The Pervasive Workflow: A Decentralized Workflow System Supporting Long Running Transactions

Frederic Montagut, *Student Member, IEEE,* Refik Molva, *Member, IEEE,* and Silvan Tecumseh Golega

***Abstract*— Workflow technologies are becoming pervasive in that they enable the execution of business processes in distributed and ubiquitous computing environments. As long running transactions, the execution of workflows in environments without dedicated infrastructures raises transactional requirements due to the dynamicity of resources available to run a workflow instance and the integration of relaxed atomicity constraints at both design and instantiation time. In this paper, we propose an adaptive transactional protocol for the pervasive workflow model developed in a previous work to support the execution of business processes in the pervasive setting. The execution of this protocol takes place in two phases. First candidate business partners are assigned to tasks using an algorithm wherein the selection process is based on both 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 business partner assignment procedure. The resulting workflow execution is compliant with the defined consistency requirements and the coordination decisions depend on the transactional characteristics offered by the partners assigned to each task. An implementation of our theoretical results relying on OWL-S and BPEL technologies is further detailed as a proof of concept.**

***Index Terms*— Decentralized workflows, transaction-aware composition, transactional consistency**

I. Introduction

Workflow technologies are becoming pervasive in that they enable the execution of long running business processes and transactions in distributed and ubiquitous environments [1], [2], [3]. The adequate execution support for pervasive workflows has to cope with the lack of dedicated infrastructure for management and control tasks in order to provide business users with means to leverage the resources available in their surrounding environment. To that effect a first step has been achieved by the design of a fully decentralized workflow architecture based on the Service Oriented Computing paradigm [4]. Featuring a dynamic assignment of tasks to workflow partners, this architecture allows users to initiate workflows in any environment where surrounding users' resources can be advertised by various means including a service discovery mechanism. Yet, this architecture does not provide any guarantee on the consistency of the outcome reached by the process execution. Considering the lack of reliability akin to distributed environments, data and transaction consistency is a main issue. Transactional requirements raised by the execution of processes on top of the pervasive workflow infrastructure are twofold: on the one hand, the workflow execution is dynamic in that the workflow partners offering different characteristics can be assigned to tasks depending on the resources available at runtime and on the other hand, 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 [5], [6] 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 management system developed in [4]. The execution of this protocol takes place in two phases. First, business partners are assigned to tasks using an algorithm wherein the selection process is based on functional and transactional requirements. These transactional requirements are defined at the workflow design stage using the Acceptable Termination States (*ATS*) model. 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 business partner assignment procedure. The resulting workflow execution is compliant with the defined consistency requirements and the coordination decisions depend on the characteristics of the partners assigned to each task. Besides, it should be noted that the practical solutions that are presented in this work do not only answer specific requirements introduced by the pervasive workflow model but are sufficiently generic to be applied to other workflow architectures supporting long-running transactions.

The remainder of the paper is organized as follows. Section II introduces preliminary definitions and the methodology underpinning our approach. We present an example of pervasive workflow execution in section III for the purpose of illustrating our results throughout the paper. Section IV introduces a detailed description of the transactional model used to represent the characteristics offered by business partners. Section V describes how transactional requirements expressed by means of the *ATS* model are derived from the inherent properties of termination states. Section VI and section VII present the transaction-aware partner assignment procedure and the associated coordination protocol, respectively. An implementation of our theoretical results based on Web services technologies including OWL-S [7] and BPEL [8] is presented in section VIII. Section IX discusses related work while section X presents concluding remarks.

Manuscript received ; revised . This work has been partially sponsored by EU IST Directorate General as a part of FP6 IST projects MOSQUITO and R4eGov and by SAP Labs France S.A.S.

F. Montagut is with SAP Labs France, 805 av Dr. Donat, 06250 Mougins, France

R. Molva is with Institut Eurecom, 2229 Route des Cretes, 06904 Sophia-Antipolis, France

S. Golega is with Hasso-Plattner-Institut, Postfach 900460 -14440 Potsdam, Germany

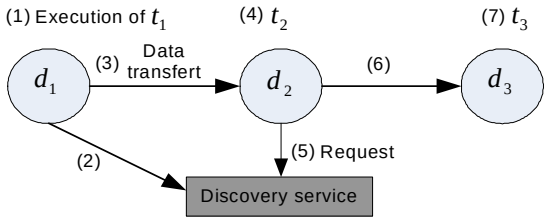

Fig. 1. Pervasive workflow runtime specification

II. 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 in charge of management and control tasks. After a short overview of the features offered by the pervasive workflow architecture, we specify the set of requirements in terms of transactional consistency that must be met by the execution of pervasive workflows.

A. Pervasive workflows

In this section, we present the pervasive workflow model that was designed in [4]. The pervasive workflow concept introduces a workflow management system supporting the execution of business processes in environments whereby computational resources offered by each business partner can potentially be used by any party within the surroundings of that business partner. This workflow management system features a distributed architecture characterized by two objectives:

- fully decentralized architecture: the management of the workflow execution is distributed amongst the partners taking part in a workflow instance in order to cope with the lack of dedicated infrastructure in the pervasive setting
- dynamic assignment of business partners to workflow tasks: the peers in charge of executing the workflow can be discovered at runtime based on available resources

The runtime specification of the pervasive workflow architecture is depicted in figure 1. Having designed an abstract representation of the workflow whereby business partners are not yet assigned to functional tasks, the workflow initiator d_1 launches the execution. d_1 executes a first set of tasks t_1 (1) before discovering in its surrounding environment a partner able to perform the next tasks t_2 (2). Once the discovery phase is complete, workflow data are transferred from the peer that performed the discovery to the discovered one (3) and the workflow execution further proceeds with the processing of the next set of tasks t_3 (4). The sequence composed of the discovery request, the transfer of workflow data and the execution of a set of tasks is iterated till the final vertex. In order to relax the availability constraints of pervasive environments, the execution is stateless so that after the completion of a set of tasks each business partner sends all workflow data to the next partner involved in the workflow execution and thus does not have to remain online till the end of a workflow instance. Along with workflow application data, the flow of data amongst business partners includes the abstract representation of the workflow which consists of the execution plan and the functional requirements associated with 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 performed by a business partner from the receipt of workflow data till the transfer of these data to the next partner. The instance of W wherein q business partners $(d_a)_{a\in[1,q]}$ have been assigned to the sets $(t_a)_{a\in[1,q]}$ is denoted $W_d = (d_a)_{a\in[1,q]}$.

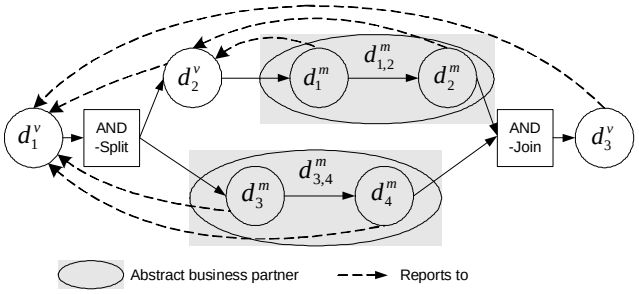

Fig. 2. Protocol actors

B. Assuring consistency of pervasive workflows

As a first step towards assuring workflow consistency one has to be able to express transactional requirements as part of the workflow model. We therefore want to offer the possibility to coordinate some tasks of a pervasive workflow instance in order to assure the consistency of termination states. Our approach consists in 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 to meet some transactional requirements. We distinguish within C:

- $(m_k)_{k\in[1,i]}$ the i vertices whose tasks only modify mobile or volatile 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 business partner 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 transactional 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 given the overall state of workflow execution so that the critical zone execution can reach a consistent state of termination. The coordination is assured in a hierarchical way and the business partners $(d_k^v)_{k\in[1,j]}$ which are subcoordinators report directly to d_1^v whereas the partners d_k^m report to the business partner d_x^v most recently executed [1]. For the sake of simplicity, we consider that the set of business partners $\{d_l^m, d_{l+1}^m, ...d_p^m\}$ reporting to the business partner d_x^v form an abstract partner named $d_{l,p}^m$ that is assigned to the abstract vertex $m_{l,p}$. C therefore denotes a set of n vertices (abstract or not) $C = (c_a)_{a\in[1,n]}$. This reporting strategy based on the type of business partners is depicted in figure 2.

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

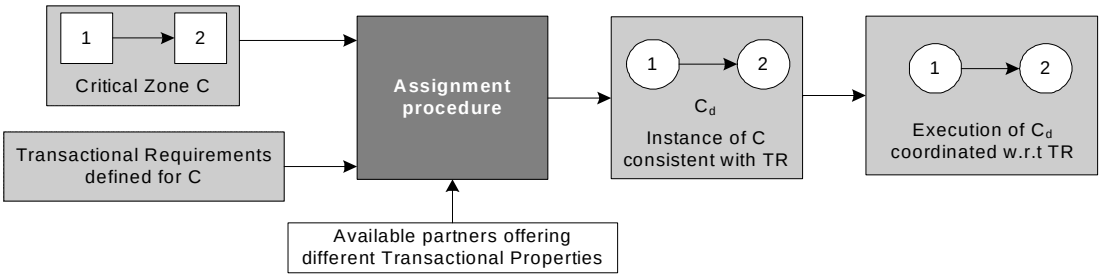

Fig. 3. Methodology

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

C. Methodology

As described in section II-B, a coordination protocol designed to support the execution of pervasive workflows has to meet two basic requirements. First, business partners 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 [9] whose results are reminded later on in this paper. In our approach, the partners part of a critical zone instance C_d are selected according to their transactional properties by means of a matchmaking procedure. We therefore need to specify first the semantic associated with the transactional properties offered by business partners. The matchmaking 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 for a given critical zone. Based on these transactional requirements, business partners can be assigned to workflow vertices. Finally, once C_d is formed we can proceed towards the second goal by expressing the coordination rules inherent to C_d and designing the actual coordination protocol in charge of processing those rules. This methodology basically follows the steps of the transactional pervasive workflow lifecyle from the instantiation to the execution as depicted in figure 3.

III. 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 used in this example is depicted in figure 4. Alice would like to buy a new computer and makes a call for offer to three available retailers. After having received some offers, she decides to go for the cheapest one and therefore contacts the corresponding retailer Bob. Bob initiates the critical zone C_1 by sending an invoice to Alice and contacting his hardware provider Jack (vertex v_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 m_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 to define transactional requirements 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 business partners 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.

IV. Transactional model

In this section, we provide and extend the semantic specifying the transactional properties offered by business partners described in [9] before specifying the consistency evaluation tool associated with this semantic. The semantic model is based on the "transactional Web service description" defined in [10].

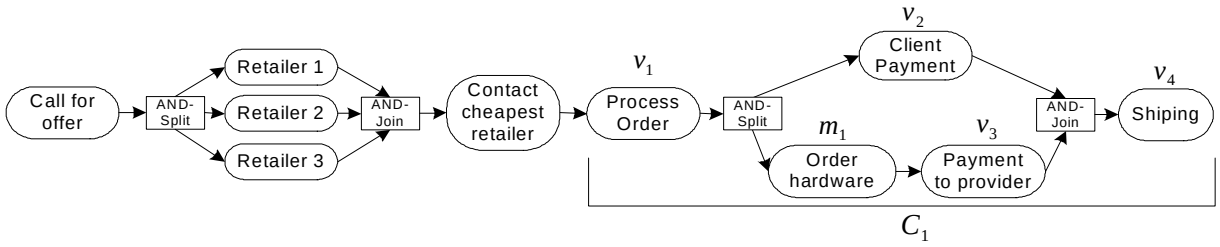

Fig. 4. Workflow example: Deal at a fair

$TS(C_1)$	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

Fig. 5. Termination states of C_1

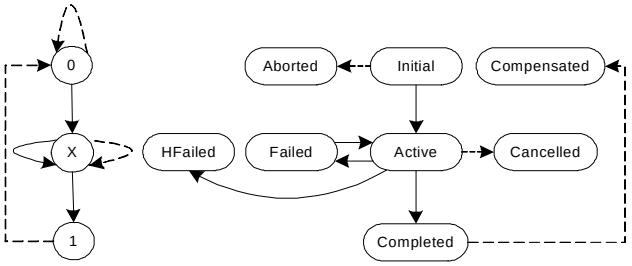

Fig. 6. State model

A. Transactional Properties of Business Partners

In [10] a model specifying semantically the transactional properties of Web services is presented. This model is based on the classification of computational tasks made in [11], [12] which considers three different types of transactional properties. A task and by extension a business partner 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 business partners: $(d_k^m)_{k\in[1,i]}$ 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 relevant to database systems whereas the pervasive setting introduces in addition transactional properties representing business partners' hardware characteristics such as battery level, reliability, connectivity, etc. A new transactional property representing the reliability of a business partner is therefore introduced.

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

To properly detail this model, we can map the transactional properties with the state of data modified by the business partners during the execution of computational tasks. This mapping is depicted in figure 6. 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 the execution is $active$, the execution has been stopped, $canceled$ before completion, the execution has $failed$ or an hardware failure, $Hfailed$ happened. These transactional properties allow to define eight types of business partners: (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 $Hfailed$ which result from the normal course of the task execution
- the forced termination states: $compensated$, $aborted$ and $canceled$ which result from a coordination message received during a coordination protocol instance and forcing a task execution to either stop or rollback

In the state diagrams of figures 6 and 7 plain and dashed lines represent the inherent transitions leading to inherent states and the forced transitions leading to forced states, respectively.

The transactional properties of the business partners are only differentiated by the states $failed$, $compensated$ and $Hfailed$ which indeed respectively specify the retriability, compensatability and reliability aspects.

Definition 4-1. We have for a given partner 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 business partner is in the state $active$. The state $compensated$ can only be reached if the partner is in the state $completed$. The state $aborted$ can only be reached if the partner is in the state $initial$.

Regarding the distinction made on the nature of vertices within a critical zone, we specify some requirements for the business partners selected for a critical zone execution.

On the one hand, as the partners $(d_k^v)_{k\in[1,j]}$ modify sensitive and permanent data, we consider that they are required to be reliable. There are therefore four types of d_k^v partners: (r_l,r_t), (r_l,c), (r_l,$r_t c$) and (r_l,p).

On the other hand, as the business partners 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 partners and there are as a result only two types of d_k^m partners: (r_l,r_t) and (ur_l,r_t). If one of the d_k^m partners 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 7 depicts the transition diagram for the six types of transactional partners that can be encountered.

B. Termination states

The crucial point of the transactional model specifying the transactional properties of business partners 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 partner 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 partner 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 partners $C_d = (d_a)_{a\in[1,n]}$ 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 transactional properties of the partners 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.

C. Transactional consistency tool

We use the Acceptable Termination States (ATS) [13] 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 deemed consistent.

Definition 4-3. An $ATS(C)$ is a subset of $TS(C)$ whose elements are considered consistent by workflow designers for a specific execution of C. A consistent termination state of C is called an acceptable termination state $ats_k(C)$, thus $ATS(C) = (ats_k(C))_{k\in[1,i]}$. A set $ATS(C)$ specifies the transactional requirements defined by designers associated with a specific execution 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 5 and 8. Depending on the application different ATS tables can of course be specified by designers for the same critical zone C, and for the sake of readability we do not introduce in this paper an index (as in $ATS_i(C)$) in the notation $ATS(C)$. As mentioned in the definition, the specification of $ATS(C)$ 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 that are specified in the next section.

V. Forming $ATS(C)$

In this section the definitions and theorems introduced and proved in [9] are reminded and adapted to specify $ATS(C)$ in the scope of the pervasive workflow model. The approach followed is based on the fact that $ATS(C) \subseteq TS(C)$ thus $ATS(C)$ inherits the characteristics of $TS(C)$. For the sake of clarity in what follows, we make the assumption that only one business partner can fail at a time during a pervasive workflow instance. Our approach can indeed be easily extended to concurrent failure scenarios as discussed later on in section VI-D.

As explained above the unicity of the coordination decision during the execution of a coordination protocol is a major requirement. We thus 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* notions.

Definition 5-1. Two termination states of C $ts_k(C)$ and $ts_l(C)$ are said incompatible from a coordination perspective iff $\exists$ 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

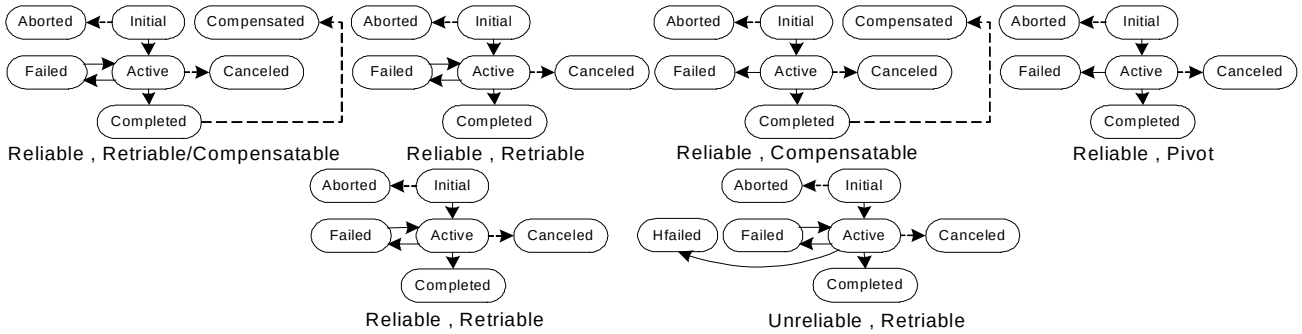

Fig. 7. State diagrams of business partners d_k^v and d_k^m

$ts_l(C, c_a) = compensated$. 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 $ts_k(C)$.

Definition 5-2. A vertex c_a is flexible against an other vertex c_b iff $\exists\ k \in [1, j]$ 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. A termination state of C $ts_k(C)$ is called generator of a vertex c_a iff $ts_k(C, c_a) \in \{failed, Hfailed\}$ and $\forall\ b \in [1, n]$ such that c_b is executed before or in parallel with 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 $(v_{a_i})_{i\in[1,z_a]}$ flexible to c_a whose execution is not $canceled$ when c_a fails. As unreliable business partners modify only volatile data we consider that cancellation is always performed if a task execution is still active as soon as a failure occurs. The set $(v_{a_i})_{i\in[1,z_a]}$ is thus only composed of vertices of type v_k. We call coordination strategy of C_d against c_a the set $CS(C_d, ts_k(C), (v_{a_i})_{i\in[1,z_a]}, c_a) = CTS(ts_k(C), c_a) - \bigcup_{i=1}^{z_a} FTS(v_{a_i}, c_a)$. If the partner d_a assigned to c_a is retriable then $CS(C_d, ts_k(C), (v_{a_i})_{i\in[1,z_a]}, c_a) = \emptyset$.

C_d is said to be coordinated according to $CS(C_d, ts_k(C), (v_{a_i})_{i\in[1,z_a]}, c_a)$ if in case of the failure of c_a, C_d reaches a termination state in $CS(C_d, ts_k(C), (v_{a_i})_{i\in[1,z_a]}, c_a)$. Of course, it assumes that the transactional properties 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 introduced above, we can now explicit some rules on $ATS(C)$ so that the unicity requirement of coordination decisions is respected.

Available Partners		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

Fig. 8. $ATS(C_1)$ and available business partners

Definition 5-5. Let $ts_k(C) \in TS(C)$ such that $ts_k(C, c_a) \in \{failed, Hfailed\}$ and $ts_k(C) \in ATS(C)$. $ATS(C)$ is valid iff $\exists\ !\ l \in [1, j]$ such that $ts_l(C)$ generator of c_a compatible with $ts_k(C)$ and $CTS(ts_l(C), c_a) - \bigcup_{i=1}^{z_a} FTS(v_{a_i}, c_a) \subset ATS(C)$ for a set of vertices $(v_{a_i})_{i\in[1,z_a]}$ 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 with this failure and the termination states contained in this coordination strategy are compatible with $ts_k(C)$. In figure 8, 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 $ats_2 = ts_4$.

VI. Assigning business partners using ATS

In this section, we specify the main steps of the partner assignment procedure whose underpinning theorems are proved in [9]. The transaction-aware business partner assignment

procedure aims at assigning n business partners 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 business partner assignment algorithm is then detailed. Finally, we specify the coordination strategy associated with the instance created from our assignment scheme.

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

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

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 $(v_{a_i})_{i\in[1,z_a]}$. We get the theorem 6-2 [9].

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

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

B. Transaction-aware assignment procedure

The business partner assignment algorithm uses $ATS(C)$ as a set of requirements during the partner assignment procedure and thus identifies those partners whose transactional properties match the transactional requirements associated with vertices defined in $ATS(C)$. The assignment procedure is an iterative process, partners 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 transactional properties of the i partners that are already assigned. Intuitively the acceptable termination states refer to the degree of flexibility offered when choosing the partners 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 transactional properties of a candidate partner can be derived since this partner can only reach the states defined in $ATS(C)$ for the considered vertex.
- The assignment process is iterative and therefore, as new partners are assigned to vertices, both $TS(C_s^i)$ and the transactional properties required for the assignment of further partners are updated. For instance, we are sure to no longer reach the termination states $CTS(ts_k(C), c_a)$ allowing the failure of the vertex c_a in $ATS(C)$ when we assign a partner retriable and reliable to c_a. In this specific case, we no longer care about the states reached by other vertices in $CTS(ts_k(C), c_a)$ and therefore there is no transactional requirements introduced for the vertices to which business partners have not already been assigned.

We therefore need to define first the transactional requirements for the assignment of a partner after i steps in the assignment procedure.

1) Extraction of transactional requirements: 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 transactional requirements that the partner assigned to c_a must meet based on previous assignments. This set is determined based on the following reasoning:
  (DIS_1): the partner must be compensatable iff $compensated \in DIS(c_a, C_d^i)$
  (DIS_2): the partner must be retriable iff $failed \notin DIS(c_a, C_d^i)$
  (DIS_3): the partner must be reliable iff $Hfailed \notin DIS(c_a, C_d^i)$

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 transactional properties a partner 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 three statements we can derive four 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 business partner 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 present the following theorems that are an extension of results proved in [9].

Theorem 6-3. *Let $a \in [1,n]$. The state $compensated$ can definitely be reached by c_a iff $\exists\ b \in [1,n] - \{a\}$ 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) iff $\exists\ b \in [1,n] - \{a\}$ 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) iff **(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 business partner d_b has been already assigned. Two cases have to be considered: either we assign a business partner 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 of type (r_l, 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 of type *(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 business partner to be assigned to a given vertex according to transactional requirements.

2) Business partner assignment process: Business partners are assigned to each vertex based on an iterative process. Depending on the transactional requirements and the transactional properties of the business partners available for each vertex, different scenarios can occur:

(i) business partners of type *$(r_l, r_t c)$* are available in the case of a vertex v_k or business partners of type (r_l) are available in the case of a vertex $m_{l,p}$ (i.e. all the business partners of the abstraction are of type (r_l)). It is not necessary to compute any transactional requirements as such partners match all transactional requirements.

(ii) a single partner is available for the considered vertex. We need to compute the transactional requirements associated with the vertex and either the transactional properties offered by this partner are sufficient or there is no solution.

(iii) business partners of type *(r_l, r_t)* and *(r_l, c)* but none of type *$(r_l, r_t c)$* are available for a vertex v_k. We need to compute the transactional requirements associated with the vertex and we have three cases. First, *$(r_l, r_t c)$* is required and therefore there is no solution. Second, *(r_l, r_t)* (resp. *(r_l, c)*) is required and we assign a business partner of type *(r_l, r_t)* (resp. *(r_l, c)*) to the vertex. Third, there is no requirement.

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

C. Actual termination states of C_d

Once all the business partners have been assigned to vertices we can coordinate their execution so that they respect the defined transactional requirements. 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 with the execution of C_d. This subset is determined using the following theorem that is proved in [9].

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 business partner has been assigned. $ts_{k_{a_i}}(C)$ is the generator of c_{a_i} present in $ATS(C)$ and $(v_{a_{ij}})_{j\in[1,z_{a_i}]}$ denotes the set of vertices which are not canceled when c_{a_i} fails.* $TS(C_d)$=$\{tscomp(C_d)\} \bigcup \bigcup_{i=1}^{n_r} (CTS(ts_{k_{a_i}}(C), c_{a_i}) - \bigcup_{j=1}^{z_{a_i}} FTS(v_{a_{ij}}, c_{a_i}))$.

$TS(C_d)$ is indeed derived from $ATS(C)$ which contains for all vertices at most a single coordination strategy as specified in 5-5. As a result, whenever the failure of a vertex c_a is detected, a transactional protocol in charge of coordinating an instance C_d resulting from our approach reacts as follows. The coordination strategy $CS(C_d, ts_k(C), (v_{a_i})_{i\in[1,z_a]}, c_a)$ corresponding to c_a is identified and a unique termination state belonging to $CS(C_d, ts_k(C), (v_{a_i})_{i\in[1,z_a]}, c_a)$ can be reached given the current state of the critical zone execution.

D. Discussion and performance evaluation

In order to handle the scenarios wherein more than one business partner can fail at a time, one would need to extend the definition of the termination state generator to take into account the failure of a set of partners as follows.

Definition 6-7. A termination state of C $ts_k(C)$ is called generator of a set of vertices $(c_{a_i})_{i\in[1,p]}$ iff $\forall i \in [1, p]$ $ts_k(C, c_{a_i}) \in \{failed, Hfailed\}$ and $\forall\ b \in [1, n]$ such that c_b is executed before one of the vertices $(c_{a_i})_{i\in[1,p]}$, or in parallel with all the vertices $(c_{a_i})_{i\in[1,p]}$, $ts_k(C, c_b) \in \{completed, compensated\}$.

The compatibility definition would be also defined for termination states in which exactly the same set of partners fail at the same time so that coordination strategies are defined for possible concurrent failures. In this case, two termination

states such that the set of failures in the first is a subset of the set of failures in the second are not incompatible. The composition algorithm and the coordination protocol are not affected by this configuration.

The operations that are relevant from the complexity point of view are twofold: the definition of transactional requirements by means of the acceptable termination states model and the execution of the transaction-aware business partner assignment procedure.

One can argue that building an *ATS* table specifying the transactional requirements of a business process W consists of computing the whole $TS(W)$ table, yet this is not the case. Building a $ATS(C)$ set in fact only requires for designers to identify the vertices of C that they allow to fail as part of the process execution and to select the termination state generator associated with each of those vertices that meet their requirements in terms of failure atomicity. Once this phase is complete, designers only need to select the vertices whose execution can be canceled when the former vertices may fail and complete the associated coordination strategy.

The second aspect concerns the complexity of the transaction aware assignment procedure that we presented in section VI.

Theorem 6-8. *Let $C = (c_a)_{a\in[1,n]}$ a critical zone. The complexity of the transaction-aware assignment procedure is $O(n^3)$.*

Proof: We can show that the number of operations necessary to compute the step i of the assignment procedure for a vertex c_a is bounded by $4 \times n \times i$. Computing the step i indeed consists of verifying the theorems 6-3, 6-4 and 6-5 and determining $ATS(C, c_a)$. On the one hand, performing the operations part of theorems 6-3 (one comparison), 6-4 (two comparisons) and 6-5 (one comparison) requires at most 4 comparisons. On the other hand, building $ATS(C, c_a)$ requires at most n operations (there is at most n generators in a $ATS(C)$ set). Therefore, we can derive that the number of operations that needs to be performed in order to compute the n steps of the assignment procedure for a critical zone composed of n tasks is bounded by $4 \times n \times \sum_{j=1}^{n} j$ which is equivalent to n^3 as $n \longrightarrow \infty$. ∎

E. Example

We consider the critical zone C_1 of figure 4. Designers have defined $ATS(C_1)$ of figure 8 as the transactional requirements. The set of available business partners for each vertex of C_1 is specified in the figure 8. The goal is to assign business partners 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 business partner of type (r_l, r_t) matching the transactional requirements. We now start to assign the business partners of type $(r_l, r_t c)$ and (r_l) for which it is not necessary to compute any transactional requirements. d_{52} which is the only available business partner of type $(r_l, r_t c)$ is therefore assigned to v_4. We then try to assign business partners to tasks for which there is no choice, and we verify whether d_{31} can be assigned to m_1. We compute $Min_{TP}(d_a, m_1, C^2_{1d}) = ATS(C_1, m_1) \bigcap DIS(m_1, C^2_{1d})$.

$TS(C_{1d})$	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

Fig. 9. $TS(C_{1d})$

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		
			Failed		
			completed		

Fig. 10. Notification messages

$ATS(C_1, m_1) = \{completed, Hfailed\}$ and $DIS(m_1, C^2_{1d}) = \{Hfailed\}$ as d_{52} and d_{11} are the only business partner already assigned and the theorems 6-3, 6-4 and 6-5 are not verified. Thus $Min_{TP}(c_a, m_1, C^2_{1d}) = \{Hfailed\}$ and d_{31} can be assigned to m_1 as it matches the transactional requirements. We get for v_3 $Min_{TP}(d_a, v_3, C^3_{1d}) = \{failed\}$. The business partner d_{41} which is of type (r_l, c) verifies the transactional requirements is assigned to v_3. Now we compute the transactional requirements of v_2 and we get $Min_{TP}(d_a, v_2, C^4_{1d}) = \{failed, compensated\}$ as theorem 6-3 is verified with the business partners d_{31}. The partner d_{21} can thus be assigned to v_2 as it matches the transactional requirements of the task. Using the created instance of C_1 we get the set $TS(C_{1d})$ of figure 9.

VII. Coordination Protocol Specification

Having introduced the method through which an instance of C is obtained by assigning partners to workflow vertices according to the transactional requirements of C, we turn to the actual coordination of partners 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.

A. Protocol actors

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

- Business partner $d_1^v = c_1$: this business partner is the critical zone initiator and is in charge of performing the business partner 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.
- Business partners d_k^v: these business partners modify sensitive data and play the role of subcoordinators. They

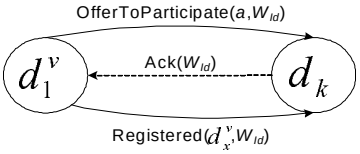

Fig. 11. Business partner registration

report their state of execution and the state of execution of the business partners d_k^m to d_1^v .

- Business partners d_k^m: these partners modify volatile data and report to the partner d_x^v most recently executed.

Actors exchange messages for the purpose of decision making and forwarding as listed in figure 10. These messages are mostly derived from the state diagram of the transactional model and the respective role of the partners 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.

B. 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 partners are registered with the coordination protocol during the partner 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 partner d_k^v and failure of a partner d_k^m.

1) Business partner registration: The first phase of the coordination protocol consists of the discovery and registration of the business partners that will be involved in the critical zone execution. The discovery process through which business partners that can be assigned to critical zone vertices are identified is performed by the business partner $c_1 = d_1^v$. The transactional requirements extraction procedure, specified in section VI-B provides the coordinator with a list of suitable business partners that match the computed transactional requirements. It is then necessary to contact the business partners 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 11, the coordinator d_1^v contacts a business partner asking it whether it agrees to commit to execute the operation a of the workflow whose identifier is W_{Id}. Once the newly assigned business partner's coordinator is known, d_1^v sends the information. In the case of business partners d_k^v this information is known from the beginning since d_1^v is their coordinator whereas for the business partners d_k^m, the information is known when d_x^v the business partner d_k^v most recently executed has been assigned to a vertex.

2) Normal course of execution: Once all involved business partners are known, the critical zone execution can start supported by the coordination protocol. Business partners are sequentially activated based on the workflow specification. A sample for normal execution of C is depicted in figure 12. The $Activate(W, k, W_{Id}, D)$ message is a workflow message defined in [4], 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

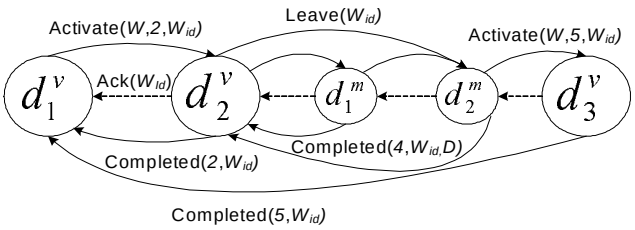

Fig. 12. Normal execution

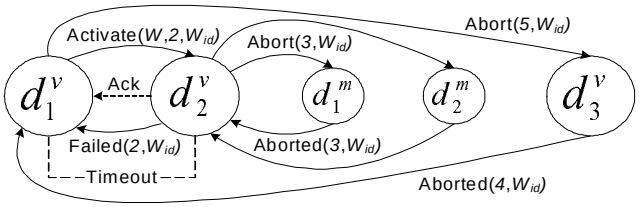

Fig. 13. Failure of a business partner d_k^v

W_{Id}. Within the critical zone execution local acknowledgments $Ack(W_{Id})$ are used. Each business partner d_k^m reports its status to the business partner 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 business partner d_k^m includes a backup copy of the volatile data modified by the business partner that can be reused later on for the recovery procedure in case of failure of a business partner d_k^m (Section VII-B.4). Once in the state $completed$, business partners of type d_k^m can leave the coordination as they won't be asked to compensate their execution. Depending on the transactional requirements defined for C, business partners d_k^v may leave the critical zone before the end of the critical zone execution. A business partner 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 business partner d_k^v to leave the coordination is therefore stated as follows.

Theorem 7-1. *A partner d_k^v assigned to a vertex c_l can leave the execution of a critical zone C iff the partner d_k^v is in the state $completed$ and $\forall\ i \in [1, n]$ such that a business partner 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 $TS(C_d)$.*

3) Failure of a business partner d_k^v: This scenario is only possible with business partners of type (r_l, p). We can encounter two situations: either the failure is total and the business partner is not able to communicate any longer or the business partner is still alive and can forward a failure message to d_1^v. Figure 13 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 business partners. It should be noted here that business partners of type (r_l, r_t) 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 business partners of type (r_l, r_t) is transparent to the rest of the coordination and does not have to be handled.

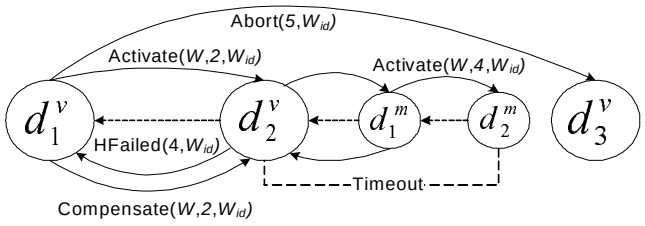

Fig. 14. Failure of a business partner d_k^m

4) Failure of a business partner d_k^m: The failure of a business partner d_k^m is detected by its subcoordinator with a timeout. As specified in the transactional model, we indeed consider that business partners of type d_k^m can only fail because of hardware problems and failure of such partners therefore implies a loss of contact with their coordinator. The failure of a partner d_k^m is reported by its subcoordinator to the partner d_1^v. The failure detection and forwarding of the $Hfailed$ message are depicted in figure 14.

C. 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 partners d_k^v and d_k^m and how coordination decisions are made upon detection of a failure.

1) Replacement of failed partners d_k^v: During the course of the execution new partners can be discovered and assigned to vertices in order to replace failed ones. In fact two situations can happen: either the failure of a partner occurs while executing its assigned vertex or the coordinator loses contact (timeout detection) prior to the activation of the partner. The first situation is specified in the previous section and no backup solution is possible as the data modified by the failed business partner are in an unknown state. In the second situation, it is possible on the contrary to assign a new business partner matching the transactional requirements to the vertex which has not yet started with the execution. Once the loss of contact with a business partner d_k^v is detected, no coordination decision is yet sent to business partners and the execution continues. If no business partner is found to be assigned to the vertex when its execution should be activated, the protocol coordinator considers the business partner it has lost contact with as failed.

2) Replacement of failed business partners d_k^m: In case of failure of a business partner 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 business partner so that the execution can go on. This is possible as on the one hand the partners d_k^m only modify volatile data and on the other hand, we have a backup copy of the data modified by the partners that are part of the abstract vertex $d_{l,p}^m$. Once the failure is detected, the subcoordinator of the failed partner tries to assign a new partner to the failed vertex. In this case, only volatile data are being modified, transactional requirements is not a concern and the assignment procedure can be repeated till a business partner manages to execute the requested vertex.

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 $CS(C_d, ts_k(C), (v_{a_i})_{i\in[1,z_a]}, c_a)$ 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.

D. Discussion

The coordination protocol integrates the semantic description of involved business partners 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 business partners characteristics which is not the case with recent efforts [14],[15].

The organization of the coordination is based on a simple hierarchical approach as in BTP [16]. In that respect, the central point of the coordination is the business partner d_1^v on which relies the whole coordination. This is the main weakness of the protocol, as a failure of this business partner would cause the complete failure of the workflow execution. The role of critical zone initiator of the coordination is therefore reserved to business partners 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, business partner 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 [17]. 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 [8], [18].

VIII. Implementation

In this section an implementation of the work presented in this paper is described. The overall system architecture is depicted in figure 15. The basic pervasive workflow infrastructure spans over the business partners taking part in a workflow

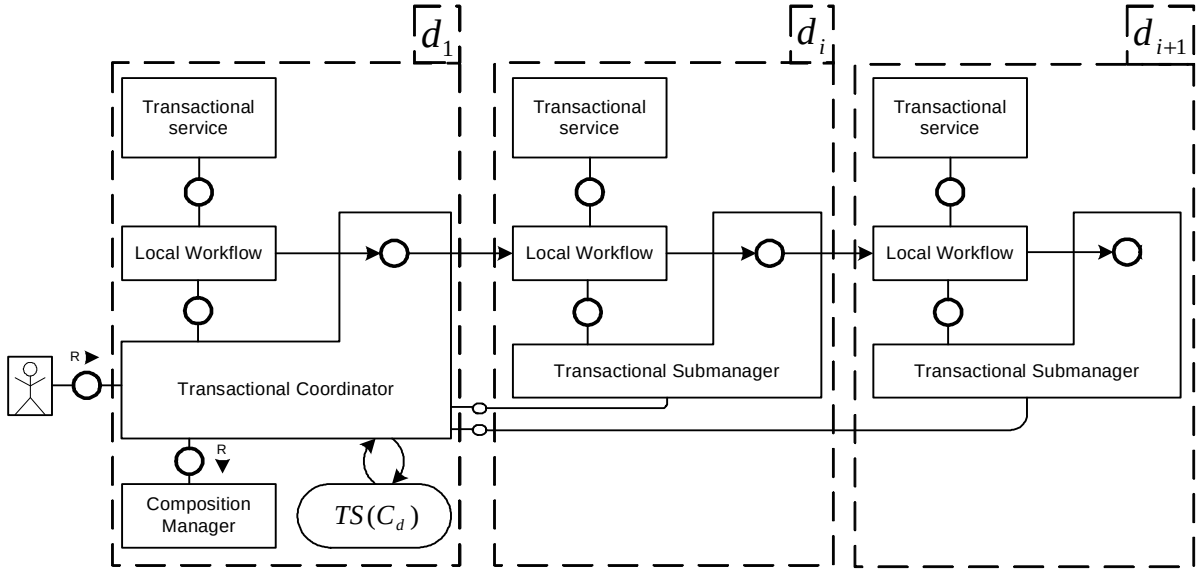

Fig. 15. Architecture

instance. A local workflow engine developed on top of BPEL [8] is in charge of handling, for each involved business partner, the workflow management and control tasks which mainly consist of:

- receiving and forwarding workflow requests
- issuing discovery requests
- invoking the appropriate local services to execute workflow tasks

In order to support the execution of pervasive workflows, we implemented in the fashion of the WS-Coordination initiative [19] a transactional stack composed of the following components:

- Transactional coordinator: this component is supported by a critical zone initiator. On the one hand it implements the transaction-aware business partner assignment procedure as part of the composition manager module and on the other hand it is in charge of assuring the coordinator role of the transactional protocol relying on the set $TS(C_d)$ outcome of the assignment procedure.
- Transactional submanager: this component is deployed on the other partners and is in charge of forwarding coordination messages from the local workflow to the appropriate subcoordinator or coordinator and conversely.

In the remainder of this section, we focus on the implementation of the transaction-aware partner assignment procedure.

A. OWL-S transactional and functional matchmaker

To implement the assignment procedure presented in this paper we augmented an existing functional OWL-S matchmaker [20] with transactional matchmaking capabilities. In order to achieve our goal, the matchmaking procedure has been split into two phases. First, the functional matchmaking based on OWL-S semantic matching is performed in order to identify subsets of the available partners that meet the functional requirements for each workflow vertex. Second, the implementation of the transaction-aware partner assignment procedure is run against the selected sets of partners in order to build an acceptable instance fulfilling defined transactional requirements.

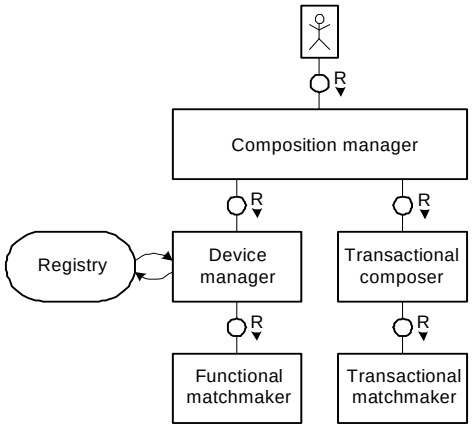

Fig. 16. OWL-S transactional matchmaker

The structure of the matchmaker consists of several components whose dependencies are displayed in figure 16. The composition manager implements the matchmaking process and provides a Java API that can be invoked to start the selection process. It gets as input an abstract process description specifying the functional requirements for the candidate partners and a table of acceptable termination states. The registry stores OWL-S profiles of partners that are available. Those OWL-S profiles have been augmented with the transactional properties offered by business partners. This has been done by adding to the non-functional information of the OWL-S profiles a new element called *transactionalproperties* that specifies three Boolean attributes that are retriable, reliable and compensatable as follows:

```
<tp:transactionalproperties retriable="true"
                            reliable="true"
                            compensatable="true"/>
```

In the first phase of the selection procedure, the business partner manager is invoked with a set of OWL-S profiles that specify the functional requirements for each workflow vertex. The business partner manager gets access to the registry, where all published profiles are available and to the functional matchmaker which is used to match the available profiles against the functional requirements specified in the workflow. For

each workflow vertex, the business partner manager returns a set of functionally matching profiles along with their transactional properties. The composition manager then initiates the second phase, passing these sets along with the process description, and the table of acceptable termination states to the transactional composer. The transactional composer starts the transaction-aware business partner assignment procedure using the transactional matchmaker by classifying first those sets into six groups:

- sets including only business partners of type (ur_l,r_t)
- sets including only business partners of type (r_l,r_t)
- sets including only business partners of type (r_l,p)
- sets including only business partners of type (r_l,c)
- sets including business partners of types (r_l,r_t) and (r_l,c)
- sets including business partners of type (r_l,$r_t c$)

Once those sets are formed the iterative transactional composition process takes place as specified above based on the table of acceptable termination states. Depending on the set of available services and the specified acceptable termination states, the algorithm may terminate without finding a solution.

IX. Related work

Transactional consistency and correctness of distributed systems such as database systems has been an active research topic over the last 15 years [21], [22], [23] yet it is still an open issue in the area of distributed processes within the Service Oriented Computing paradigm (SOC) [24], [25], [26], [27]. In this paper we specified a transactional protocol for the pervasive workflow architecture presented in [4] and our solution uses and extends the results proved in [9].

The execution of distributed processes wherein business partners are not assigned at design time raises new requirements for transactional systems such as dynamicity, semantic description and relaxed atomicity. Existing transactional models for advanced applications and workflows [5] do not offer the flexibility to integrate these requirements [28]. Our solution allows the specification of transactional requirements supporting relaxed atomicity for an abstract workflow specification and the selection of semantically described business partners or services fulfilling the defined transactional requirements. In addition, we provide the means to compute a coordination protocol suited to the workflow instance resulting from our business partner assignment procedure.

In [10], 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 some means to verify the consistency of composite services but it does not take into account transactional requirements 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, transactional requirements do not play any role in the component business partners 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 business partner assignment procedure. A transactional web service composition framework is also presented in [29] yet this approach does not allow to define coordination strategies as fine-grained as the termination state model underpinning our composition algorithm.

The transactional protocol we propose offers suitable means to respond to the constraints introduced by environments where heterogeneous business partners share resources in a collaborative manner. Using relaxed atomicity features, the protocol indeed offers the flexibility for business partners to release their resources as soon as their participation to the workflow is no longer required. Moreover using a flexible semantic, business partners are able to advertise their capabilities so that they can assume a role suited to any workflow in which their resources can be used. Current efforts in the design of transactional framework supporting the coordination of business processes [14], [15], [30], [31] do not offer such flexibility. They suffer from the lack of tools for the specification of transactional requirements and their integration into a dynamic business partners' selection process. As opposed to our solution the WS-BA specification and its implementation [32], for instance, do not provide designers with the adequate means to specify the business logic associated with their long running transactions. Furthermore, no recovery procedure is specified as part of the protocol for the replacement of partners in case of failure.

X. Conclusion

We presented an adaptive transactional protocol developed in the scope of the pervasive workflow model [4]. The contributions of the paper are threefold. First we provide a transactional model that captures the typical transactional properties associated with Web services and make it possible for business partner to advertise the latter as a non functional attribute to potential clients. This transactional model and its associated semantic are actually the core of the overall approach and consider the transactional properties offered by business partner as part of the SLA. Second we propose a composition algorithm whose goal is to mix the transactional properties offered by business partners in order to meet some transactional constraints identified by workflow application designers. This algorithm does not only build consistent workflow instances but also provides the coordination rules to adequately coordinate the workflow execution. These rules are directly derived from the requirements set by designers in the first place but also from the composite application outcome of the assignment procedure. Third we propose a transactional protocol that meet the dynamicity requirements introduced by a flexible execution environment. We believe that our approach can be used to augment recent specifications [19] in increasing their flexibility to incorporate transactional properties of business partners in the definition of adaptive coordination rules. Besides, a complete transactional framework has been implemented as a proof of concept of our theoretical results. Future work will focus on the design of security solutions for the pervasive workflow model.

XI. Appendix

Table of notations

t_i	Vertex
d_i	Business partner assigned to t_i
W	Workflow

W_d	Workflow instance
C	Critical zone
C_d	Instance of C
ats	Acceptable Termination State
ts	Termination State
m_k	Vertex that modifies only volatile data
v_k	Vertex that modifies data other than volatile ones
d_k^v	Business partner assigned to v_k
d_k^m	Business partner assigned to m_k
r_l	Reliable
r_t	Retriable
ur_l	Unreliable
p	Pivot
c	Compensatable

REFERENCES

[1] 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*, July 2004, pp. 288–295.

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

[3] 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.

[4] 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.

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

[6] 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.

[7] "Owl-s specifications," http://www.daml.org/services 2003.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

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

[15] ——, "Ws-businessactivity," 2005.

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

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

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

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

[20] "Parameterized semantic matching for workflow composition," IBM, Tech. Rep. RC23133, 2004.

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

[22] S. Lu, A. Bernstein, and P. Lewis, "Automatic workflow verification and generation," *Theor. Comput. Sci.*, vol. 353, no. 1, pp. 71–92, 2006.

[23] "Correct execution of transactions at different isolation levels," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1070–1081, 2004, member-Shiyong Lu and Fellow-Arthur Bernstein and Fellow-Philip Lewis.

[24] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana, "The next step in web services," *Commun. ACM*, vol. 46, no. 10, 2003.

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

[26] M. Little, "Transactions and web services," *Commun. ACM*, vol. 46, no. 10, pp. 49–54, 2003.

[27] S. Tai, R. Khalaf, and T. Mikalsen, "Composition of coordinated web services," in *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, New York, NY, USA, 2004, pp. 294–310.

[28] 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.*

[29] M.-C. Fauvet, H. Duarte, M. Dumas, and B. Benatallah, "Handling transactional properties in web service composition." in *Web Information Systems Engineering - WISE 2005, 6th International Conference on Web Information Systems Engineering, New York, NY, USA*, 2005, pp. 273–289.

[30] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web services: Concepts, Architectures, and Applications*. Springer Verlag, 2003.

[31] M. P. Papazoglou, "Web services and business transactions," *World Wide Web*, vol. 6, no. 1, pp. 49–91, 2003.

[32] "Apache kandula," http://ws.apache.org/kandula/.

Frederic Montagut is a PhD candidate at SAP Research in Mougins, France. Frederic obtained his engineering diploma from Telecom INT, France and the Diploma of Advanced Studies (M.Sc.) in Network and Distributed Systems from Nice University, France in 2004. He joined SAP Research in October 2004 working under the supervision of Pr Refik MOLVA, Eurecom Institute. His research interests range from distributed workflow systems, workflow coordination to workflow security.

Refik Molva is a full professor and the head of the Computer Communications Department at Eurecom Institute in Sophia Antipolis, France. His current research interests are in security protocols for self-organizing systems and privacy. He has been responsible for research projects on multicast and mobile network security, anonymity and intrusion detection. Beside security, he worked on distributed multimedia applications over high speed networks and on network interconnection. Prior to joining Eurecom, he worked as a Research Staff Member in the Zurich Research Laboratory of IBM where he was one of the key designers of the KryptoKnight system. He also worked as a security consultant in the IBM Consulting Group in 1997. Refik Molva has a Ph.D. in Computer Science from the Paul Sabatier University in Toulouse (1986) and a B.Sc. in Computer Science (1981) from Joseph Fourier University, Grenoble, France.

Silvan Tecumseh Golega is a master student at the Hasso-Platner-Institut of the Universitt Potsdam, Germany. Silvan received a B.Sc. from the Hasso-Platner-Institut in 2006 after spending one year at the Universidad Rey Juan Carlos in Madrid, Spain. Silvan performed the practical work of his master thesis under the supervision of Frederic Montagut at SAP Research France on the topic "Composition and Coordination of Transactional Business Processes". He contributed to the work presented in this paper during his stay at SAP Research.