View Driven Interorganizational Workflows

Amirreza Tahamtan*

Vienna University of Technology,
Dept. of Software Technology & Interactive Systems,
Information & Software Engineering Group,
Favoritenstrasse 9-11/188,
A-1040 Vienna, Austria
E-mail: tahamtan@ifs.tuwien.ac.at
*Corresponding author

Johann Eder

Alpen-Adria University of Klagenfurt,
Dept. of Informatic-Systems,
Universitaetsstrasse 65-67,
9020 Klagenfurt, Austria
E-mail: eder@isys.uni-klu.ac.at

Abstract: We propose a layered architecture for choreographies and orchestrations of web services. The proposed architecture uses the concept of process views. The distributed nature of the model and the concept of views improve the privacy of business partners but do not limit their interaction capabilities, an essential feature in B2B and interorganizational applications. Our approach enables description of business processes in different levels of detail with a uniform modeling language and is fully distributed.

Keywords: Web Service Composition, Choreography, Orchestration, Workflow View, Interorganizational process, Abstraction, Aggregation

Reference to this paper should be made as follows: Tahamtan, A. and Eder, J. (xxxx) 'View Driven Federation of Choreographies', *Int. J. Intelligent Information and Database Systems (IJIIDS)*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Amirreza Tahamtan is an assistant professor at the Dept. of Software Technology & Interactive Systems at TU-Vienna. He received his M.Sc. and Ph.D. in computer science both with distinction from TU-Vienna and University of Vienna. From 2001 till 2005 he has been a research assistant at the Medical University of Vienna, from 2005 till 2009 researcher and post doctoral fellow at the Dept. of Knowledge and Business Engineering, University of Vienna and since 2009 he works at the Vienna University of Technology. He is author of several peer-reviewed research articles including book, journal articles and conference and workshop proceedings. He has served as a member of program committee of different international conferences and workshops and reviewer of international journals.

Johann Eder is full professor for Information and Communication Systems at the University of Klagenfurt, Austria. Since 2005 he is vice president of the Austrian Science Funds (FWF), heading the department of natural sciences and technology. He successfully directed several funded research projects e.g. in the fields of workflow management systems, temporal data warehousing, interoperability, information systems for biobanks, self-healing web services. He coauthored one book, edited 21 books/proceedings/special issues of journals and published more than 140 peer reviewed papers. He served as chair and member in numerous program committees for international conferences and as referee or editorial board member for international journals. He held positions at the Universities of Vienna and Klagenfurt and visiting positions at the University of Hamburg and with AT&T Research.

1 Introduction

Web Services enable application development and integration over the Web by supporting interactions and business processes within and across the boundaries of organizations. Two mostly used concepts in the realm of Web Service composition are choreographies and orchestrations.

An orchestration belongs to and is controlled by one partner and describes an executable process which is run by its owner. A partner's internal logic is contained in his orchestration. An orchestration is solely visible to its owner and other external partners have no view on and knowledge about it. An orchestration is a process viewed only from the perspective of its owner. Different languages such as WS-BPEL executable process or BPML can be used for definition of orchestrations.

On the other hand, a choreography is a non-executable, abstract process that defines the message exchange protocol and collaboration among partners. Exchanged messages are visible to all participants. External parties are not able to view and monitor the messages and have no view on the choreography. A choreography has no owner or a super user in charge of control and all partners are treated equally. A choreography is a shared process definition from a global perspective [23].

A typical scenario [2, 10, 12] of web service composition assumes one choreography shared among several partners where each partner realizes its parts of the choreography in its orchestration. The shared choreography defines the communication among orchestrations. This scenario is depicted in figure 1.

This typical scenario misses an important facet. Presence of only one choreography is not adequate for all real life applications and more choreographies may be necessary for implementation of a business process. Besides, one choreography implies that all interactions are visible to all partners. Even if the combination of all choreographies into one choreography would be possible, the separation offers obvious advantages. Unless two partners do not set up a separate choreography they cannot hide their interactions from other partners. In addition, orchestrations must be allowed to take part in more than one

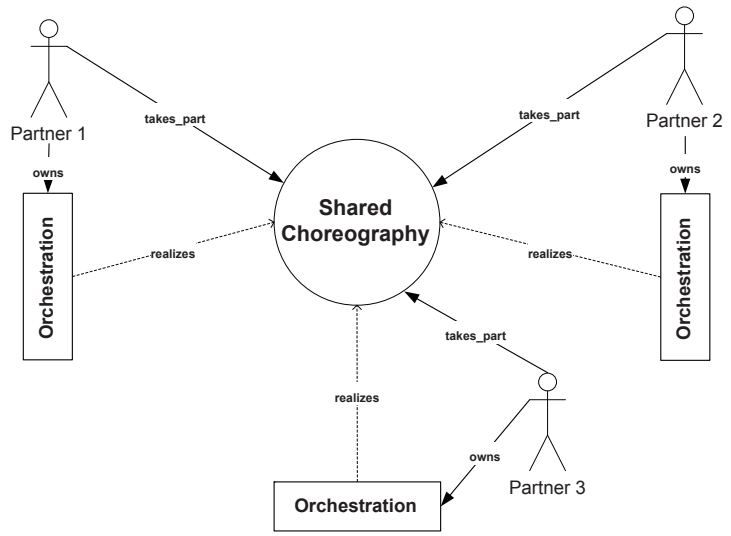

Figure 1 A typical scenario of Web Service composition

choreographies. Imagine an organization which has implemented a detailed and complex orchestration to serve its different groups of partners. Its orchestrations must be able to take part in a choreography serving customers and in another choreography with providers.

To overcome the restrictions of the typical scenario, a modular architecture for definition of business processes is needed which at the same time caters for the privacy needs of participating partners. The contribution of this paper is a new architecture and a nouvelle approach called *view driven federated choreographies*. In this approach a combination of choregraphies and orchestrations together implement a business process. In other words the whole process logic is distributed among different independent but cooperating business processes in a distributed manner without any need for a centralized control. Our approach allows, as well, for access restriction and information hiding through process views. A partner can restrict the access to its underlying orchestration such that its private data and steps are hidden without affecting the interaction with other partners. Finally we introduce a more general architecture for other use cases and present a meta model for uniform modeling of choregraphies and orchestrations.

2 A Motivating Example

In order to show the advantages of our approach we use a web shopping scenario as a motivating example. When shopping online a buyer takes part in a choreography whose partners are a buyer, a seller like Amazon and a shipper like FedEx. The buyer knows the following partners and steps: the buyer orders something at the seller and receives the items from a shipper. Figure 2 shows such an exemplary choreography. The partners' orchestrations have additional activities which are not

contained in the shared choreography. The buyer's orchestration is depicted in figure 3. The buyer before making a request for quote, searches for available sellers for his requested item and consequently selects one, activities *Search sellers* and *Select seller* in the buyer's orchestration.

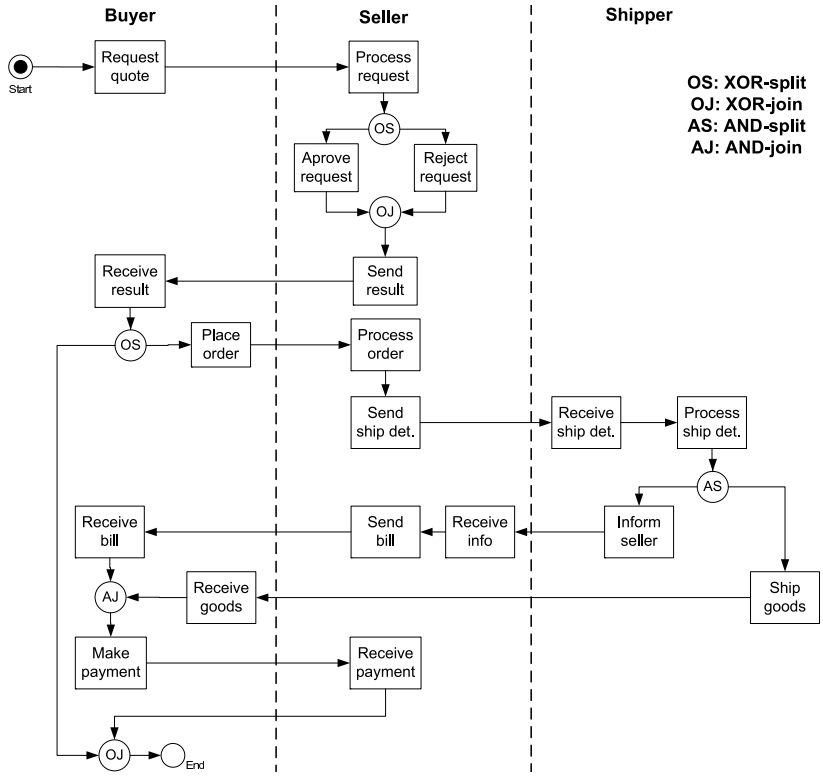

Figure 2 The shared choreography between buyer, seller and shipper

At the same time the seller takes part in several other choreographies which are not visible to the buyer, e.g. the seller and the shipper realize another protocol containing other actions such as money transfer from the seller's bank to the shipper for balancing shipment charges.

As this example shows, more than one choreography may be needed for reaching the goals of a business process. Besides, two partners involved in one choreography may also take part in another choreography that is not visible to other partners of the choreography, however essential for the realization of business goals. All these choreographies overlap in some parts but cannot be composed into one single global choreography. Moreover, such choreographies must be realized by orchestrations of partners that take part in them. In the above example the seller implements an orchestration enacting the different interaction protocols other partners.

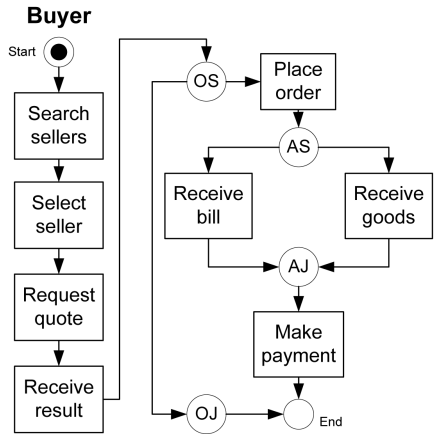

Figure 3 The buyer's orchestration

3 Related Work

Interorganizational workflows have attracted a lot of research. [18] presents an architecture for using web technology for the cooperation of workflow management systems. [11] considers loosely coupled interorganizational workflows with a a global source place and a global sink place. IO-soundness is considered as correctness criteria. [7] proposes a model for interaction among workflows based on send and request nodes. [26] introduce an architecture for cross-organizational applications. Their model is two-tiered. The first tier consists of private business processes and the second tier contains shared business processes. Authors in [19] present an approach for modeling choreography for B2B applications based on a predefined set of interoperability patterns including contract processes, executable processes and interface protocols. [32] proposes a technique for inter-enterprize business collaboration based on service choreography based on a centralized global business process and its decomposition into subprocesses. Rinderle et al. in [24] deal with the controlled evolution of process choreographies i.e. the problem of change of interaction structure among partners, e.g. when a change in a private orchestration shall be propagated to the choreography. Authors in [25] represent a top-down approach for integrated modeling of business processes. The proposed approach consists of seven steps: building a centralized process containing all tasks, building swimlane partitions of the centralized process according to participants, identification of control transfer within and between swimlanes, extraction of distributed process models for each partition, transformation of activities into sender-receiver activities and finally deployment of the process. Schulz and Orlowska in [27] propose an application of workflow views as a mean for realizing communication and cooperation in an interorganizational setting. A workflow view is an abstraction of the corresponding private workflow and reflects the communication requirements of the shared workflow. [21] proposes an order preserving approach for constructing views. A workflow view is composed of an aggregation of base activities of a workflow. Chebi, Dustdar and Tata in [8]

propose an SOA-based approach for interconnection and cooperation of workflows. In this work workflow views contain only tasks that either send or receive data from or to external workflows. [9] proposes the application of workflow views for interoperability in cross-organizational workflows. Views restrict the access on a workflow and conceal the internal, private or unnecessary information. A workflow view is defined as a structurally correct subset of a workflow definition. In [22] authors present a technique for designing workflow views and identification of relevant tasks included in a workflow view for a specific role. In their approach, workflow views are visible part of a process for a specific role. Shan et al. in [28] introduce an object oriented approach for workflow views called the object deputy model. A deputy object has its own persistent identifier and may have additional attributes and methods that are not derived from its source objects. Moreover, there is a bilateral link between objects and one of its deputy objects, which allows not only for inheritance but also for update propagations between them. [17] provides an approach for constructing optimized process views. The authors apply a two-phase approach based on inheritance for construction of process views. In the second phase it is necessary that process views be exposed to the external partner. Bobrik at al. in [4, 5] propose a technique for constructing views aimed at process visualization. The limitation of their approach is that they consider no loops and provide no proof for correctness verification of resulting process views. For a comparison of our approach with the related works and how our approach improves them please refer to [29] and [30] respectively.

4 View Driven Federated Choreographies

View driven federated choregraphies overcome the limitations of the typical web service composition scenario and offer several advantages. This architecture is an extension of our previous work [13, 31] by the concept of views. A more general approach is introduced in section 6. Note that in this work choreographies, orchestrations and view are modeled as a directed acyclic graph $G = (N, E)$, where N denotes the set of nodes and E the set of edges. Nodes correspond to activities and control nodes and edges to the dependencies between activities and control nodes. We limit the workflow model to full-blocked workflows [1], where each split-node or start of loop has a counterpart join-node or end of loop and each outgoing path of a split node or start of a loop ends eventually in a join node or end of a loop and vice versa. Reasons for choosing full-blocked workflows include avoidance of structural failures and its application in dominant industry standard WS-BPEL. A control node is either an AND-split (AS), AND-join (AJ), XOR-split (XS), XOR-join (XJ), start of loop (LS) or end of loop (LE). The activities contained between LS and LE are repeated as long as the exit condition is not satisfied. If the exit condition is satisfied, the loop terminates and the execution of the flow continues with the immediate successor activity of LE. In this work we use a generic graph notation which can be translated and mapped in a straightforward manner onto other notations. [29] describes the mapping of our notation onto petri-nets.

Consideration of views has several advantages:

Improvement of privacy: Views improve the privacy of partners in a business process. By using views, partners can decide which parts of their internal

private process can be exposed to external partners and there is no need for exposing the whole internal logic and thus preventing cooperating partners to become competitors.

Serving different (groups of) partners: One single private process can have many views and each view can be used for interaction with a partner or a group of partners. By application of views, one single underlying private process can serve different groups of partners.

Interaction compatibility: Partners can change and modify their private processes without any effect on the interaction with other partners, as long as the views remain the same, i.e. they can be defined on the new processes as well. In this way the interaction can be performed in a consistent way and there is no need to inform other partners about changes in the private process.

Similar to object oriented programming, views provide a mean for information hiding. Using the view constructing techniques (described in section 5) it is possible to hide and pack what is meant to be hidden and restrict the access only to specific parts of a process. Further, views provide interfaces to a process for client interaction such that interaction with clients is implementation independent.

Each partner has an orchestration for realization of its tasks and takes part in one or more choreographies. Each partner has a view on his/her orchestration. A view is not only a view on an orchestration but also a view on the shared choreography and identifies the parts that belong to a specific partner of the shared choreography whose realization this partner is in charge of. By realization of the activities that belong to a partner, the partner has a conformant behavior with respect to the agreed upon choreography. A view shows a single partner's perspective on the choreography and can be used as a skeleton for designing the partner's orchestrations by adding other internal tasks. In other words they show the minimum amount of tasks as well as the structure of the tasks that a partner's orchestration must contain in order to be conformant with the shared choreography. For a more detailed discussion on views and how correct views can be constructed refer to section 5.

The main idea of the view driven federated choreographies is presented in figure 4. It consists of two layers. The upper layer consists of the federated choreographies shared between different partners, e.g. in figure 4 *Purchase processing choreography* is shared between buyer, seller and shipper. A choreography is composed of views of the orchestrations by which the choreography is (partially) realized, i.e. the activities contained in a choreography are only those in the views. A choreography may support another choreography. This means the former, the supporting choreography, contributes to the latter, the supported choreography, and partially elaborates it. E.g. *Shipment processing choreography* is the supporting choreography and *Purchase processing choreography* is the supported choreography. The set of activities contained in a supporting choreography is an extended subset of the activities of the supported choreography. The supporting choreography describes parts of the supported choreography in more detail. The choreography which supports no other choreography and is only supported by other choreographies is called the *global choreography*, in our

running example *Purchase processing choreography* is the global choreography. Informally, the global choreography captures the core of a business process and other choreographies which support the global choreography describe parts of the global choreography in the needed detail for implementation. In figure 4 the global choreography, *Purchase processing choreography*, describes how an item is sold and shipped to the buyer. It contains the activities and steps which are interesting for the buyer and the buyer needs to know them in order to take part in or initiate the business process. How shipping of the items and debiting the buyer's credit card is handled in reality are described in the *Shipment processing choreography* and the *Payment processing choreography* respectively.

The bottom layer consists of orchestrations that realize the choreographies in the upper layer. Each orchestration provides several views for different interactions with other partners. The interactions with other partners are reflected in the choreographies. Hence, an orchestration needs to provide as many views as the number of choreographies this orchestration (partially) realizes. Let figure 2 be the *Purchase processing choreography*. The buyer's orchestration and its shared view with the *Purchase processing choreography* are presented in figures 3 and 5 respectively. Each partner provides its own internal realization of relevant parts of the according choreographies, e.g. buyer has an orchestration which realizes its part in all three choreographies.

The presented approach is fully distributed and there is no need for a centralized coordination. Each partner has local models of all choreographies in which it participates. All local models of the same choreography are identical by which partners know to which activities they have access, which activities they have to execute and in which order. In addition, partners are aware when they can expect and send messages. In other words, the knowledge about execution of the model is distributed among involved partners and each partner is aware of its duties. Hence, there is no need for a super-user with the whole knowledge about process execution. Note that if there is a link between two chorographies and/or orchestrations, either a support link between two choreographies or a realize link between a choreography and an orchestration, it implies that these two choreographies and orchestrations have at least one activity in common.

In fact, one can argue that supporting choreographies may be combined by the means of composition as described e.g. in [6] where existing choreography definitions can be reused and recursively combined into more complex choreographies. But, in fact, the relationship between choreographies can be more sophisticated than merely a composition. For example the relationship between the seller and the shipper may include not only the passing of shipment details from the seller to the shipper but also payment of shipment charges through the seller's bank. This can be described by a separate choreography between seller, shipper and the bank. This choreography has additional activities and partners which are not visible in its supported choreography. This choreography contributes to the *Purchase processing choreography* and elaborates the interaction between the seller and the shipper. The *Shipment processing choreography* is illustrated in figure 6.

Figure 4 The main idea of the view driven federated choreographies

4.1 Advantages of the View Driven Federated Choreographies

View driven federated choreographies are more flexible than typical compositional approaches used in proposals like WS-CDL and closes the gap between choreographies and orchestrations by providing a coherent and integrated view on both choreographies and orchestrations. View driven federated choreographies offer obvious advantages such as:

Protection of business know-how: View driven federated choreographies improve business secrecy and protect business know-how. If the whole business process is modeled as one single choreography, all message exchanges are visible to all partners. But if the interactions are separated into different choreographies, other external observers have no knowledge about the message exchanges and partners can keep the actual handling of their business private.

Avoidance of unnecessary information: The proposed approach avoids unnecessary information. Even if there is no need for protection of business

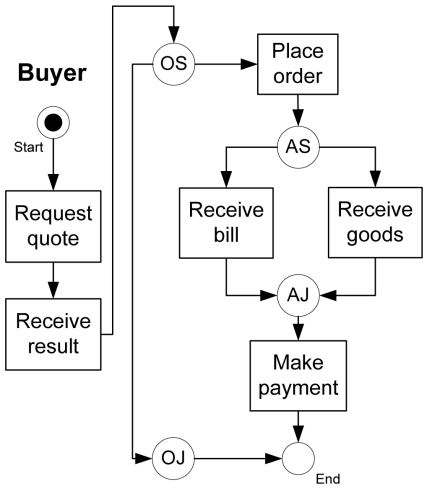

Figure 5 The view on the buyer's orchestration

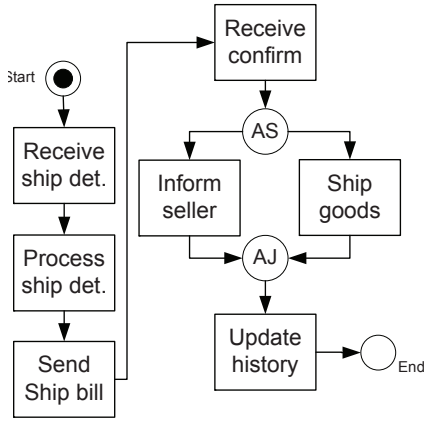

Figure 6 The Shipment processing choreography

know-how, it is desirable to separate choreographies and limit them only to the interested parties.

Extendability: The model is extendable, when such a need arises. As long as the conformance conditions are satisfied, the model can be extended and there is no need to interfere with the running process and notifying the partners for setting up new choreographies. The conformance issues are discussed briefly in subsection 6.1.

Uniform modeling: Finally, view driven federated choreographies use a uniform modeling for both choreographies and orchestrations and eliminates the need for different modeling languages and techniques for choreographies and

orchestrations. The uniform modeling technique reduces the cost of process at design phase.

5 Construction of Views

We consider two ways for construction of views: *Abstraction* and *Aggregation*.

In order to construct correct views on workflows it is necessary that activities in a view have the same ordering as in the underlying workflow. In other words, a view on a flow can not change the ordering of the activities of the underlying workflow.

Definition 5.1: ***(Correctness of Views)***
A workflow $G\prime = (N\prime, E\prime)$ is a correct view of a workflow $G = (N, E)$ if and only if the following properties hold:

(a) *$G\prime$ is a valid full-blocked workflow definition*

(b) *The nodes contained in $G\prime$ are a subset of the nodes of G or represent a subset of the nodes of G*

(c) *$\forall$ nodes $a, b \in N \bigcap N\prime : [a > b]_{G\prime} \Leftrightarrow [a > b]_G$, where $[a > b]_G$ denotes that there is a path from node a to node b in the graph G and $N\prime \subset N$*

A view $G\prime$ of a workflow G with the property (c) is called an order preserving view of G. An order preserving view does not change the order of the nodes of the underlying workflow.

5.1 Abstraction Operator

By application of the abstraction operator (compare τ-operator in process algebra [3]) it is possible to make parts of a process invisible to external observers. The abstraction operator is like a renaming operator that renames the label of a node and makes it invisible from outside. The abstraction operator provides a mean for construction of views. By application of this operator parts of a process can be made unobservable. This may be because the hidden parts of the process do not contribute to the interaction with another partner, are not interesting for external partners or are intended to be hidden because of privacy issues. The rest of the process can be exposed as a view on the process.

Definition 5.2: ***(Abstraction Operator)***
Let $G = (N, E)$ be a workflow. Application of the abstraction operator on a node j of a workflow G results in a workflow $G\prime = (N\prime, E\prime)$, denoted as $ABS(G, j) = G\prime$

case 1:** j **is an activity
Let j be an activity,
$ABS(G, j) = G\prime = (N\prime, E\prime)$ *with*
$N\prime = N - \{j\}$,
$E\prime = \{(n_s, n_t) | [(n_s, n_t) \in E \bigwedge n_s \neq j \bigwedge n_t \neq j]\}$
$\bigcup \{(n_s, n_t) | [(n_s, j) \in E \bigwedge (j, n_t) \in E]\}$

case 2: *j* ***is a control node***

Let j be a control node and $j\prime$ its counterpart control node,
$|\{(n_s, n_t) \mid (n_s, n_t) \in E \bigwedge (n_s = j) \bigwedge (n_t \neq j\prime)\}| \leq 1$
$\Rightarrow ABS(G, j) = G\prime = (N\prime, E\prime)$ *with*
$N\prime = N - \{j, j\prime\}$,
$E\prime = \{(n_s, n_t) | [(n_s, n_t) \in E \bigwedge n_s \neq j \bigwedge n_t \neq j \bigwedge n_s \neq j\prime \bigwedge n_t \neq j\prime]\}$
$\bigcup\{(n_s, n_t) | n_t \neq j\prime \bigwedge [(n_s, j) \in E \bigwedge (j, n_t) \in E]\}$
$\bigcup\{(n_s, n_t) | n_s \neq j \bigwedge [(n_s, j\prime) \in E \bigwedge (j\prime, n_t) \in E]\}$

Abstraction of an activity j removes the activity j from the set of nodes N as well as all the edges whose source node or target node is the activity j from the set of edges E. An edge from the predecessor of the activity j to its successor is added to the set of edges E.

The precondition for abstraction of a control node is that at most one of its outgoing paths (or incoming paths of its counterpart) contains activities. All other paths between a control node and its counterpart must directly connect them. This is reflected by checking the cardinality of edges that do not directly connect j and $j\prime$. Activities of other paths must be previously abstracted as explained in case 1 of the definition 5.2. Abstraction of a control node results in simultaneous removal of the control node and its counterpart as well as all edges that directly connect them. Again here edges are added from the predecessor of the control node to its successor activity and from the predecessor of counterpart control node to the successor of counterpart control node. The properties for abstraction of control nodes ensure that after abstraction the resulting flow is again a valid full-blocked workflow. The view on the buyer's orchestration (figure 5) is constructed by application of the abstraction operator on the buyer's orchestration (figure 3). As it can be seen the activities *Search sellers* and *Select seller* as well as their corresponding edges are abstracted.

Proposition 5.3: ***(Properties of the Abstraction Operator)***

(a) *Abstraction operator is commutative, i.e.*
Let $G = (N, E), \forall$ nodes $a, b \in N$:
$ABS(ABS(G, a), b) = ABS(ABS(G, b), a)$

(b) *Abstraction operator is associative, i.e.*
Let $G = (N, E), \forall$ nodes $a, b, c \in N$:
$ABS(ABS(ABS(G, a), b), c) = ABS(ABS(ABS(G, c), b), a)$

(c) *Abstraction is an order preserving operation and does not change the order of remaining nodes, i.e. Let $G = (N, E)$ and $G\prime = (N\prime, E\prime)$ be the resulting graph after application of the abstractor operator on a node of G,*
$\forall$ nodes $a, b \in N\prime : [a > b]_{G\prime} \Leftrightarrow [a > b]_G$, where $[a > b]_G$ denotes that there is a path from a to b in G

(d) *The resulting graph $G\prime$ is a valid full-blocked workflow definition.*

5.2 Aggregation Operator

Another way of constructing workflow views is application of aggregation. By aggregation some activities are grouped into a so-called aggregated activity. Aggregation can be used for hiding the internal structure of a group of activities. The aggregated activity is consequently contained in the workflow view and is in charge of sending and receiving data to and from activities in the underlying workflow.

Definition 5.4: ***(Aggregated Activity)***
Let $G = (N, E)$ be a workflow. An aggregated activity A_{G_A} represents a graph $G_A = (N_A, E_A)$ with the following properties:

- *G_A is a connected subgraph of G*
- *G_A has a unique first node and a unique last node, i.e. G_A has only one incoming edge and only one outgoing edge*
- *If a split-node (join-node) or start of loop is in N_A, its counterpart join-node (split-node) or end of loop is also in N_A*

$N_A \subseteq N$ denotes the set of activities and control nodes in the aggregated activity and E_A the dependencies between activities and control nodes, where $E_A = \{(n_s, n_t) \in E | n_s, n_t \in N_A\}$.

Note that G_A is the graph representing internal structure of the aggregated activity and A_{G_A} identifies the aggregated activity.

Definition 5.5: ***(Aggregation Operator)***
Aggregation is an operator that groups a subset of nodes of a workflow into an aggregated activity. Let $G = (N, E)$ be a workflow and $G\prime = (N\prime, E\prime)$ the resulting graph after application of aggregation on some activities of G. Application of the aggregation operator on a workflow G results in a new workflow $G\prime$, denoted by $AG(G, N_A) = G\prime$.
$AG(G, N_A) = G\prime(N\prime, E\prime)$ with
$N\prime = N - N_A \bigcup \{A_{G_A}\}$,
$E\prime = \{(n_s, n_t) \in E | n_s \notin N_A \bigwedge n_t \notin N_A\} \bigcup \{(n_s, A_{G_A}), (A_{G_A}, n_t) | \exists (n_s, n_f), (n_l, n_t) \in E, n_f, n_l \in N_A\}$, where n_f denotes the first node of G_A and n_l its last node respectively.

All nodes $n \in N_A$ are removed from N and the aggregated activity A_{G_A} is added. All edges of the nodes contained in G_A are as well removed from the set of edges E and two edges are added: one edge from the predecessor of the first node of the aggregated activity to the aggregated activity and another from aggregated activity to the successor of the last node of the aggregated activity.

The above properties for N_A are required to ensure that the workflow after application of aggregation is still a valid full-blocked workflow definition. Figure 7 demonstrates an application of aggregation.

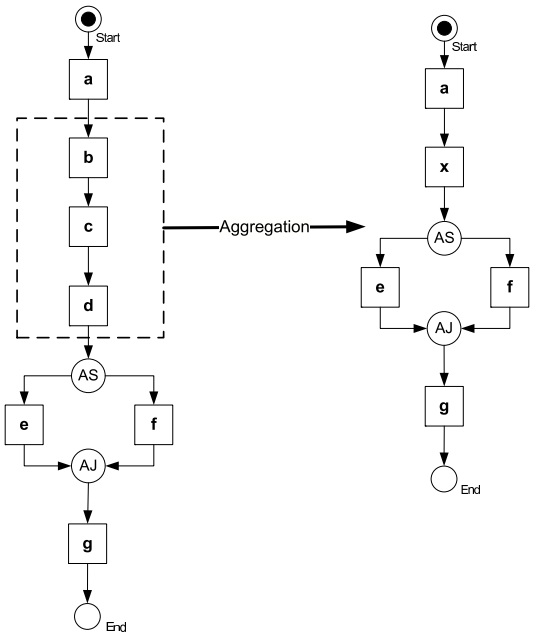

Figure 7 Application of the aggregation operator

The activities contained in an aggregated activity are not necessarily executable activities. Aggregated activities can again be aggregated into a new aggregated activity. In other words, aggregation can be applied in a recursive fashion. Aggregation is transitive in the sense that if an activity j is contained in an aggregated activity x and the aggregated activity x is itself contained in another aggregated activity y, then the activity j is contained in the aggregated activity y, $j \in x \bigwedge x \in y \Rightarrow j \in y$.

Proposition 5.6: ***(Properties of the Aggregation Operator)***

(a) *Let $G = (N, E)$ be a workflow and $G\prime = (N\prime, E\prime)$ the resulting workflow after application of aggregation on some activities of G and N_A the set of nodes of the aggregated activity A_{G_A}:*
$\forall a, b \in N$:
$[a > b]_G \bigwedge a, b \notin N_A \Rightarrow [a > b]_{G\prime}$
$[a > b]_G \bigwedge a \notin N_A \bigwedge b \in N_A \Rightarrow [a > A_{G_A}]_{G\prime}$
$[a > b]_G \bigwedge a \in N_A \bigwedge b \notin N_A \Rightarrow [A_{G_A} > b]_{G\prime}$

(b) $\forall a, b \in N\prime$:
$[a > b]_{G\prime} \bigwedge a, b \neq G_A \Rightarrow [a > b]_G$
$[a > A_{G_A}]_{G\prime} \Rightarrow \forall i \in N_A : [a > i]_G$

(c) *$G\prime$ is a valid full-blocked workflow definition*

5.3 Construction of Views

Application of the operators on a workflow guarantees a correct construction of view. The presented operators provide powerful tools for many purposes. Through abstraction a subset of the activities of the workflow or the internal activities within loops, parallel or conditional structures can be exposed in a view and aggregation provides tools for hiding the internal structures of a sub-workflow. As shown in the propositions 5.3 and 5.6 construction of views by application of abstraction or aggregation operator as defined in definitions 5.2 and 5.5 on a worklfow Let $N = (G, E)$ results in a graph $N\prime = (G\prime, E\prime)$ which is again a full-blocked workflow.

Theorem 5.7: ***(Aggregation and Abstraction Construct Correct Views)***
Let $N = (G, E)$ be a workflow. Application of abstraction operator or aggregation as defined in definitions 5.2 and 5.5 on N results in a graph $N\prime = (G\prime, E\prime)$ which is a correct view on N.

Definition 5.8: ***(View Constructor Operator)***
Let $G = (N, E)$ be a workflow and $G\prime = (N\prime, E\prime)$ a view on G. The view constructor operator, $VC(G) = G\prime$, is a sequence of abstraction or aggregation operators, denoted by $G \stackrel{\alpha}{\rightarrow} .. \stackrel{\alpha}{\rightarrow} .. \stackrel{\alpha}{\rightarrow} G\prime$, where α is either the abstraction operator or the aggregation operator.

After each application of α the workflow transforms to a new workflow. Note that the sequence is finite and can be empty. Obviously there is no unique way of constructing views rather the same view can be constructed by different sequence of operators. The view constructor is transitive in the sense that if $G\prime$ is a view on G and $G\prime\prime$ a view on $G\prime$ then $G\prime\prime$ is a view on G. However, the commutativity is not valid because different sequence of operators, produce different views. We have proved the propositions and theorems of this subsection in [30].

6 A More General Architecture for Interorganizational Workflows

In the view driven federated choreographies we have assumed that there is a support-link between two choreographies. The set of activities of a supporting choreography is an extended subset of its supported choregraphy. Here we relax this assumption and allow for a set of independent choreographies as depicted in figure 8. In this case, as well, an interorganizational workflow is composed of a set of choreographies and orchestrations. Choreographies, again, are abstract processes that define the message exchange protocols among involved partners and orchestrations are in charge of realizing the abstract activities defined in a choreography. In contrast to the view driven federated choreographies (compare figures 4 and 8) in this architecture choreographies are independent from each other. However, here partners again can take part in several choreographies and their according parts must be realized in their orchestrations. In addition, there

is no *global choreography* in the model. Remember that the global choreography is a choreography that is only supported by other choreographies and in turn supports no choreography. If there is only one choreography present in the model, we have the typical scenario(cf. figure 1), i.e. one shared choreography and a set of orchestrations, one for each partner. However, it must be noted that even if there is no direct link between two choreographies, two choreographies may indirectly affect each other. This is the case if one orchestration takes part in two different choreographies. In this scenario, the choreographies may affect each other indirectly through the shared activities with the orchestration, which is illustrated in figure 10. The indirect dependency of choreography is specially important for checking the temporal conformance of the model.

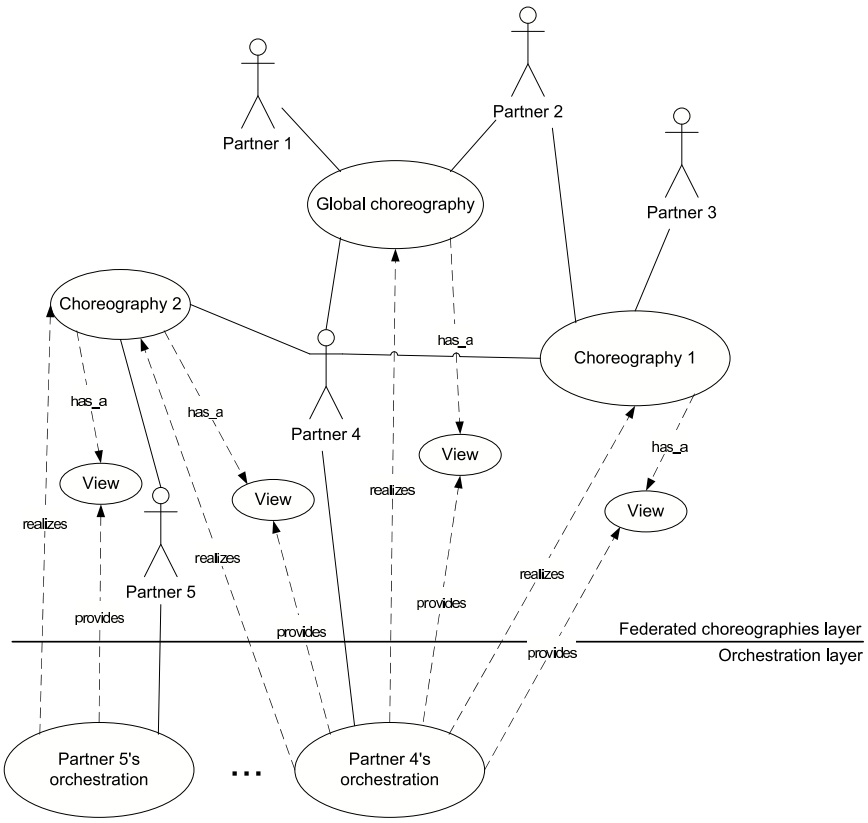

Figure 8 A typical scenario: a set of choreographies and orchestrations

6.1 Conformance Issues

The central requirement of the proposed model is inter-layer conformance (between choreographies and orchestrations) as well as intra-layer conformance i.e. inside the choreographies and orchestrations layers itself. This includes structural conformance [14], temporal conformance [16, 15], messaging conformance and

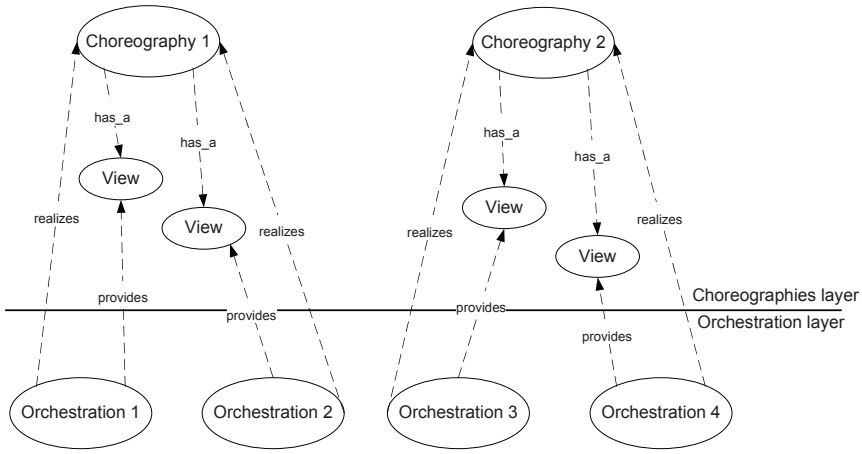

Figure 9 A set of independent choreographies

p

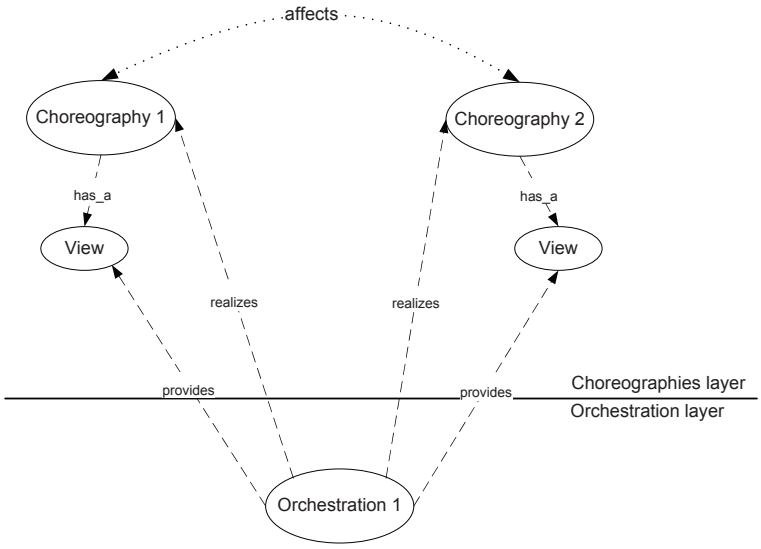

Figure 10 A set of temporally dependent choreographies

dataflow conformance. Conformance issues are out of scope of this work. For structural and temporal conformance please refer to the references in this subsection as well as [29].

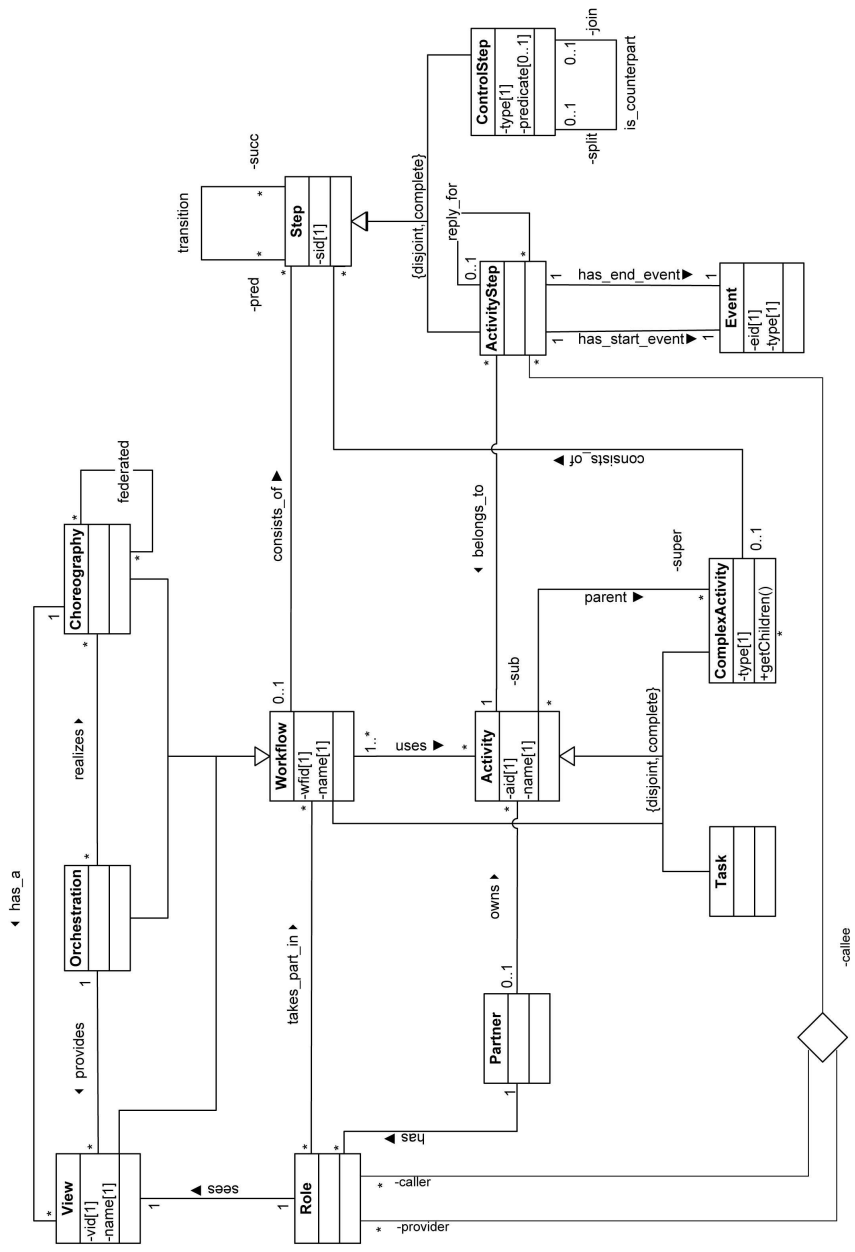

Figure 11 Metamodel of view driven federated choreographies

7 Metamodel of the View Driven Federated Choreographies

Choreographies, orchestrations and views are treated as workflows and can be modeled using typical workflow control flow structures. Choreographies can be federated into more complex ones. Choreographies are composed of views.

Moreover, as all choreographies are workflows, they can be composed out of other choreographies by means of complex activities and control structures available in the workflow models. The same applies to orchestrations. The metamodel allows one to describe several choreographies and orchestrations on different levels of detail. Choreographies and orchestrations can share the same activities. These activities are contained in a view that is provided by the orchestration. Such a view identifies which activities of the choreography must be realized in the orchestration. An activity visible in one choreography can be extended by its relationships with other activities in a federated choreography. On the other hand, an activity visible in a choreography can have a complex implementation described in an orchestration. Thus, choreographies and orchestrations together with their activities can be viewed on different levels of detail and in context of different relationships. The metamodel of the view driven federated choreographies is represented in figure 11.

A workflow is either a *choreography*, an *orchestration* or a *workflow view*. A workflow can have many *views*. A workflow defines views for different roles (of partners). Each role sees and accesses the workflow through the view. A workflow uses *activities*. An activity is either a *task* or a *complex activity* . An activity can be used to compose complex activities. An activity occurrence in such a composition is represented by an *activity step*. One activity can be represented by several activity steps in one or several workflows or complex activities and each activity step belongs to exactly one activity. In other words, activity steps are placeholders for reusable activities. The same activity can occur in different workflows. The control structure of a complex activity is described by its *type*.

An activity may be owned by a *partner*. Orchestrations and tasks must have an owner, whereas choreographies must not have an owner. A partner may have several *roles* and one role can be played by several partners. A role may take part in a workflow and call an activity step in this workflow. An activity step is provided by another role. Thus a single parter can use different roles to participate in a workflow and provide or call activity steps. A role sees and accesses a defined view on the workflow.

The notion of a *step* is very important for the presented metamodel. Both workflows and complex activities consist of steps. Between the subsequent steps there can be a *transition* from a predecessor to a successor which represents control flow dependencies between steps.

A complex activity can be decomposed in a given workflow into steps that constitute this complex activity only if all of the activities corresponding to these steps are also used and visible in this workflow. Therefore, a workflow can be decomposed and analyzed on different levels of detail with complex activities disclosing their content, but without revealing protected information on the implementation of these complex activities. To allow a correct decomposition, a complex activity must have only one activity without any predecessor and only one activity without any successor. The same applies to workflows.

A step can be either an *activity step* or a *control step*. As mentioned above, activity steps are placeholders for reusable activities and each activity step belongs to exactly one activity. Activity steps can be called in a workflow definition. An activity step can be used as a reply for a previous activity step. A single activity step can have several alternative replies.

A control step represents a control flow element such as a split or a join. Conditional and parallel structures are allowed, i.e. the type of a control step is one of the followings: *par-split*, *par-join*, *cond-split* or *cond-join*. An attribute *predicate* is specified only for steps corresponding to a conditional split and represents a conditional predicate. Conditional splits have XOR-semantics. A split control step have a corresponding join control step which is represented by the recursive relation *is_counterpart*. This relation is used to represent well structured workflows [20] where each split node has a corresponding join node of the same type and vice versa.

8 Conclusions

We introduced a layered, distributed architecture for web service composition, composed of choreographies and orchestrations. The concept of views and distributed nature of this model allow business partners to interact and at the same time protect their business know-how and improve their privacy. Besides, business processes can be described in different levels of detail and with a uniform modeling language, as an executable process in orchestrations and as an abstract process in choreographies and views.

References

[1] Workflow process definition interface- xml process definition language. Technical Report WFMC-TC-1025, Workflow Management Coalition, 2002.

[2] A. Barros, M. Dumas, and P. Oaks. A critical overview of the web services choreography description language(ws-cdl). Technical report, Business Process Trends, 2005.

[3] J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Comput. Sci.*, 37:77–121, 1985.

[4] R. Bobrik and T. Bauer. Towards configurable process visualizations with proviado. In *Proc. of WETICE'07*, 2007.

[5] R. Bobrik, M. Reichert, and T. Bauer. View-based process visualization. In *Proc. of BPM'07*, 2007.

[6] D. Burdett and N. Kavantzas. Ws choreography model overview. Technical report, W3C, 2004.

[7] F. Casati and A. Discenza. Supporting workflow cooperation within and across organizations. In *Proc. of the 2000 ACM symposium on Applied computing - Volume 1*, 2000.

[8] I. Chebbi, S. Dustdar, and S. Tata. The view-based approach to dynamic inter-organizational workflow cooperation. *Data and Knowledge Engineering*, 56(2):139–173, 2006.

[9] D.K.W. Chiu, S.C. Cheung, K. Karlapalem, Q. Li, and S. Till. Workflow view driven cross-organizational interoperability in a web-service environment. In *Proc. of WES'02*, 2002.

[10] G. Decker, H. Overdick, and J.M. Zaha. On the suitability of ws-cdl for choreography modeling. In *Proc. of EMISA'06*, 2006.

[11] W.M.P. Van der Aalst. Interorganizational workflows: An approach based on message sequence charts and petri nets. *Systems Analysis - Modelling - Simulation*, 34(3):335–367, 1999.

[12] R.M. Dijkman and M. Dumas. Service-oriented design: A multi-viewpoint approach. *Int. J. Cooperative Inf. Syst.*, 13(4):337–368, 2004.

[13] J. Eder, M. Lehmann, and A. Tahamtan. Choreographies as federations of choreographies and orchestrations. In *Proc. of CoSS'06*, 2006.

[14] J. Eder, M. Lehmann, and A. Tahamtan. Conformance test of federated choreographies. In *Proc. of I-ESA'07*, 2007.

[15] J. Eder, H. Pichler, and A. Tahamtan. Probabilistic time management of choreographies. In *Proc. of QSWS-08*, 2008.

[16] J. Eder and A. Tahamtan. Temporal conformance of federated choreographies. In *Proc. of DEXA'08*, 2008.

[17] R. Eshuis and P. Grefen. Constructing customized process views. *Data & Knowledge Engineering*, 64(2):419–438, 2008.

[18] H. Groiss and J. Eder. Workflow systems for inter-organizational business processes. *SIGGroup Bulletin*, 18(3):23–26, 1997.

[19] J.Y. Jung, W. Hur, S.H. Kang, and H. Kim. Business process choreography for b2b collaboration. *Internet Computing*, 8(1):37–45, 2004.

[20] B. Kiepuszewski, A.H.M. ter Hofstede, and C. Bussler. On structured workflow modelling. In *Proc. of CAiSE*, 2000.

[21] D.R. Liu and M. Shen. Workflow modeling for virtual processes: An order-preserving process-view approach. *Information Systems*, 28(6):505–532, 2003.

[22] D.R. Liu and M. Shen. Discovering role-relevant process-views for disseminating process knowledge. *Expert Systems with Applications*, 26(3):301–310, 2004.

[23] C. Peltz. Web services orchestration and choreography. *IEEE Computer*, 36(10):4653, 2003.

[24] S. Rinderle, A. Wombacher, and M. Reichert. On the controlled evolution of process choreographies. In *Proc. of ICDE'06*, 2006.

[25] W. Sadiq, S. Shazia, and K. Schulz. Model driven distribution of collaborative business processes. In *Proc. of SCC'06*, 2006.

[26] K. Schulz and ME. Orlowska. Architectural issues for cross-organisational b2b interactions. In *Proc. of DDMA'01*, 2001.

[27] K.A. Schulz and M.E. Orlowska. Facilitating cross-organisational workflows with a workflow view approach. *Data and Knowledge Engineering*, 51(1):109–147, 2004.

[28] Z. Shan, Z. Long, Y. Luo, and Z. Peng. Object-oriented realization of workflow views for web services an object deputy model based approach. In *Proc. of WAIM'04*, 2004.

[29] A. Tahamtan. *Modeling and Verification of Web Service Composition Based Interorganizational Workflows*. PhD thesis, University of Vienna, 2009.

[30] A. Tahamtan and J. Eder. Privacy preservation through process views. In *Proc. of AMCA'10 in conjunction with the 24th AINA*, 2010.

[31] A. Tahamtan and J. Eder. View driven federation of choreographies. In *Proc. of ACiiDS'10*, 2010.

[32] Q. Xiaoqiang and J. Wei. A decentralized services choreography approach for business collaboration. In *Proc. of SCC'06*, 2006.