Towards Transactional Pervasive Workflows

Frederic Montagut
SAP Labs France, Institut Eurecom
805, Avenue du Docteur Maurice Donat
Font de l'Orme, 06250 Mougins, France
frederic.montagut@sap.com

Refik Molva
Institut Eurecom
2229 Route des Cretes
06904 Sophia-Antipolis, France
refik.molva@eurecom.fr

Abstract

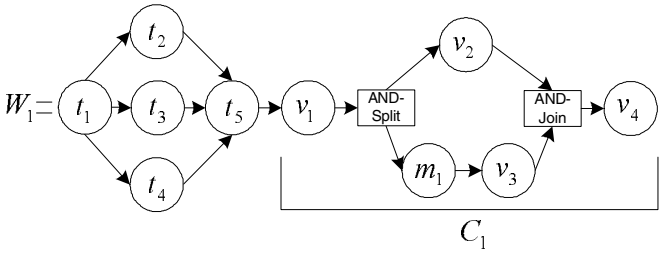
Workflow technologies are becoming pervasive in that they enable the execution of business processes in distributed and ubiquitous computing environments. The execution of pervasive workflows raises transactional requirements due to their dynamicity and the need for relaxed atomicity. In this paper, we propose an adaptive transactional protocol for the pervasive workflow model developed in a previous work. The execution of this protocol takes place in two phases. First devices are assigned to tasks using an algorithm whereby workflow partners are selected based on functional and transactional requirements. The workflow execution further proceeds through a hierarchical coordination protocol managed by the workflow initiator and controlled based on a decision table computed as an outcome of the device assignment procedure. The resulting workflow execution is compliant with the defined consistency requirements and the coordination decisions depend on the transactional characteristics of the devices assigned to each task.

1. Introduction

Workflow technologies are becoming pervasive in that they enable the execution of business processes in distributed and ubiquitous environments [20], [2], [5]. The adequate execution support for pervasive workflows has to cope with the lack of dedicated infrastructure for management and control tasks. To that effect a first step has been achieved by the design of a fully decentralized workflow architecture built on top of the "Enterprise Services Architecture" paradigm [18]. Featuring a dynamic assignment of tasks to workflow partners, this architecture allows users to initiate workflows in any environments where surrounding users' resources can be advertised by various means including a discovery service. Yet, this architecture does not give any guarantee on the consistency of data modified during the process execution. Considering the lack of reliability of the devices present in distributed environments, data and transaction consistency is a main issue. Transactional requirements raised by pervasive workflows are twofold: on the one hand, the workflow execution is dynamic in that the workflow partners offering different characteristics can be assigned to tasks at runtime and atomicity of the workflow execution can be relaxed as intermediate results produced by the workflow may be kept despite the failure of one partner. Existing transactional protocols [8, 10] are not adapted to solve this paradigm as they do not offer enough flexibility to cope for instance with the runtime assignment of computational tasks.

In this paper, we propose an adaptive transactional protocol for the pervasive workflow model developed in [18]. The execution of this protocol takes place in two phases. First, devices are assigned to tasks using an algorithm whereby workflow partners are selected based on functional and transactional requirements. These transactional requirements are defined at the workflow design stage using the Acceptable Termination States model (*ATS*). The workflow execution further proceeds through a hierarchical coordination protocol managed by the workflow initiator and controlled using a decision table computed as an outcome of the device assignment procedure. The resulting workflow execution is compliant with the defined consistency requirements and the coordination decisions depend on the characteristics of the devices assigned to each task.

The paper is organized as follows. Section 2 introduces preliminary definitions and the methodology followed by our approach. We present an example of pervasive workflow in section 3 for the purpose of illustrating our results throughout the paper. Section 4 is a detailed description of the transactional model used to represent the characteristics offered by devices. Section 5 describes how an *ATS* is derived from the properties of the termination states. Section 6 and section 7 present the transaction-aware device assignment procedure and the associated coordination protocol respectively. Section 8 discusses related work and section 9



TS(C_1)	v_1	v_2	m_1	v_3	v_4
ts_1	completed	completed	completed	completed	completed
ts_2	completed	completed	completed	completed	failed
ts_3	completed	completed	completed	compensated	failed
ts_4	completed	compensated	completed	completed	failed
ts_5	completed	compensated	completed	compensated	failed
ts_6	compensated	completed	completed	completed	failed
ts_7	compensated	completed	completed	compensated	failed
ts_8	compensated	compensated	completed	completed	failed
ts_9	compensated	compensated	completed	compensated	failed
ts_{10}	completed	completed	completed	failed	aborted
ts_{11}	completed	compensated	completed	failed	aborted
ts_{12}	completed	canceled	completed	failed	aborted
ts_{13}	compensated	completed	completed	failed	aborted
ts_{14}	compensated	compensated	completed	failed	aborted
ts_{15}	compensated	canceled	completed	failed	aborted
ts_{16}	completed	completed	hfailed	aborted	aborted
ts_{17}	completed	compensated	hfailed	aborted	aborted
ts_{18}	completed	canceled	hfailed	aborted	aborted
ts_{19}	compensated	completed	hfailed	aborted	aborted
ts_{20}	compensated	compensated	hfailed	aborted	aborted
ts_{21}	compensated	canceled	hfailed	aborted	aborted
ts_{22}	completed	failed	completed	aborted	aborted
ts_{23}	completed	failed	canceled	aborted	aborted
ts_{24}	compensated	failed	completed	aborted	aborted
ts_{25}	compensated	failed	canceled	aborted	aborted
ts_{26}	completed	failed	completed	completed	aborted
ts_{27}	completed	failed	completed	compensated	aborted
ts_{28}	completed	failed	completed	canceled	aborted
ts_{29}	compensated	failed	completed	completed	aborted
ts_{30}	compensated	failed	completed	compensated	aborted
ts_{31}	compensated	failed	completed	canceled	aborted
ts_{32}	failed	aborted	aborted	aborted	aborted

Figure 1. Deal at a fair

presents concluding remarks.

2. Definitions and goals statement

Defining a transactional protocol for pervasive workflows raises challenges that are mainly due to the flexibility of their execution and their lack of dedicated infrastructure. After a reminder of the features offered by the pervasive workflow architecture, we give a definition of the transactional properties that must be fulfilled throughout the execution of pervasive workflows.

2.1. Pervasive workflows

In this section, we present the model of pervasive workflow that was designed in [18]. The pervasive workflow concept introduces a workflow management system supporting the execution of business processes in environments wherein devices or services can expose their resources to their surrounding environment. This workflow management system features a distributed architecture characterized by two objectives:

- fully decentralized architecture: the management of the workflow execution is carried out by a set of devices in order to cope with the lack of dedicated infrastructure in the pervasive setting
- dynamic assignment of devices to workflow tasks: the devices in charge of executing the workflow can be discovered at runtime

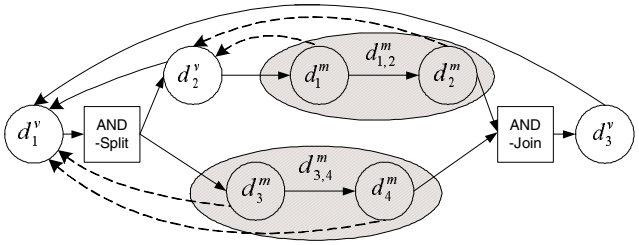


Figure 2. Protocol actors

Having designed an abstract representation of the workflow whereby devices or persons are not yet assigned to tasks, the workflow initiator launches the execution. He executes a first set of tasks before discovering in its surrounding environment a device able to perform the next set of workflow tasks. Once the discovery phase is complete, workflow data are transferred from the device that performed the discovery to the discovered one and the workflow execution continues with a new discovery procedure. To relax the availability constraints of pervasive environments, the execution is stateless so that after the completion of a set of tasks each device sends all workflow data to the next device involved in the workflow execution and thus does not have to remain online till the end of the workflow execution. In addition to workflow data, the data flow among devices includes the abstract representation of the workflow which consists of the execution plan and the functional requirements associated to each workflow step. We note W the abstract representation of a pervasive workflow and $W = (t_a)_{a\in[1,q]}$ where t_a denotes a vertex which is a set of workflow tasks that are contiguously performed by a device. The instance of W wherein q devices $(d_a)_{a\in[1,q]}$ have been assigned to the sets $(t_a)_{a\in[1,q]}$ is denoted $W_d = (d_a)_{a\in[1,q]}$.

2.2. Assuring consistency of pervasive workflows

The first step towards assuring the consistency of workflows is to be able to express transactional properties as part of the workflow model. We therefore want to offer the possibility to coordinate some tasks of a pervasive workflow to reach consistent termination states of execution when required. Our approach consists of partitioning the specification of a pervasive workflow into subsets or zones and identifying some zones called critical zones wherein transactional requirements defined by designers have to be fulfilled.

Definition 2-1. We define a critical zone C of a workflow W as a subset of W composed of contiguous vertices which require transactional consistency. We distin-

guish within C:

- $(m_k)_{k\in[1,i]}$ the i vertices whose tasks only modify mobile data
- $(v_k)_{k\in[1,j]}$ the j vertices whose tasks modify data other than mobile ones, v_1 being the first vertex of C

The device assigned to the vertex v_k (resp. m_k) is noted d_k^v (resp. d_k^m) and the instance of C is noted C_d.

We adopt a simple coordination protocol in which the coordination is managed in a centralized manner by d_1^v assigned to v_1. The role of the coordinator consists in making decisions based on the transactional requirements defined for the critical zone and the overall state of workflow execution so that the critical zone participants can reach consistent states of termination. The coordination is assured in a hierarchical way and the devices $(d_k^v)_{k\in[1,j]}$ which are subcoordinators report directly to d_1^v whereas the devices d_k^m report to the device d_x^v most recently executed [1]. For the sake of simplicity, we consider that the set of devices $\{d_l^m, d_{l+1}^m, ...d_p^m\}$ reporting to the device d_x^v form a same abstract device noted $d_{l,p}^m$ assigned to the abstract vertex $m_{l,p}$. C therefore denotes a set of n vertices (abstract or not) $C = (c_a)_{a\in[1,n]}$. In figure 2 the general features of the pervasive workflow coordination protocol are depicted.

Within the pervasive workflow model, the workflow execution is performed by devices which are assigned to vertices at runtime. Considering the diversity of devices encountered in the pervasive setting, we assume that these devices might offer in addition to different functional capabilities, a variety of transactional properties. For instance, a device can have the capability to compensate the effects of a given operation or to re-execute the operation after failure whereas some other device does not have any of these capabilities. It thus becomes necessary to select the devices executing a critical zone of a pervasive workflow not only based on functional requirements but also on transactional ones. The device assignment procedure through which devices are assigned to vertices using a match-making procedure based on functional requirements has to be augmented to integrate transactional ones. The purpose of the device assignment procedure is to create an instance of C consistent with the transactional requirements imposed by designers. It is thus required to discover first all the devices that will be executing a given critical zone before its execution is started in order to verify the existence of a set of devices that can be assigned to C. Once the instance of C has been created, the execution can start supported by the coordination protocol. The execution of the coordination protocol therefore consists of two phases: the first phase that includes the discovery and assignment of devices to vertices and the second one with the actual execution.

[1] device of type d_k^v that is located on the same branch of the workflow as these d_k^m devices and that has most recently completed its execution.

2.3. Methodology

As described in section 2.2, a coordination protocol for pervasive workflows has to meet two basic requirements. First, devices have to be assigned based on a transaction-aware process. Second, a runtime mechanism should process and assure the coordination of the execution in the face of failure scenarios. In order to achieve the first, we capitalize on the work presented in [19] whose results are reminded later on in this paper. In our approach, the devices part of a critical zone instance C_d are selected according to their transactional properties *(TP)* by means of a match-making procedure. We therefore need first to remind the semantic associated to the *TP* defined for the devices. The match-making procedure is indeed based on this semantic. This semantic is also used in order to define a tool allowing workflow designers to specify their transactional requirements *(TR)* for a given critical zone. Based on these *TR*, we are then able to assign devices to workflow vertices. Once C_d is formed, we can proceed towards the second goal by designing the actual coordination protocol.

3. Motivating example

In this section we describe a motivating example that will be used throughout the paper to illustrate the design methodology. We consider a workflow executed during a computer fair where clients, retailers and hardware providers can exchange electronically orders and invoices. The workflow W_1 specifying this example is depicted in figure 1. Alice who would like to buy a new computer makes a call for offer (vertex t_1) to three available retailers (vertices t_2, t_3, t_4). After having received some offers, she decides to go for the cheapest one and therefore contacts the corresponding retailer Bob (vertex t_5). Bob initiates the critical zone C_1 by sending an invoice to Alice and contacting his hardware provider Jack (vertex v_1 initiating the critical zone C_1). Alice pays using Bob's trusted payment platform (vertex v_2). In the meantime Jack receives the order from Bob and sends him an invoice (vertex d_1) which he pays (vertex v_3) using Jack's trusted payment platform. Afterwards, Bob starts to build the computer and ships it to Alice (vertex v_4).

Of course in this example, we need transactional properties as for instance Bob would like to have the opportunity to cancel his payment to Jack if Alice's payment is not done. Likewise, Alice would like to be refunded if Bob does not manage to assemble and ship the computer. These different scenarios refer to characteristics offered by the devices or services assigned to the workflow tasks. For example, the payment platform should be able to compensate Alice's payment and Jack's payment platform should offer the possibility to cancel an order. Yet, it is no longer necessary for Jack to provide the cancellation option if the payment platform claims that it is reliable and not prone to transaction

errors. In this example we do not focus on the trust relationship between the different entities and therefore assume the trustworthiness of each of them yet we are rather interested in the transactional characteristics offered by each participant.

4. Transactional model

In this section, we remind and extend the semantic specifying the *TP* offered by devices described in [19] before specifying the consistency evaluation tool associated to this semantic. This semantic model is based on the "transactional Web service description" defined in [6].

4.1. Transactional Properties of Devices

In [6] a model specifying semantically the *TP* of Web services is presented. This model is based on the classification of computational tasks made in [17, 22] which considers three different types of *TP*. A task and by extension a device executing this task can be:

- Compensatable: the data modified by the task can be rolled back
- Retriable: the task is sure to complete successfully after a finite number of tries
- Pivot: the task is neither compensatable nor retriable

In the definition of a critical zone, we distinguish two sets of devices: $(d_k^m)_{k\in[1,i]}$ which only modify mobile or volatile data and $(d_k^v)_{k\in[1,j]}$ which only modify data other than mobile ones, e.g. remote database, production of an item, etc. Based on this distinction, the above mentioned transactional model has to be extended. This model describes the modification of permanent data and is thus only appropriated for database systems whereas the pervasive setting introduces in addition transactional properties representing devices' characteristics such as battery level, fiability, connectivity, etc. A new transactional property representing the reliability of a device is therefore introduced.

- A device is reliable (resp. unreliable) if it is highly unlikely (resp. likely) that the device will fail due to hardware failures (battery level, communication medium access, ...)

To properly detail this model, we can map the *TP* with the state of data modified by the devices during the execution of computational tasks. This mapping is depicted in figure 3. Basically, data can be in three different states: initial (0), unknown (x), completed (1). In the state (0), it means either that the vertex execution has not yet started $initial$, the execution has been $aborted$ before starting, or the data modified have been $compensated$ after completion. In state (1) it means that the vertex has been properly $completed$. In state (x) it means either that

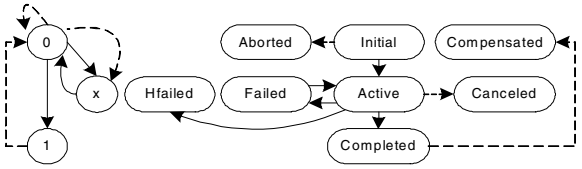


Figure 3. State model

the execution is $active$, the execution has been stopped, $canceled$ before completion, the execution has $failed$ or an hardware failure, $Hfailed$ happened. These *TP* allow to define eight types of devices: (Reliable,Retriable) (r_l,r_t), (Reliable,Compensatable) (r_l,c), (Reliable,Retriable and Compensatable) (r_l,r_tc), (Reliable,Pivot) (r_l,p) and the four others Unreliable (ur_l). We must distinguish within this model the inherent termination states: $failed$, $completed$ and $Hfailure$ which result from the normal course of the execution and the ones received during a coordination protocol instance: $compensated$, $aborted$ and $canceled$ which force a vertex to either stop or rollback. The *TP* of the devices are only differentiated by the states $failed$, $compensated$ and $Hfailed$ which indeed respectively specify the retriability, compesatability and reliability aspects.

Definition 4-1. We have for a given device d:

- $failed$ is not a termination state of $d \Leftrightarrow d$ is retriable
- $compensated$ is a termination state of $d \Leftrightarrow d$ is compensatable
- $Hfailed$ is not a termination state of $d \Leftrightarrow d$ is reliable

From the state transition diagram, we can also derive some simple rules. The states $failed$, $completed$, $Hfailed$ and $canceled$ can only be reached if the device is in the state $active$. The state $compensated$ can only be reached if the device is in the state $completed$. The state $aborted$ can only be reached if the device is in the state $initial$.

Regarding the distinction made on the nature of vertices within a critical zone, we specify some requirements for the devices selected for a critical zone execution. On the one hand, as the devices $(d_k^v)_{k\in[1,j]}$ modify sensitive and permanent data, we consider that they are required to be reliable. There are therefore four types of d_k^v devices: (r_l,r_t), (r_l,c), (r_l,r_tc) and (r_l,p). On the other hand, as the devices of type d_k^m only modify mobile and volatile data, we consider first that they are retriable besides compensatability is not required for volatile data. Second, we assume that these tasks can be executed by unreliable devices and there are as a result only two types of d_k^m devices: (r_l,r_t) and (ur_l,r_t). If one of the d_k^m devices part of the abstraction $d_{l,p}^m$ is unreliable then $d_{l,p}^m$ is unreliable, otherwise $d_{l,p}^m$ is reliable. Figure 4 depicts the transition diagram for the six types of transactional devices that can be encountered.

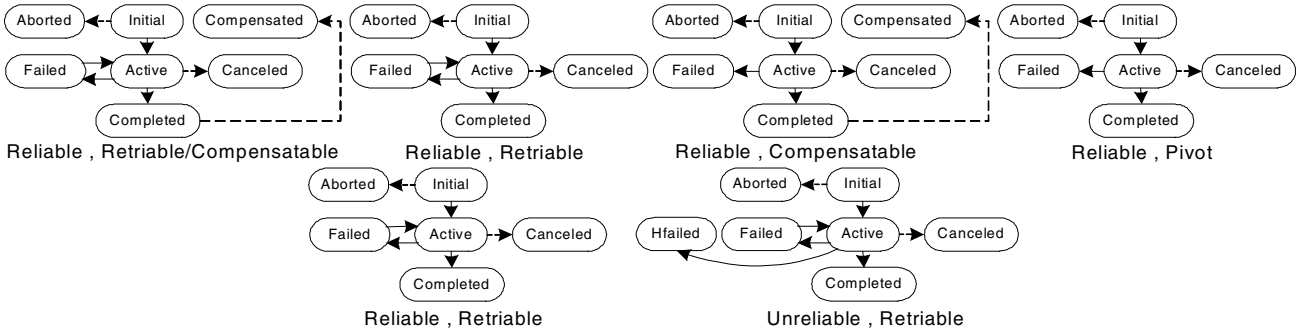


Figure 4. State diagrams of devices d_k^v and d_k^m

4.2. Termination states

The crucial point of the transactional model specifying the *TP* of devices is the analysis of their possible termination states. The ultimate goal is indeed to be able to define consistent termination states for a critical zone i.e. determining for each device executing a critical zone vertex which termination states it is allowed to reach.

Definition 4-2. We define the operator termination state $ts(x)$ which specifies the possible termination states of the element x. This element x can be:

- a device d and $ts(d) \in \{aborted, canceled, failed, Hfailed, completed, compensated\}$
- a vertex c and $ts(c) \in \{aborted, canceled, failed, Hfailed, completed, compensated\}$
- a critical zone composed of n vertices $C = (c_a)_{a\in[1,n]}$ and $ts(C) = (ts(c_1), ts(c_2), ..., ts(c_n))$
- an instance C_d of C composed of n devices $C_d = (d_a)_{a\in[1,n]}$ and $ts(C_d) = (ts(d_1), ts(d_2), ..., ts(d_n))$

The operator $TS(x)$ represents the finite set of all possible termination states of the element x, $TS(x) = (ts_k(x))_{k\in[1,j]}$. We have especially, $TS(C_d) \subseteq TS(C)$ since the set $TS(C_d)$ represents the actual termination states that can be reached by C_d according to the *TP* of the devices assigned to C. We also define for x critical zone or critical zone instance and $a \in [1, n]$:

- $ts(x, c_a)$: the value of $ts(c_a)$ in $ts(x)$
- $tscomp(x)$: the termination state of x such that $\forall\ a \in [1, n]\ ts(x, c_a) = completed$.

For the remaining of the paper, $C = (c_a)_{a\in[1,n]}$ denotes a critical zone of n vertices and $C_d = (d_a)_{a\in[1,n]}$ an instance of C.

4.3. Transactional consistency tool

We use the Acceptable Termination States (ATS) [21] model as the consistency evaluation tool for the critical zone. ATS defines the termination states a critical zone is allowed to reach so that its execution is judged consistent.

Definition 4-3. $ATS(C)$ is the subset of $TS(C)$ whose elements are considered consistent by workflow designers. A consistent termination state of C is called an acceptable termination state $ats_k(C)$ and we note $ATS(C) = (ats_k(C))_{k\in[1,i]}$ the set of Acceptable Termination States of C i.e. the *TR* of C.

$ATS(C)$ and $TS(C)$ can be represented by a table which defines for each termination state the tuple of termination states reached by each vertex as depicted in figures 1 and 5. As mentioned in the definition, the specification of the set $ATS(C)$ is done at the workflow designing phase. $ATS(C)$ is mainly used as a decision table for a coordination protocol so that C_d can reach an acceptable termination state knowing the termination state of at least one vertex. The coordination decision, i.e. the termination state that has to be reached, made given a state of the critical zone execution has to be unique, this is the main characteristic of a coordination protocol. In order to cope with this requirement, $ATS(C)$ which is used as input for the coordination decision-making process has thus to verify some properties.

5. Forming $ATS(C)$

In this section the definitions and theorems introduced and proved in [19] are reminded and adapted to specify $ATS(C)$ in the case of pervasive workflows. The approach followed is based on the fact that $ATS(C) \subseteq TS(C)$ thus $ATS(C)$ inherits the characteristics of $TS(C)$. We make the assumption that only one device can fail at a time. As explained above the unicity of the coordination decision during the execution of a coordination protocol is a major requirement. We try here to identify the elements of $TS(C)$ that correspond to different coordination decisions given the same state of a workflow execution. There are two situations whereby a protocol coordination has different possibilities of coordination given the state of a workflow vertex. Let $a, b \in [1, n]$ and assume that the vertex c_b has failed:

- the vertex c_a is in the state $completed$ and either it remains in this state or it is $compensated$

- the vertex c_a is in the state $active$ and either it is $canceled$ or the coordinator let it reach the state $completed$

From these two statements, we define the *incompatibility from a coordination perspective* and the *flexibility*.

Definition 5-1. Let $k, l \in [1, j]$. $ts_k(C)$ and $ts_l(C)$ are said incompatible from a coordination perspective $\Leftrightarrow \exists\ a,b \in [1, n]$ such that $ts_k(C, c_a) = completed$, $ts_k(C, c_b) = ts_l(C, c_b) \in \{failed, Hfailed\}$ and $ts_l(C, c_a) = compensated$. Otherwise, $ts_l(C)$ and $ts_k(C)$ are said compatible from a coordination perspective.

The value in $\{compensated, completed\}$ reached by a vertex c_a in a termination state $ts_k(C)$ whereby $ts_k(C, c_b) \in \{failed, Hfailed\}$ is called recovery strategy of c_a against c_b in $ts_k(C)$. By extension, we can consider the recovery strategy of a set of vertices against a given vertex.

Definition 5-2. Let $a, b \in [1, n]$. A vertex c_a is flexible against $c_b \Leftrightarrow \exists\ k \in [1, j]$ such that $ts_k(C, c_b) \in \{failed, Hfailed\}$ and $ts_k(C, c_a) = canceled$. Such a termination state is said to be flexible to c_a against c_b. The set of termination states of C flexible to c_a against c_b is denoted $FTS(c_a, c_b)$.

From these definitions, we now study the termination states of C according to the compatibility and flexibility criterias in order to identify the termination states that follow a common strategy of coordination.

Definition 5-3. Let $a \in [1, n]$. A termination state of C $ts_k(C)$ is called generator of $c_a \Leftrightarrow ts_k(C, c_a) \in \{failed, Hfailed\}$ and $\forall\ b \in [1, n]$ such that c_b is executed before or in parallel of c_a, $ts_k(C, c_b) \in \{completed, compensated\}$. The set of termination states of C compatible with $ts_k(C)$ generator of c_a is denoted $CTS(ts_k(C), c_a)$.

The set $CTS(ts_k(C), c_a)$ specifies all the termination states of C that follow the same recovery strategy as $ts_k(C)$ against c_a.

Definition 5-4. Let $ts_k(C) \in TS(C)$ be a generator of c_a. Coordinating an instance C_d of C in case of the failure of c_a consists in choosing the recovery strategy of each vertex of C against c_a and the $z_a < n$ vertices $(c_{a_i})_{i\in[1,z_a]}$ flexible to c_a whose execution is not $canceled$ when c_a fails. We call coordination strategy of C_d against c_a the set $CS(C_d, ts_k(C), (c_{a_i})_{i\in[1,z_a]}, c_a) = CTS(ts_k(C), c_a) - \bigcup_{i=1}^{z_a} FTS(c_{a_i}, c_a)$. If the device d_a assigned to c_a is retriable then $CS(C_d, ts_k(C), (c_{a_i})_{i\in[1,z_a]}, c_a) = \emptyset$

C_d is said to be coordinated according to $CS(C_d, ts_k(C), (c_{a_i})_{i\in[1,z_a]}, c_a)$ if in case of the failure of c_a, C_d reaches a termination state in $CS(C_d, ts_k(C), (c_{a_i})_{i\in[1,z_a]}, c_a)$. Of course, it assumes that the *TP* of C_d are sufficient to reach $ts_k(C)$.

Given a vertex c_a the idea is to classify the elements of $TS(C)$ using the sets of termination states compatible with the generators of c_a. Using this approach, we can identify the different recovery strategies and the coordination strategies associated with the failure of c_a as we decide which vertices can be $canceled$. Defining $ATS(C)$ is therefore deciding at design time the termination states of C that are consistent. $ATS(C)$ is to be inputted to a coordination protocol in order to provide it with a set of rules which leads to a unique coordination decision in any cases. According to the definitions and properties we introduce above, we can now explicit some rules on $ATS(C)$ so that the unicity requirement of coordination decisions is respected.

Available Devices		Retriable	Compensatable	Reliable
v_1	d_{11}	yes	no	yes
v_2	d_{21}	no	yes	yes
	d_{22}	yes	no	yes
m_1	d_{31}	yes	no	no
v_3	d_{41}	no	yes	yes
v_4	d_{51}	yes	no	yes
	d_{52}	yes	yes	yes

ATS(C_1)		v_1	v_2	m_1	v_3	v_4
ats_1	ts_1	completed	completed	completed	completed	completed
ats_2	ts_4	completed	compensated	completed	completed	failed
ats_3	ts_{11}	completed	compensated	completed	failed	aborted
ats_4	ts_{12}	completed	canceled	completed	failed	aborted
ats_5	ts_{17}	completed	compensated	hfailed	aborted	aborted
ats_6	ts_{18}	completed	canceled	hfailed	aborted	aborted
ats_7	ts_{22}	completed	failed	completed	aborted	aborted
ats_8	ts_{23}	completed	failed	canceled	aborted	aborted
ats_9	ts_{27}	completed	failed	completed	compensated	aborted
ats_{10}	ts_{28}	completed	failed	completed	canceled	aborted
ats_{11}	ts_{32}	failed	aborted	aborted	aborted	aborted

Figure 5. $ATS(C_1)$ **and available devices**

Definition 5-5. Let $a, k \in [1, n] \times [1, j]$ such that $ts_k(C, c_a) \in \{failed, Hfailed\}$ and $ts_k(C) \in ATS(C)$. $ATS(C)$ is valid $\Leftrightarrow \exists\ !\ l \in [1, j]$ such that $ts_l(C)$ generator of c_a compatible with $ts_k(C)$ and $CTS(ts_l(C), c_a) - \bigcup_{i=1}^{z_a} FTS(c_{a_i}, c_a) \subset ATS(C)$ for a set of vertices $(c_{a_i})_{i\in[1,z_a]}$ flexible to c_a.

A valid $ATS(C)$ therefore contains for all $ts_k(C)$ in which a vertex fails a unique coordination strategy associated to this failure and the termination states contained in this coordination strategy are compatible with $ts_k(C)$. In figure 5, an example of possible ATS is presented for the critical zone C_1. It just consists in selecting the termination states of the table $TS(C_1)$ that we consider consistent and respect the validity rule for the created $ATS(C_1)$. For example here the payment of Alice has to be compensated if Bob fails to deliver the computer as specified in $ats_2 = ts_4$.

6. Assigning devices using ATS

In this section, we remind the main steps of the device assignment procedure whose underpinning theorems are proved in [19]. The transaction-aware device assignment procedure aims at assigning n devices to the n vertices c_a in order to create an instance of C *acceptable* with respect to a valid $ATS(C)$. We first define a validity criteria for the instance C_d of C with respect to $ATS(C)$, the device assignment algorithm is then detailed. Finally, we specify the coordination strategy associated to the instance

created from our assignment scheme.

6.1. Acceptability of C_d with respect to $ATS(C)$

Definition 6-1. C_d is an acceptable instance of C with respect to $ATS(C) \Leftrightarrow TS(C_d) \subseteq ATS(C)$.

Now we express the condition $TS(C_d) \subseteq ATS(C)$ in terms of coordination strategies. The termination state generator of c_a present in $ATS(C)$ is noted $ts_{k_a}(C)$. The set of vertices whose execution is not $canceled$ when c_a fails is noted $(c_{a_i})_{i\in[1,z_a]}$. We get the theorem 6-2 [19].

Theorem 6-2. $TS(C_d) \subseteq ATS(C) \Leftrightarrow \forall\ a \in [1,n]$ $CS(C_d, ts_{k_a}(C), (c_{a_i})_{i\in[1,z_a]}, c_a) \subset ATS(C)$.

It should be noted that if $failed \notin ATS(C, c_a)$ where $ATS(C, c_a)$ represents the acceptable termination states of the vertex c_a in $ATS(C)$ then $CS(C_d, ts_{k_a}(C), (c_{a_i})_{i\in[1,z_a]}, c_a) = \emptyset$.

6.2. Transaction-aware assignment procedure

The device assignment algorithm uses $ATS(C)$ as a set of requirements during the device assignment procedure and thus identifies those devices whose *TP* match the *TR* associated with vertices defined in $ATS(C)$. The assignment procedure is an iterative process, devices are assigned to vertices sequentially. At each step i, the assignment procedure therefore generates a partial instance of C noted C_d^i. $TS(C_d^i)$ refers to the termination states of C that can be reached based on the *TP* of the i devices that are already assigned. Intuitively the acceptable termination states refer to the degree of flexibility offered when choosing the devices with respect to the different coordination strategies complying with $ATS(C)$. This degree of flexibility is influenced by two parameters:

- The list of acceptable termination states for each workflow vertex. This list can be determined based on $ATS(C)$. Using this list, the requirements on the *TP* of a candidate device can be derived since this device can only reach the states defined in $ATS(C)$ for the considered vertex.
- The assignment process is iterative and therefore, as new devices are assigned to vertices, both $TS(C_s^i)$ and the *TP* required for the assignment of further devices are updated.

We therefore need to define first the *TR* for the assignment of a device after i steps in the assignment procedure.

6.2.1. Extraction of *TR*. From the two requirements above, we define for a vertex c_a :

- $ATS(C, c_a)$: Set of acceptable termination states of c_a which is derived from $ATS(C)$
- $DIS(c_a, C_d^i)$: Set of *TR* that the device assigned to c_a must meet based on previous assignments. This set is determined based on the following reasoning:

  (DIS_1): the device must be compensatable $\Leftrightarrow$ $compensated \in DIS(c_a, C_d^i)$

  (DIS_2): the device must be retriable $\Leftrightarrow$ $failed \notin DIS(c_a, C_d^i)$

  (DIS_3): the device must be reliable $\Leftrightarrow$ $Hfailed \notin DIS(c_a, C_d^i)$

Using these two sets, we are able to compute $Min_{TP}(d_a, c_a, C_d^i) = ATS(C, c_a) \bigcap DIS(c_a, C_d^i)$ which defines the minimal *TP* a device d_a has at least to comply with in order to be assigned to the vertex c_a at the $i+1$ assignment step. We simply check the retriability and compensatability properties for the set $Min_{TP}(d_a, c_a, C_d^i)$:

- $failed \notin Min_{TP}(d_a, c_a, C_d^i) \Leftrightarrow d_a$ has to verify the retriability property
- $Hfailed \notin Min_{TP}(d_a, c_a, C_d^i) \Leftrightarrow d_a$ has to verify the reliability property
- $compensated \in Min_{TP}(d_a, c_a, C_d^i) \Leftrightarrow d_a$ has to verify the compensatability property

The set $ATS(C, c_a)$ is easily derived from $ATS(C)$. We need now to compute $DIS(c_a, C_d^i)$. We assume that we are at the $i+1$ step of an assignment procedure, i.e. the current partial instance of C is C_d^i. Computing $DIS(c_a, C_d^i)$ means determining if (DIS_1), (DIS_2) and (DIS_3) are true. From these two statements we can derive three properties:

1. (DIS_1) implies that state $compensated$ can definitely be reached by c_a
2. (DIS_2) implies that c_a can not $fail$
3. (DIS_2) implies that c_a can not be $canceled$
4. (DIS_3) implies that c_a can not $Hfail$

The third property is derived from the fact that if a vertex can not be $canceled$ when the failure of a vertex has occurred, then it has to finish its execution and reach at least the state $completed$. In this case, if a device can not be $canceled$ then it can not fail, which is the third property. To verify whether 1., 2., 3. and 4. are true, we introduce the following theorems [19].

Theorem 6-3. *Let $a \in [1,n]$. The state $compensated$ can definitely be reached by $c_a \Leftrightarrow \exists\ b \in [1,n] - \{a\}$ verifying (6-3b): d_b not retriable (resp. reliable) is assigned to c_b and $\exists\ ts_k(C) \in ATS(C)$ generator of c_b such that $ts_k(C, c_a) = compensated$.*

Theorem 6-4. *Let $a \in [1,n]$. c_a can not fail (resp. Hfail) $\Leftrightarrow \exists\ b \in [1,n] - \{a\}$ verifying (6-4b): (d_b not compensatable is assigned to c_b and $\exists\ ts_k(C) \in ATS(C)$ generator of c_a such that $ts_k(C, c_b) = compensated$) or ($c_b$ is flexible to c_a and d_b not retriable is assigned to c_b and $\forall\ ts_k(C) \in ATS(C)$ such that $ts_k(C, c_a) = failed$ (resp. $ts_k(C, c_a) = Hfailed$), $ts_k(C, t_b) \neq canceled$).*

Theorem 6-5. *Let $a, b \in [1,n]$ such that c_a is flexible to c_b. c_a is not $canceled$ when c_b fails (resp. Hfail) $\Leftrightarrow$ (6-5b): d_b not retriable (resp. not reliable) is assigned to c_b and*

$\forall\, ts_k(C) \in ATS(C)$ *such that* $ts_k(C, c_b) = failed$ *(resp.* $ts_k(C, c_b) = Hfailed$*),* $ts_k(C, c_a) \neq canceled$*.*

In order to compute $DIS(c_a, C_d^i)$, we have to compare c_a with each of the i vertices $c_b \in C - \{c_a\}$ to which a device d_b has been already assigned. Two cases have to be considered: either we assign a device to a vertex v_k or to an abstract vertex $m_{l,p}$. This is an iterative procedure. At the initialization phase in the first case we have: since no vertex has been yet compared to $c_a = v_k$, d_a can be *(p)*: $DIS(c_a, C_d^i) = \{failed\}$.

1. if c_b verifies (*6-3b*) $\Rightarrow compensated \in DIS(c_a, C_d^i)$
2. if c_b verifies (*6-4b*) $\Rightarrow failed \notin DIS(c_a, C_d^i)$
3. if c_b is flexible to c_a and verifies (*6-5b*) $\Rightarrow failed \notin DIS(c_a, C_d^i)$

In this case, the verification stops if $failed \notin DIS(c_a, C_d^i)$ and $compensated \in DIS(c_a, C_d^i)$. For the vertices of type v_k, we indeed only need to check the retriability and compensatability properties.

In the second case, we have at the initialization phase: since no vertex has been yet compared to $c_a = m_{l,p}$, d_a can be *(*ur_l*)*: $DIS(c_a, C_d^i) = \{Hfailed\}$.

4. if c_b verifies (*6-4b*) $\Rightarrow Hfailed \notin DIS(c_a, C_d^i)$

In that case, the verification stops if $Hfailed \notin DIS(c_a, C_d^i)$. For the vertices of type $m_{l,p}$ we only need to check the reliability property.

Finally, when $Min_{TP}(d_a, c_a, C_d^i)$ is computed, we are able to select the appropriate device to be assigned to a given vertex according to ***TR***.

6.2.2. Device assignment process. Devices are assigned to each vertex based on an iterative process. Depending on the ***TR*** and the ***TP*** of the devices available for each vertex, different scenarios can occur:

(i) devices *(rc)* are available in the case of a vertex v_k or devices (r_l) are available in the case of a vertex $m_{l,p}$ (i.e. all the devices of the abstraction are (r_l)). It is not necessary to compute ***TR*** as such devices match all ***TR***.

(ii) only devices *(p)* are available in the case of a vertex v_k or only devices (ur_l) are available in the case of a vertex $m_{l,p}$ (i.e. one of the device of the abstraction is (ur_l)). We need to compute the ***TR*** associated to the vertex and either pivot (resp. unreliable) is sufficient or there is no solution.

(iii) devices *(r)* and *(c)* but no *(rc)* are available for a vertex v_k. We need to compute the ***TR*** associated to the vertex and we have three cases. First, *(rc)* is required and therefore there is no solution. Second, *(r)* (resp. *(c)*) is required and we assign a device *(r)* (resp. *(c)*) to the vertex. Third, there is no requirement.

The assignment procedure is performed by the coordinator c_1. Devices have to be assigned to all vertices prior to the beginning of the critical zone execution. The first vertex is assigned to the critical zone initiator and of course it has to match the ***TR*** of the first vertex. The idea is then to assign first devices to the vertices verifying *(i)* and *(ii)* since there is no flexibility in the choice of the device. Vertices verifying *(iii)* are finally analyzed. Based on the ***TR*** raised by the remaining vertices, we first assign devices to vertices with a non-empty ***TR***. We then handle the assignment for vertices with an empty ***TR***. Note that the ***TR*** of all the vertices to which devices are not yet assigned are also affected (updated) as a result of the current device assignment. If no vertex has ***TR*** then we assign the devices *(r)* to assure the completion of the remaining vertices' execution.

6.3. Actual termination states of C_d

Once all the devices have been assigned to vertices we can coordinate their execution so that they respect the defined ***TR***. In order to do so, we need to know the actual termination states subset of ***ATS(C)*** that can be reached by the defined instance of C. Having computed $TS(C_d)$, we can deduce the coordination rules associated to the execution of C_d. This subset is determined using the following theorem that is proved in [19].

Theorem 6-6. *Let* C_d *be an acceptable instance of* C *with respect to* $ATS(C)$*. We note* $(c_{a_i})_{i\in[1,n_r]}$ *the set of vertices to which neither a retriable nor a reliable device has been assigned.* $ts_{k_{a_i}}(C)$ *is the generator of* c_{a_i} *present in* $ATS(C)$ *and* $(c_{a_{i_j}})_{j\in[1,z_{a_i}]}$ *denotes the set of vertices which are not canceled when* c_{a_i} *fails.* $TS(C_d)$=$\{tscomp(C_d)\} \bigcup \bigcup_{i=1}^{n_r} (CTS(ts_{k_{a_i}}(C), c_{a_i}) - \bigcup_{j=1}^{z_{a_i}} FTS(c_{a_{i_j}, c_{a_i}}))$.

6.4. Example

We consider the critical zone C_1 of figure 1. Designers have defined $ATS(C_1)$ of figure 5 as the ***TR***. The set of available devices for each vertex of C_1 is specified in the figure 5. The goal is to assign devices to vertices so that the instance of C_1 is valid with respect to $ATS(C_1)$ and we apply the presented assignment procedure. The critical zone initiator assigned to v_1 uses a device (r) matching the ***TR***. We now start to assign the devices *(rc)* and (r_l) for which it is not necessary to compute any ***TR***. d_{52} which is the only available device *(rc)* is therefore assigned to v_4. We then try to assign the devices *(p)* and (ur_l), and we verify whether d_{31} can be assigned to m_1. We compute $Min_{TP}(d_a, m_1, C_{1d}^2) = ATS(C_1, m_1) \bigcap DIS(m_1, C_{1d}^2)$. $ATS(C_1, m_1) = \{completed, Hfailed\}$ and $DIS(m_1, C_{1d}^2) = \{Hfailed\}$ as d_{52} and d_{11} are the only device already assigned and the theorems 6-3, 6-4 and 6-5 are not verified. Thus $Min_{TP}(c_a, m_1, C_{1d}^2) = \{Hfailed\}$ and d_{31} can be assigned to m_1 as it matches the ***TR***. Now we compute the ***TR***

$TS(C_{1d})$	d_{11}	d_{21}	d_{31}	d_{41}	d_{51}
ts_1	completed	completed	completed	completed	completed
ts_{11}	completed	compensated	completed	failed	aborted
ts_{12}	completed	canceled	completed	failed	aborted
ts_{17}	completed	compensated	hfailed	aborted	aborted
ts_{18}	completed	canceled	hfailed	aborted	aborted
ts_{22}	completed	failed	completed	aborted	aborted
ts_{23}	completed	failed	canceled	aborted	aborted
ts_{27}	completed	failed	completed	compensated	aborted
ts_{28}	completed	failed	completed	canceled	aborted

Figure 6. $TS(C_{1d})$

d_1^v		d_k^v		d_k^m	
Receives	**Sends**	**Receives**	**Sends**	**Receives**	**Sends**
Completed	Compensate	Compensate	Abort	Leave	Aborted
Failed	Cancel	Cancel	Aborted	Cancel	Canceled
Aborted	Abort	Abort	Canceled	Ack	Alive
Canceled	Leave	Leave	Cancel	Abort	Ack
Compensated	Ping	Aborted	Leave	Ping	Completed
Ack		Alive	Ping		
Hfailed		Ack	Ack		
Alive		Ping	Hfailed		
		Canceled	Alive		
		Completed	Compensated		
		Canceled	Failed		
			completed		

Figure 7. Notification messages

of v_2 and we get $Min_{TP}(d_a, v_2, C_{1d}^3) = \{compensated\}$ as theorem 6-3 is verified with the devices d_{31}. The device d_{21} can thus be assigned to v_2 as it matches the *TR* of the task. We get for v_3 $Min_{TP}(d_a, v_3, C_{1d}^4) = \{compensated\}$. The device d_{41} which is *(c)* verifies the *TR* is assigned to v_3. Using the created instance of C_1 we get the set $TS(C_{1d})$ of figure 6.

7. Coordination Protocol Specification

Having introduced the method through which an instance of C is obtained by assigning devices to workflow vertices according to the *TR* of C, we turn to the actual coordination of devices during the execution of the critical zone. The protocol that is in charge of the coordination is specified in terms of the different actors, notification messages and coordination cases. We finally motivate the chosen solution by comparing it with existing coordination protocols.

7.1. Protocol actors

As mentioned in section 2.2 and figure 2 we distinguish three main entities within the coordination protocol execution:

- Device $d_1^v = c_1$: this device is the critical zone initiator and is in charge of performing the device assignment procedure and coordinating the execution of C. The coordination decisions are made using the table $TS(C_d)$ specifying the subset of $ATS(C)$ C_d is actually able to reach.

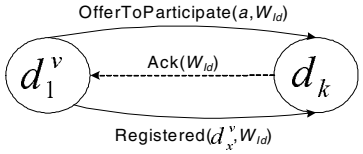


Figure 8. Device registration

- Devices d_k^v: these devices modify sensitive data and play the role of subcoordinators. They report their state of execution and the state of execution of the devices d_k^m to d_1^v .
- Devices d_k^m: these devices modify volatile data and report to the device d_x^v most recently executed.

Actors exchange messages for the purpose of decision making and forwarding as listed in figure 7. These messages are mostly derived from the state diagram of the transactional model and the respective role of the devices in the protocol. The flow of notification messages within the protocol execution and the mechanisms involved in the processing of these notification messages are stated in the next section.

7.2. Coordination scenarios

In this section, we detail the different phases and coordination scenarios that can be encountered during the execution of the protocol. First, we explain how devices are registered with the coordination protocol during the device assignment phase. Then, we analyse the message flow between the different actors of the protocol in three different scenarios: normal course of execution, failure of a device d_k^v and failure of a device d_k^m.

7.2.1. Device registration. The first phase of the coordination protocol consists of the discovery and registration of the devices that will be involved in the critical zone execution. The discovery process through which devices that can be assigned to critical zone vertices are identified is performed by the device $c_1 = d_1^v$. The *TR* extraction procedure, specified in section 6.2.1 provides the coordinator with a list of suitable devices that match the computed *TR*. It is then necessary to contact the devices of this list in order to receive from of one of them the commitment to execute the requested vertex. Based on the registration handshake depicted in figure 8, the coordinator d_1^v contacts a device asking it whether it agrees to commit to execute the operation a of the workflow whose identifier is W_{Id}. Once the newly assigned device's coordinator is known, d_1^v sends the information. In the case of devices d_k^v this information is known from the beginning since d_1^v is their coordinator whereas for the devices d_k^m, the information is known when d_x^v the device d_k^v most recently executed has been assigned to a vertex.

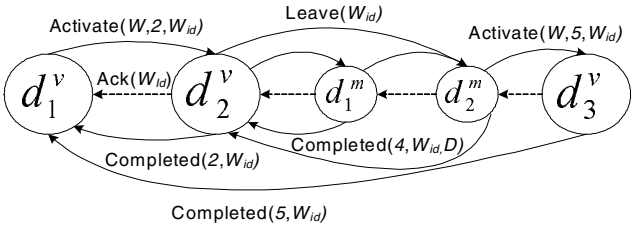


Figure 9. Normal execution

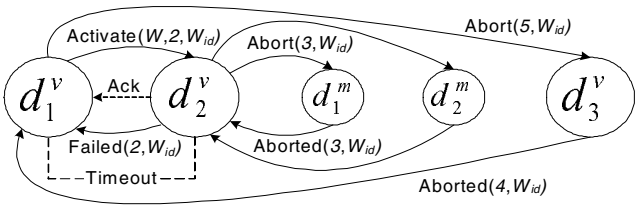


Figure 10. Failure of a device d_k^v

7.2.2. Normal course of execution. Once all involved devices are known, the critical zone execution can start as part of the coordination protocol. Devices are sequentially activated based on the workflow specification. A sample for normal execution of C is depicted in figure 9. The $Activate(W, k, W_{Id}, D)$ message is a workflow message defined in [18], it especially contains the workflow specification W, the requested vertex k to be executed, the workflow data D modified during the execution and the workflow identifier W_{Id}. Within the critical zone execution local acknowledgments $Ack(W_{Id})$ are used. Each device d_k^m reports its status to the device d_x^v most recently executed and once its execution is complete it can leave the critical zone execution. The $Completed(k, W_{Id}, D)$ message sent by a device d_k^m includes a backup copy of the volatile data modified by the device that can be reused later on for the recovery procedure in case of failure of a device d_k^m (Section 7.2.4). Once in the state $completed$, devices of type d_k^m can leave the coordination as they won't be asked to compensate their execution. Depending on the *TR* defined for C, devices d_k^v may leave the critical zone before the end of the critical zone execution. A device d_k^v is indeed able to leave the coordination if it reaches the state $completed$ regardless of possible failures in the sequel of the critical zone execution. The condition allowing a device d_k^v to leave the coordination is therefore stated as follows.

Theorem 7-1. *A device d_k^v assigned to a vertex c_l can leave the execution of a critical zone $C \Leftrightarrow$ the device d_k^v is in the state $completed$ and $\forall\, i \in [1, n]$ such that a device d_i not retriable (resp. not reliable) is assigned to the vertex c_i, d_i is in the state $initial$ and $ts_k(C, c_l) = completed$ where $ts_k(C)$ is the termination state generator of c_i in $TS(C_d)$.*

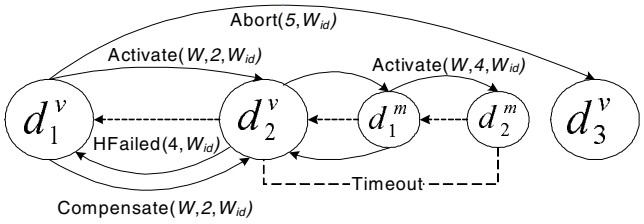


Figure 11. Failure of a device d_k^m

7.2.3. Failure of a device d_k^v. This scenario is only possible with devices *(p)*. We can encounter two situations: either the failure is total and the device is not able to communicate any longer or the device is still alive and can forward a failure message to d_1^v. Figure 10 depicts the two cases whereby the total failure is detected using a simple timeout in Ping/Alive message exchanges. Once the failure has been detected, the coordinator forwards the coordination decision to all involved devices. It should be noted here that devices *(r)* can also reach the state $failed$ but the retriability property implies that they have at their disposal recovery solutions ensuring that the contact is never permanently lost. Thus, the failure of devices *(r)* is transparent to the rest of the coordination and does not have to be handled.

7.2.4. Failure of a device d_k^m. The failure of a device d_k^m is detected by its subcoordinator with a timeout. As specified in the transactional model, we indeed consider that devices of type d_k^m can only fail because of hardware problems and failure of such devices therefore implies a loss of contact with their coordinator. The failure of a device d_k^m is reported by its subcoordinator to the device d_1^v. The failure detection and forwarding of the $Hfailed$ message are depicted in figure 11.

7.3. Coordination decisions and recovery

Having detailed various coordination scenarios that can occur during the execution of a critical zone, we analyse the possible recovery strategies, in particular the replacement of failed devices d_k^v and d_k^m and how coordination decisions are made upon detection of a failure.

7.3.1. Replacement of failed devices d_k^v. During the course of the execution new devices can be discovered and assigned to vertices in order to replace failed ones. In fact two situations can happen: either the failure of a device occurs while executing its assigned vertex or the coordinator loses contact (timeout detection) prior to the activation of the device. The first situation is specified in the previous section and no backup solution is possible as the data modified by the failed device are in an unknown state. In the second situation, it is possible on the contrary to assign a new device matching the *TR* to the vertex which has not yet

started with the execution. Once the loss of contact with a device d_k^v is detected, no coordination decision is yet sent to devices and the execution continues. If no device is found to be assigned to the vertex when its execution should be activated, the protocol coordinator considers the device it has lost contact with as failed.

7.3.2. Replacement of failed devices d_k^m. In case of failure of a device d_k^m, be it before or after its activation, a recovery procedure can be executed prior to informing the coordinator of the hardware failure. It is indeed possible to assign to the vertex a new device so that the execution can go on. This is possible as on the one hand the devices d_k^m only modify volatile data and on the other hand, we have a backup copy of the data modified by the devices that are part of the abstract vertex $d_{l,p}^m$. Once the failure is detected, the subcoordinator of the failed device tries to assign a new device to the failed vertex. In this case, only volatile data are being modified, *TR* is not a concern and the assignment procedure can be repeated till a device manages to execute the requested vertex.

7.3.3. Reaching consistent termination states. Once all possible recovery mechanisms have been attempted, a coordination decision is made by the coordination d_1^v. The table $TS(C_d)$ is the input to the coordination decisions that are made throughout the execution of a critical zone. Once the failure of a vertex c_a has been detected, the protocol coordinator reads in $TS(C_d)$ the set $CS(C_d, ts_k(C), (c_{a_i})_{i\in[1,z_a]}, c_a)$ listing the possible termination states reachable by C_d whereby c_a is $failed$. There is a unique element of this set that is reachable by C_d with respect to its current state of execution and d_1^v sends the appropriate messages so that the overall critical zone can reach this consistent termination state.

7.4. Discussion

The coordination protocol integrates the semantic description of involved devices and relies on an adaptive decision table which is computed during the assignment procedure. The coordination protocol is flexible as it completely depends on the designers' choice for the specification of Acceptable Termination States. This solution therefore offers a full support of relaxed atomicity constraints for workflow based applications and is also self-adaptable to the devices characteristics which is not the case with recent efforts [12],[13].

The organization of the coordination is based on a simple hierarchical approach as in BTP [3]. In that respect, the central point of the coordination is the device d_1^v on which relies the whole coordination. This is the main weakness of the protocol, as a failure of this device would cause the complete failure of the workflow execution. The role of critical zone initiator of the coordination is therefore reserved to devices that are both reliable and retriable. Nonetheless, this centralized and hierarchical approach facilitates the management of the coordination process.

In addition to usual coordination phases such as coordination registration, device completion and failure our protocol, offers the possibility to replace participants at runtime depending on their role within the coordination and the volatility degree of data they have to modify during the workflow execution. This makes the protocol flexible and adapted to the pervasive paradigm whereas such recovery procedure is not specified in other transactional protocols.

In the protocol description, we do not specify the data recovery strategy especially for the compensated states. Different approaches can be integrated with our work to support either forward error recovery or backward error recovery [15]. The choice of the recovery strategy basically depends on the application and its fault-handling protocol. For instance, a simple backward error recovery strategy is sufficient for workflows used for payment in the example of the paper whereas a forward recovery strategy might be required for a hotel booking system. Existing mechanisms in this area can therefore be used to augment our transactional protocol to specify complex fault-handling and compensation scenarios [1], [23].

8. Related work

Transactional consistency of distributed systems such as workflows and database systems has been an active research topic over the last 15 years [9] yet it is still an open issue in the area of distributed processes within the enterprise services architecture paradigm (ESA) [7, 11, 16]. In this paper we specified a transactional protocol for the pervasive workflow architecture presented in [18] and our solution uses and extends the results presented and proved in [19].

The execution of distributed processes wherein business partners are not assigned at design time introduces new requirements for transactional systems such as dynamicity, semantic description and relaxed atomicity. Existing transactional models for advanced applications and workflows [8] do not offer the flexibility to integrate these requirements [4]. In comparison, our solution allows the specification of *TR* supporting relaxed atomicity for an abstract workflow specification and the selection of semantically described devices or services fulfilling the defined *TR*. In addition, we provide the means to compute a coordination protocol adapted to the workflow instance resulting from our device assignment procedure.

In [6], the first approach specifying relaxed atomicity requirements for Web services based workflow applications using the *ATS* tool and a transactional semantic is presented. Despite a solid contribution, this work provides only means to verify the consistency of composite services but no means

to integrate *TR* at the composition phase. This work therefore appears to be limited when it comes to the possible integration into dynamic and distributed business processes. In this approach, *TR* do not play any role in the component devices selection process which may result in several attempts to determine a valid workflow instance. As opposed to this work, our solution provides a systematic procedure enabling the creation of valid workflow instances by means of a transaction-aware device assignment procedure.

The transactional protocol proposed in this paper offers adapted means to respond to the constraints introduced by environments where heterogeneous devices share resources in a collaborative manner. Using relaxed atomicity features, the protocol indeed offers the flexibility for devices to release their resources as soon as their participation to the workflow is no longer required. Moreover using a flexible semantic, devices are able to advertise their capabilities so that they can assume a role suited to any workflows in which their resources can be used. Current efforts in the area of the transactional coordination of business processes [12, 13] do not offer such flexibility. On the one hand, they suffer from the lack of tools for the specification of transactional requirements and their integration into business partners' selection process. On the other hand, no recovery procedure is specified as part of the protocol for the replacement of devices in case of failure.

9. Conclusion

We presented an adaptive transactional protocol for the pervasive workflow model developed in [18]. Our solution enables first the selection of the devices part of the workflow execution based on transactional requirements defined at the workflow design phase. Using transactional properties offered by selected devices and the defined transactional requirements, a decision table is computed as a basis for the coordination of the execution. The coordination protocol itself offers a framework that supports relaxed atomicity requirements and takes into account the respective role and characteristics of each device involved in the workflow execution. We believe that our approach can be used to augment recent specifications [14] in increasing their flexibility to incorporate devices' transactional properties in the definition of adaptive coordination rules. Yet, aspects such as the trust that one can place in the transactional properties claimed by devices or services are still open issues.

References

[1] Business process execution language for web sevices, http://www.ibm.com/developerworks/library/ws-bpel/.

[2] Web services tool kit for mobile devices, http://www.alphaworks.ibm.com/tech/wstkmd 2002.

[3] M. Abbott and al. Business transaction protocol, 2005.

[4] G. Alonso, D. Agrawal, A. E. Abbadi, M. Kamath, R. Gnthr, and C. Mohan. Advanced transaction models in workflow contexts. In *Proc. 12th International Conference on Data Engineering, New Orleans, February 1996.*

[5] S. Berger, S. McFaddin, C. Narayanaswami, and M. Raghunath. Web services on mobile devices - implementation and experience. In *Fifth IEEE Workshop on Mobile Computing Systems and Applications*, 2003.

[6] S. Bhiri, O. Perrin, and C. Godart. Ensuring required failure atomicity of composite web services. In *Proc. of the 14th international conference on World Wide Web*, 2005.

[7] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana. The next step in web services. *Commun. ACM*, 46(10), 2003.

[8] A. K. Elmagarmid. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, 1992.

[9] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[10] P. Greenfield, A. Fekete, J. Jang, and D. Kuo. Compensation is not enough. In *Proc. of the 7th International Enterprise Distributed Object Computing Conference (EDOC'03)*, 2003.

[11] M. Gudgin. Secure, reliable, transacted; innovation in web services architecture. In *Proc. of the ACM International Conference on Management of Data, Paris, France*, 2004.

[12] D. Langworthy and al. Ws-atomictransaction, 2005.

[13] D. Langworthy and al. Ws-businessactivity, 2005.

[14] D. Langworthy and al. Ws-coordination, 2005.

[15] P. A. Lee and T. Anderson. *Fault Tolerance: Principles and Practice*. Morgan Kaufmann, 1990.

[16] M. Little. Transactions and web services. *Commun. ACM*, 46(10):49–54, 2003.

[17] S. Mehrotra, R. Rastogi, A. Silberschatz, and H. Korth. A transaction model for multidatabase systems. In *Proc. of the 12th IEEE International Conference on Distributed Computing Systems (ICDCS92)*, 1992.

[18] F. Montagut and R. Molva. Enabling pervasive execution of workflows. In *Proceedings of the 1st IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom*, 2005.

[19] F. Montagut and R. Molva. Augmenting Web services composition with transactional requirements. In *ICWS 2006, IEEE International Conference on Web Services, September 18-22, 2006, Chicago, USA*, Sep 2006.

[20] A. Ranganathan and S. McFaddin. Using workflows to coordinate web services in pervasive computing environments. In *Proceedings of the IEEE International Conference on Web Services 2004*, pages 288–295, July 2004.

[21] M. Rusinkiewicz and A. Sheth. Specification and execution of transactional workflows. In *Modern database systems: the object model, interoperability, and beyond*, 1995.

[22] H. Schuldt, G. Alonso, and H. Schek. Concurrency control and recovery in transactional process management. In *Proc. of the Conference on Principles of Database Systems*, 1999.

[23] F. Tartanoglu, V. Issarny, A. Romanovsky, and N. Levy. Coordinated forward error recovery for composite web services, 2003.