[3] R. Alur, S. Kannan, and M. Yannakakis, Communicating Hierarchical State Machines, in: Proc. ICALP'99, Lecture Notes in Computer Science, Vol. 1644 (Springer, 1999) 169–178.

[4] T. Amnell, G. Behrmann, J. Bengtsson, P. R. D'Argenio, A. David, A. Fehnker, T. Hune, B. Jeannet, K. G. Larsen, M. O. Moeller, P. Pettersson, C. Weise, and W. Yi, Uppaal–now, next and future, Lecture Notes in Computer Science, Vol. 2067 (Springer, 2000) 99–124.

[5] B. Benatallah and R. Himadi, A Petri Net–Based Model for Web Service Composition, in: Proc. ADC'03 (Australian Computer Society, 2003) 191–200.

[6] L. Bocchi, Compositional Nested Long Running Transacitions, Fundamental Approaches to Software Engineering, Lecture Notes in Computer Science, Vol. 2984 (Springer, 2004) 194–208.

[7] L. Bocchi, C. Laneve, and G. Zavattaro, A Calculus of Long Running Transactions, in: Proc. FMOODS'06, Lecture Notes in Computer Science, Vol. 4037 (Springer, 2006) 124–138.

[8] A. Brogi, C. Canal, E. Pimentel, and A. Vallecillo, Formalizing Web Services Choreographies, in: Proc. WS–FM'04, Electronic Notes in Theoretical Computer Science, Vol. 105 (Elsevier, 2004) 73–94.

[9] A. Brogi and R. Popescu, From BPEL Processes to YAWL Workflows, in: Proc. WS–FM'06, Lecture Notes in Computer Science, Vol. 4184 (Springer, 2006) 107–122.

[10] R. Bruni, H. Melgratti, and U. Montanari, Theoretical Foundations for Compensations in Flow Composition Languages, in: Proc. POPL'05 (ACM Press, 2005) 209–220.

[11] BPEL specifications:
`http://www-128.ibm.com/developerworks/library/ws-bpel/`.

[12] BPMN web page: `http://www.bpmn.org`.

[13] M. J. Butler and C. Ferreira, A Process Compensation Language, in: Proc. IFM'00, Lecture Notes in Computer Science, Vol. 1945 (Springer, 2000) 61–76.

[14] M. J. Butler and C. Ferreira, An Operational Semantics for StAC, a Language for Modelling Long–Running Business Transactions, in: Proc. COORDINATION'04, Lecture Notes in Computer Science, Vol. 2949 (Springer, 2004) 87–104.

[15] M. J. Butler, C. A. R. Hoare, and C. Ferreira, A Trace Semantics for Long-Running Transactions, in: 25 Years Communicating Sequential Processes, Lecture Notes in Computer Science, Vol. 3525 (Springer, 2004) 133–150.

[16] A. David, Hierarchical Modeling and Analysis of Timed Systems, Ph.D. Thesis, Uppsala: Mathematics and Computer Science, Department of Information Technology, 2003.

[17] A. David, and O. Moeller, From HUPPAAL to UPPAAL: A Translation from Hierarchical Timed Automata to Flat Timed Automata, BRICS Technical Report Series RS–01–11, 2001.

[18] A. David, M. O. Moeller, and W. Yi, Formal Verification of UML Statecharts with Real-Time Extensions, in: Proc. FASE'02, Lecture Notes in Computer Science, Vol. 2306 (Springer, 2002) 218–232.

[19] H. Garcia–Molina and K. Salem, Sagas, in: Proc. SIGMOD'87 (ACM Press, 1987) 249–259.

[20] I. Houston, M.C. Little, I. Robinson, S. K. Shrivastava, and S. M. Wheater, The CORBA Activity Service Framework for Supporting Extended Transactions, Software – Practice and Experience 33 (2003) 351–373.

[21] M. Koshkina and F. van Breugel, Modelling and Verifying Web Service Orchestration by Means of the Concurrency Workbench, ACM SIGSOFT Software Engineering Notes 29 (2004) 1–10.

[22] C. Laneve and G. Zavattaro, Foundations of Web Transactions, in: Proc. FOSSACS'05, Lecture Notes in Computer Science, Vol. 3441 (Springer, 2005) 282–298.

[23] C. Laneve and G. Zavattaro, web–pi at Work, in: Proc. TGC'05, Lecture Notes in Computer Science, Vol. 3705 (Springer, 2005) 182–194.

[24] R. Lanotte, A. Maggiolo-Schettini, P. Milazzo and A. Troina, Modeling Long-Running Transactions with Communicating Hierarchical Timed Automata, in: Proc. FMOODS'06, Lecture Notes in Computer Science, Vol. 4037 (Springer, 2006) 108–122.

[25] M. Mazzara and S. Govoni, A Case Study of Web Services Orchestration, in: Proc. COORDINATION'05, Lecture Notes in Computer Science, Vol. 3454 (Springer, 2005) 1–16.

[26] M. Mazzara and R. Lucchi, A pi-calculus based semantics for WS-BPEL, Journal of Logic and Algebraic Programming 70 (2006) 96–118.

[27] G. Plotkin, A structured approach to operational semantics, Technical Report DAIMI FM–19, Department of Computer Science, Aarhus University, Denmark, 1981.

[28] G. Pu, H. Zhu, Z. Qiu, S. Wang, X. Zhao and J. He, Theoretical Foundation of Scope–based Compensable Flow Language for Web Service, in: Proc. FMOODS'06, Lecture Notes in Computer Science, Vol. 4037 (Springer, 2006) 251–266.

[29] G. Pu, X. Zhao, S. Wang and Z. Qiu, Towards the Semantics and Verification of BPEL4WS, in: Proc. WS–FM'05, Electronic Notes in Theoretical Computer Science, Vol. 151 (Elsevier, 2005) 33–52.

[30] WSCI Specification, version 1.0:
`http://www.w3.org/TR/wsci/`.

[31] M. Viroli, Towards a Formal Foundation to Orchestration Languages, in: Proc. WS–FM'04, Electronic Notes in Theoretical Computer Science, Vol. 105 (Elsevier, 2004) 51–71.

[32] S. Yovine, Kronos: A verification tool for real-time systems, International Journal on Software Tools for Technology Transfer 1 (1997) 123–133.