HAL
archives-ouvertes.fr

Towards Decentralized Monitoring of Supply Chains

Aymen Baouab, Walid Fdhila, Olivier Perrin, Claude Godart

▶ **To cite this version:**

Aymen Baouab, Walid Fdhila, Olivier Perrin, Claude Godart. Towards Decentralized Monitoring of Supply Chains. The 2012 IEEE Nineteenth International Conference on Web Services (ICWS 2012), Jun 2012, Honolulu, United States. IEEE Service Computing, 2012. <hal-00718203>

HAL Id: hal-00718203
https://hal.inria.fr/hal-00718203

Submitted on 16 Jul 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Towards Decentralized Monitoring of Supply Chains

Aymen Baouab*, Walid Fdhila†, Olivier Perrin* and Claude Godart*
*Université de Lorraine, LORIA - INRIA, - UMR 7503, BP 239, F-54506 Vandoeuvre-les-Nancy Cedex, France
Email: {aymen.baouab, olivier.perrin, claude.godart}@loria.fr
†University of Vienna, Faculty of Computer Science, 1010 Vienna, Austria
Email: walid.fdhila@univie.ac.at

***Abstract*—Cross-organizational service-based processes are increasingly adopted by different companies when they can not achieve goals on their own. In order to guarantee that all involved partners are informed about errors that may happen in the collaboration, it is necessary to monitor the execution process by continuously observing and checking message exchanges during runtime. This allows a global process tracking and evaluation of process metrics. In this paper, we present an approach for decentralized monitoring of cross-organizational choreographies. We introduce the concept of *External Flow Monitoring* and define a hierarchical propagation model for exchanging external notifications between the collaborating parties. We also introduce the concept of *EFM-view* which allows partners to track the state of the choreography beyond their own processes. The collected monitoring data can be further used for the evaluation of global process metrics by expressing and evaluating statistical queries over execution traces.**

***Keywords*-web service, choreography monitoring, cross-organizational business process.**

I. Introduction

The ability of linking cross-organizational business processes is receiving increased attention in an ever more networked economy [17]. Indeed, collaborative computing grows in importance and processes have to deal with complicated transactions that may take days or weeks to complete across wide ranging geographies, time zones, and enterprise boundaries. Organizations interact with each other forming complex chains including multiple administrative domains in order to drive increased business value.

Building complex distributed processes, without introducing unintended consequences, represents a real challenge. Choreography description languages help to reduce such complexity by providing means of describing complex systems at a higher level. The birth of a service choreography is often determined by putting together external norms, regulations, policies, best practices, and business goals of each participating organization. All these different requirements have the effect of constraining the possible allowed interactions between a list of partners. However, this does not necessarily guarantee that erroneous situations cannot occur due to inappropriately specified interactions. Thus, runtime verification must be taken into consideration, to the aim of checking if the actual behavior of the interacting entities effectively adheres to the modeled business constraints.

Run-time compliance monitoring of services composition have been a subject of interest of several research efforts [20], [2], [1], [3], [4], [5]. Traditional monitoring approaches consist of a central monitor which is notified by each participant whenever an event happens (e.g. sending or receiving a message). Nevertheless, this is mostly limited to inner-organizational roles and may not be adaptive to some situations specially in case of business value chains that span across many circle of trust (i.e. there is no common trusted party among all participants). By removing a central authority responsible for that, it is required to distribute this aspect among the involved participants. Thus, there is a need to find an efficient way to enable an instant propagation of internal exceptions across organizational boundaries. The main purpose behind that is to enable global process tracking, and to guarantee that all involved partners are informed about errors that happen as soon as possible. Indeed, the sooner violations are detected, the better caused failures are properly handled.

In this paper, we present an approach for exchanging monitoring data between collaborating partners. Our approach relies only on state change of choreographies (i.e. occurrence of a message exchange) without providing information about the internal processes. In order to limit data exposure and avoid bandwidth overhead, we define a hierarchical model for selective notification propagation. We design a new component called *External Flow Monitor* (EFM) to be implemented within each participating organization. According to the pre-defined choreography specification, each EFM supervises all incoming and outgoing calls of its organization and automatically exchanges notifications with a pre-calculated subset of other participants. The collected data can be further used for the evaluation of global process metrics and might help business-level stakeholders gaining insight and understanding into their business operations.

The rest of this paper is structured as follows. *Section 2* illustrates our work by a motivating example. *Section 3* introduces the formal definitions of choreography and related concepts. *Section 4* details our decentralized monitoring approach while *Section 5* illustrates it by a case study. *Section 6* outlines some implementation guidelines, *Section 7* discusses related work and *Section 8* summarizes the

contribution and outlines future directions.

II. Scenario and Motivation

To illustrate the motivation and concepts of our work, we adopt the scenario of a supply chain choreography. In this type of cross-organizational business value chains, a reseller interacts with multiple suppliers who in turn interact with multiple manufacturers in order to get a quote for some goods or services.

The basic structure of such a network is presented in *figure* 1.

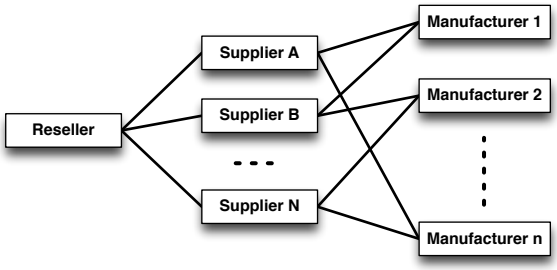

Figure 1. Supply chain structure.

For instance, *figure* 2 shows a choreography involving seven main parties: the customer (C), the reseller (R), two suppliers (SA and SB), and three manufacturers ($A1$, $A2$, and $A3$). First, the reseller receives a request for a quote from the customer (M_1). Then, it selects one of the two suppliers and formulates a new request. In case of selecting $SupplierA$ (M_2), manufacturers 1 and 2 are requested in sequence ($M_{3,\ldots,6}$). In the other case (M_8), manufacturers 2 and 3 are requested in parallel ($M_{9,\ldots,12}$). After getting the final results (M_7 or M_{13}), the reseller replies the customer (M_{14}).

Global process tracking : In the inter-organizational setting, each participant has concern only with the partners immediately connected to it and cannot see beyond. Technically, a service (or a subprocess exposed as a service) can see its providers and its consumers with which it is making service level agreements. During runtime, it has no information about the rest of the service choreography. However, a service is dependent on its lower services and might be interested in monitoring the outsourced process fragments. In our example, the customer C only interacts with the reseller R and cannot see beyond. After sending its request to R, C still waits for the response without having any information about the current state of its request. Thus, it would be interesting to keep C notified of what is happening during its request. Such a case shows the need for exchanging notifications in order to enable choreography tracking. Furthermore, exchanging notifications permits to have a wider view on the choreography and to collect additional monitoring data that might be used a posteriori for measuring global process performance and controlling the achievement of business goals.

Exceptions : Exceptions can occur at any time throughout the execution of this choreography. The reason can be an internal problem of one participant or an incorrect message that differs from the one that is specified in the process description. In the latter case, the problem can be managed rapidly especially when using a reliable messaging protocol between the two partners. However, in the case of an internal error which cannot be reported back to the choreography to be treated as a global exception (e.g. using WS-CDL[16] exceptions), the involved participants may not be notified. Traditionally, this kind of situation is handled by adding confirmation calls after the end of each subprocess and/or fixing global timeouts. However, additional calls may complicate the choreography control-flow and global timeouts are still not well adapted to the long-running collaborations. To address this issue, it is preferable to guarantee that all involved partners are informed about errors that happen as soon as an internal subprocess suffer an exception.

Timeouts : Timeout represents the maximum length of time for which a piece of software logic will continue to wait for a certain event to occur. If that event does not happen in the pre-defined amount of time, it will stop the success execution flow and raise this non-occurrence as an exception of some sort [6]. For instance, when using the BPEL language, a timeout on a receive activity is used when the process is waiting for an external message to arrive. In this case, the timeout value is fixed in accordance with the time in which the service can be expected to complete a request under normal conditions. Typically, this kind of information is prepared for an SLA (service level agreement) which takes into consideration the service levels supported by services it invokes.

In our example, each party may fix a timeout for each invocation to indicate how long an outbound SOAP request will wait for a result message to come back before the participant signals an error.

In case of an internal error within the reseller or a non reception of a response from another participant (e.g. supplier A or B) within the fixed local timeout, the customer remains not informed about that exception and will continue waiting for an answer until its timeout occurs. This may not be adaptive for such a long running collaboration.

III. Choreography (Global / Local View)

A choreography defines re-usable common rules that govern the ordering of exchanged messages, and the provisioning patterns of collaborative behavior, as agreed upon between two or more interacting participants [18]. In this paper, we perceive a choreography as a description of admissible sequences of send and receive messages between collaborating parties. For the sake of simplicity, we omit assignment of global variables. We use ”participant” and

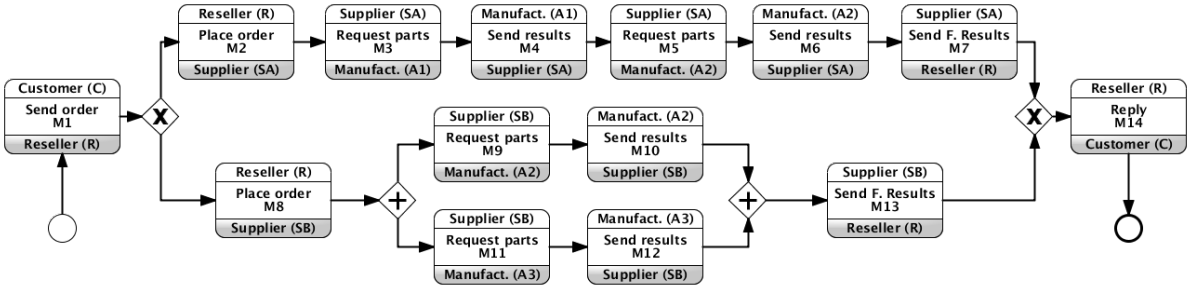

Figure 2. A Supply Chain Example (BPMN 2.0 Choreography Diagram).

"partner" interchangeably. We formalize the semantics of a choreography as follows.

Definition 1 (Choreography global view): Formally, a choreography $\mathcal{C}$ is a tuple ($\mathcal{P}$, $\mathcal{I}$, $\mathcal{L}$) where

- $\mathcal{P}$ is a finite set of participants (Partners),
- $\mathcal{I}$ is a finite set of interactions,
- $\mathcal{L}$ is the set of constraints on the sequencing of interactions.

Every interaction $i \in \mathcal{I}$ corresponds to a certain type of messages, and is associated with a direction of communication. Let $\mathcal{M}_{\mathcal{T}}$ be the set of message types. Formally, an interaction is defined as follows.

Definition 2 (Interaction): An interaction $I \in \mathcal{I}$ is a tuple (s,d,m_t) where s,$d \in \mathcal{P}$ (the source and the destination of the message), and $m_t \in \mathcal{M}_{\mathcal{T}}$ (the type of the message).

In a given choreography, two interactions are either in strict order, in interleaving order, or exclusive to each other. Then, the constraints specified in $\mathcal{L}$ are generally of three types :

- The strict order constraint, denoted by the function $Seq(M_1, M_2)$, holds for two messages M_1 and M_2, if M_2 never occur before M_1.
- The exclusiveness constraint, denoted by the function $Ex(M_1, M_2)$, holds for two messages M_1 and M_2, if they never occur together with the same instance identifier.
- The co-occurrence constraint, denoted by the function $And(M_1, M_2)$, holds for two messages M_1 and M_2, if they occur always together in any execution trace.

The global view of a choreography specifies a high-level view of the conversation between the participants, and can be considered as being interpreted from an observer point of view. The local view is derived from the global view, and specifies the communication-relevant behavior of each participant. Each local view only include interactions where at least one of the communication actions (e.g. receive, send) is executed by the participant for which the local view is generated. It includes also the set of constraints on the sequencing of those interactions. We define a local view as follows.

Definition 3 (Choreography local view): A local view $\mathcal{C}_i$ of a participant P_i is a tuple ($\mathcal{I}_i$, $\mathcal{L}_i$) where

- $\mathcal{I}_i \subseteq \mathcal{I}$ is a set of interactions involving P_i (i.e. having P_i as a source or a destination):
  $i = (s, d, m_t) \in \mathcal{I}_i \Leftrightarrow s = P_i$ or $d = P_i$,
- $\mathcal{L}_i \subseteq \mathcal{L}$ is the set of constraints on $\mathcal{I}_i$.

IV. External Flow Monitoring

A traditional monitor is supposed to monitor the conformance of incoming and outgoing messages choreography of its own organization referring only to its local view. This kind of monitoring does not allow for choreography tracking because it is not possible to see what is happening beyond the borders (e.g. communications between other collaborating parties). To monitor the whole choreography, proposed solutions [20], [4], [14] consist of a central monitor which is notified by each participant whenever an event happens (e.g. using a $publish/subscribe$ mechanism [14]). Nevertheless, this may present a single point of failure and not be adaptive when there is no common trusted party among all participants. In order to enable cross-organizational business process tracking in a decentralized manner, we propose a new automated mechanism for exchanging monitoring data between selected participants, without inducing significant network overhead.

A. External Flow Monitor

We introduce a new component called *External Flow Monitor* (EFM) which has to be deployed to each party's infrastructure. Each EFM is configured considering which events it has to provide and to which EFMs. Based on each choreography specification, notifications are automatically exchanged across the formed interconnected network of EFMs. A description of the EFM is shown in *Figure* 3.

During runtime, the choreography model is used as a basis to transparently observe the behavior of peer-to-peer service interaction. Our approach relies only on choreography state changes (i.e. when a global message is sent or received) monitoring only the interactions between the peers in an unobtrusive way (i.e. the exchanged messages are not altered

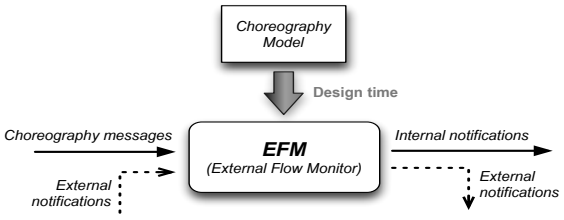

Figure 3. External Flow Monitor.

by the monitors and the peers are not aware of the monitors). We provide an automated mechanism of generating and exchanging notifications. Notifications are defined based on the choreography model by specifying which data it should contain, and to which partner it should be sent.

B. Notifications Exchange

For each exchanged message between two participants, their EFMs generate each a new notification that has to be sent to specified partners. Each notification is correlated to a choreography message and is generated in order to inform other partners about the occurrence of that message. A notification is defined as follows.

Definition 4 (Notification Event): A notification $n \in \mathcal{N}$ is a tuple (ci,t,s,d,m_t) where ci is an instance identifier (used for correlation), t is the timestamp of generation, s,$d \in \mathcal{P}$ (the source and the destination of the associated message), and $m_t \in \mathcal{M}_\mathcal{T}$ (the set of message types).

Exchanging notifications between partners induce additional traffic. In order to enable selective notification exchange and reduce the the additional network overhead (i.e. reduce the number of partners that have to be notified), we introduce a new type of relationship between the participants of the choreography. With such a relationship, we define the direction of each notification.

C. Hierarchical Classification of Participants

Notifications pertaining to different choreography messages of a given participant are meant to be restricted and may be only partially interesting to a subset of business partners. These interested partners are those viewed as consumers of that participant. In a choreography, we can classify participants hierarchically in a producer-consumer manner to form service supply-chains of added value. In other words, the organization of partners is performed in multiple hierarchical layers. Putting the client (i.e. the initiator of the choreography) on the top layer, the lowest layers consist of a set of service providers. Then, each layer is dependent on its lower layers and the propagation of notifications is allowed only upward in the hierarchy. These layers denote the visibility levels of the participants (i.e. service providers and the client). To enable such a classification, we introduce the notion of *super/sub-partner* as follows.

Definition 5 (Super / Sub Partner): A participant $P_i \in \mathcal{P}$ is called the $super-partner$ of a participant P_j iff the first interaction in the local view of P_j has P_i as a sender. In other words, P_i is responsible for the participation of P_j in the choreography. Then, P_j is called a $sub-partner$ of P_i. Formally, we note : $Super(P_j) = P_i$. Then, $P_i \in Sub(P_j)$.

It should be noticed that we assume all choreographies to be realizable [7] with only one initiator. In a supply-chain, a service provider may have sub-contractors which may have also further sub-contractors making a hierarchical structure [15]. In this way, each participant has exactly one *super-partner* (except the one who initiates the choreography) and may have many *sub-partners*. As such, participants can be classified in a tree : the initiator represents the root (see *Figure* 4.a).

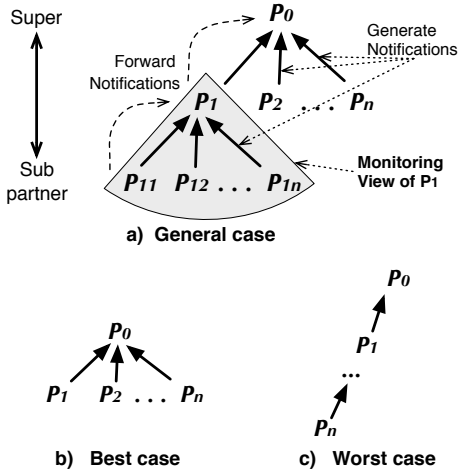

Figure 4. Super/Sub-Partner Tree

Each participant notifies its *super-partner* by generating new notifications (when a message is exchanged) and forwarding those coming from its *sub-partners*. Using a hierarchical architecture permits to lower the complexity in terms of notifications overhead. Indeed, the complexity depends on the number of choreography messages and the number of participants. Let d be the depth of the generated tree. For each message, there are at most $d-1$ notifications that have to be transmitted (one generation and $d-2$ forwards). Then, the number of notifications is upper bounded by $O(m)$ (m is the number of choreography messages).

When considering the number p of participants, the complexity in the best case (when the depth is equal to 1) is obviously linear (*figure* 4.b). In the average case, our approach requires $O(p.log(p))$ notifications, but needs $O(p^2)$ notifications in the worst case, when the unbalanced tree resembles a linked list (degenerate tree) as shown in *figure* 4.c.

Proof: Let c be the average number of leaves of each node (i.e. the number of sub-partners of each participant), m be the maximum number of messages sent by each

participant, and I_i ($i \in [0..d]$) be the level number i of the tree. For each node (participant) in the first level I_1, there is only m sent notifications, for I_2, there are $2*m$, and for I_d, there are $d*m$. Then, after multiplying by c^i (which is the average number of nodes of the level i), the total number of notifications n is equal to $i*c^i*m$ (with $0 < i < d$) which is upper bounded by $O(d*c^d)$. We know that depth d of a c-tree is $O(log_c(p))$. We deduce that n is upper bounded by $O(log_c(p)*c^{log_c(p)}) = O(p*log_c(p))$ if $c > 1$. Indeed, $c = 1$ describes the worst case depicted in *figure* 4.c. ■

D. EFM-View

A local view in a choreography represents the visibility of a business partner. Every partner is limited only to its own view. Using such hierarchical setting, each EFM receives notifications on any state change of all its subsequent sub-partners. Then the *EFM-view* allows for a wider visibility than the local view. The *EFM-view* of a participant P_i includes interactions where at least one of the communication actions (e.g. receive, send) is performed by P_i or all its subsequent sub-partners. It includes also the set of constraints on the sequencing of those interactions. Formally, we define the *EFM-view* as follows.

Definition 6 (EFM View): An *EFM-view* $\mathcal{V}_i$ of a participant P_i is a tuple ($\mathcal{IS}_i$, $\mathcal{LS}_i$) where

- $\mathcal{IS}_i = \cup_{j \in sub(i)} \mathcal{IS}_j \cup \mathcal{I}_i$
- $\mathcal{LS}_i \subseteq \mathcal{L}$ is the set of constraints on $\mathcal{IS}_i$.

Note. If the participant P_i do not have *sub-partners* ($sub(P_i) = \emptyset$) then we have $\mathcal{IS}_i = \mathcal{I}_i$, which means that its *EFM-view* is equal to its local view ($\mathcal{V}_i = \mathcal{C}_i$).

Considering the lifecycle of our monitoring solution we can distinguish between two phases : the setup phase, and the concrete monitoring.

E. Setup Phase

In the setup phase, we mainly identify the *super* of each participant as well as the set of notifications it should send or receive. *Algorithm* 1 presents a pseudocode of the setup procedure.

Algorithm 1: Setup Algorithm

```
1  Require: - C: a choreography (a global view)
2  - P: a set of participants
3  for (each P_i in P) do
4      /* Identifying the Super of P_i */
5      I_i0 ← GetFirstInteraction(P_i)
6      Super_i ← GetSource(I_i0)
7      SubPartners(Super_i) ← SubPartners(Super_i) ∪
       {P_i}
8      /* Define the set of needed
       notifications */
9      V_i ← ConstuctNotificationSet(P_i)
10 end
11 End
```

Let's C be a choreography and $\mathcal{P}$ the set of participants. For each participant P_i, we look for the first partner who initiated a conversation with it. The latter is considered as responsible for initiating P_i in the collaboration (super of P_i). Then, we define the set of needed notifications (i.e. that have to be generated during runtime phase).

F. Runtime Phase

An EFM can be seen as a special case of finite state automaton whose transitions are labeled with message events. The automaton is automatically generated from its defined *EFM- view*. Starting from the initial state when the choreography is locally instantiated, the current state is updated whenever a new exchanged message is detected or an external notification is received. As such, each EFM follows each choreography to which its organization is participating without modifying anything about it (i.e. the structure of the choreography remains the same).

Algorithm 2: EFM : Runtime monitoring and notification management

```
1  Require: - S: The super of the current participant
2  - V: The EFM-view
3  for (each event occurrence e_i ) do
4      if (e_i.type ≠ Exception and CheckConformance(e_i))
       then
5          UpdateMonitorState(e_i)
6          if (e_i.msrc ≠ Super and e_i.mdst ≠ Super) then
           /* Generate a new notification if e_i
           is not coming from/going to the
           Super */
7              if (e_i.type = message exchange) then
8                  n ← GenerateNotification(e_i)
9                  SendNotification(n, Super)
10             else
11                 /* e_i.type = notification
                      exchange                    */
12                 ForwardNotification(e_i, Super)
13             end
14         end
15     else
16         AlertInternalMonitor()
17         ReportExceptionTo(sub-partners(P_i))
18     end
19 end
20 End
```

In *algorithm* 2, we explain the runtime monitoring process from a single partner view, and show how notifications and messages are managed by each EFM. We mention that an EFM deals with three types of events: (i) events related to messages exchanged with other partners (i.e. choreography messages), (ii) events related to exchanged notifications (i.e. for monitoring purpose), and (iii) events stating exceptions. According to the event type, the EFM behaves differently.

- If the event is a message exchange, then the EFM checks if it is conform to its *view*. The conformance test

checks if the message is received or sent with respect to the constraints on the sequencing of interactions (i.e. order, exclusiveness and co-occurence constraints). If the message is conform, then the monitor updates its status in the view graph and eventually instantiates a new notification and sends it to its super. Indeed, we don't need to notify the *super* if he is the source or the destination of the message causing the notification.
- If the event is a notification reception, then the EFM checks if it is conform to its *view*, updates its view status and forwards the notification to its super.
- If the event is an exception generated by another partner, then the EFM would simply forward it to its subsequent sub-partners and treat the exception.

In the two first scenarios, if the event is not conform to the EFM view, an exception is raised, the internal monitor is alerted and all its sub-partners are notified. Each sub-partner propagates the exception to its sub-partners according to the *Super/Sub-partner* tree. It should be noted that the super and its subsequent supers would detect the error automatically and do not need to be notified since they will be blocked in their view graphs.

V. Case Study: Supply Chain Choreography

As an evaluation, we applied our approach on the supply chain choreography introduced in *Section* II. The local views of the reseller and the two suppliers are shown in *figure* 5. For the sake of space, we omit the local views of the other participants.

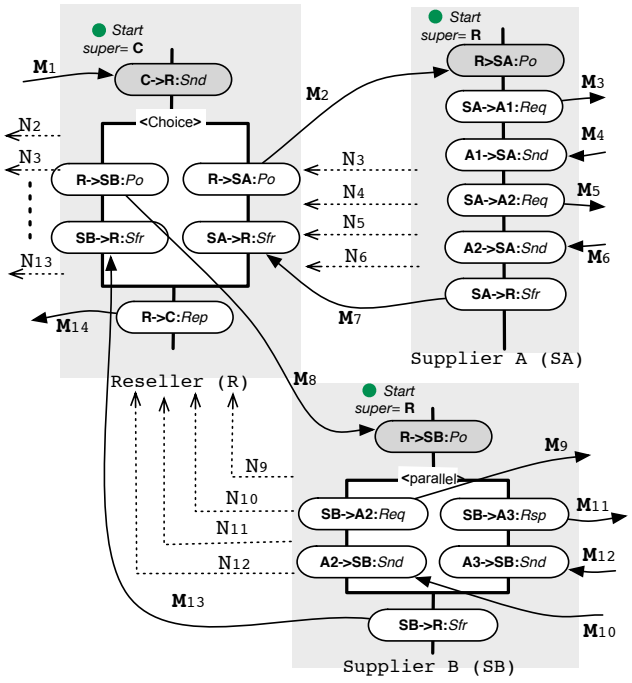

Figure 5. Local Views (Reseller, Supplier A and B).

After the execution of *Algorithm* 1, the first interaction of each local view defines the $Super$ of each party. Then we have $Super(R) = C$, $Super(SA) = R$, and $Super(SB) = R$. The customer C, which is the initiator of the choreography, has no $Super$. Afterwards, the set of needed notifications to be sent to each $Super$ is automatically defined in each party. In this case, we discern twelve definitions of notifications $N_{2,\ldots,13}$ associated, respectively, to the defined choreography messages $M_{2,\ldots,13}$.

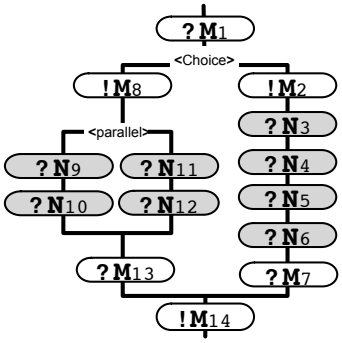

Figure 6. Reseller's Monitoring View.

Figure 6 shows the generated *EFM-view* of the reseller (R). For sake of space, each interaction is labeled with either a send action (prefixed with "!") or a receive action (prefixed with "?"). We remark that the manufacturers $A1$, $A2$, and $A3$ do not have *sub-partners* ($sub(Ai) = \emptyset$) then their *EFM-views* are equal to their local views ($\mathcal{V}_{Ai} = \mathcal{C}_{Ai}$).

Local constraints on each participant behavior can be re-enforced by its super. For instance, the co-occurrence constraint for the two messages M_9 and M_{11} that can be enforced by applying $And(M_9, M_{11})$ on the Supplier B (SB) can be further verified by the EFM of R using $And(N_9, N_{11})$.

Figure 7 sketches an example of execution and the set of sent notifications between the partners. $ci1$ is the instance identifier. τ_i represents a timestamp associated with the occurrence time of the message M_i (either its reception or sending time). A timestamp is a mandatory attribute to verify the ordering of the messages.

VI. Implementation Guidelines

In order to present the concept in a technology-independent way, we prefer to outline a general architecture rather than describing technical details related to any service choreography language. We propose to extend any organization architecture with two components : the *External Flow Controller* and the *External Flow Monitor* (*Figure* 8).

The *External Flow Controller (EFC):* is responsible for the control and the conformance verification of the exchanged choreography messages according to the local view. Two types of control are depicted: (i) incoming message

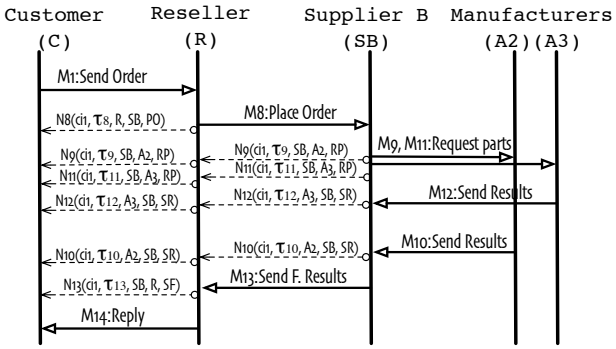

Figure 7. Example of a Successful Scenario.

control in order to restrict the access to internal resources taking into account the context of the call and the control-flow of the involved inter-organizational process, and (ii) outgoing message control used to prevent from divulging sensitive information in unexpected ways (i.e. without a consistent inter-organizational process context). This aspect is not covered in this paper. More details can be found in our previous work [3]. Furthermore, the EFC generates a notification whenever a received or sent choreography message is validated. The generated notification is then locally sent to the EFM.

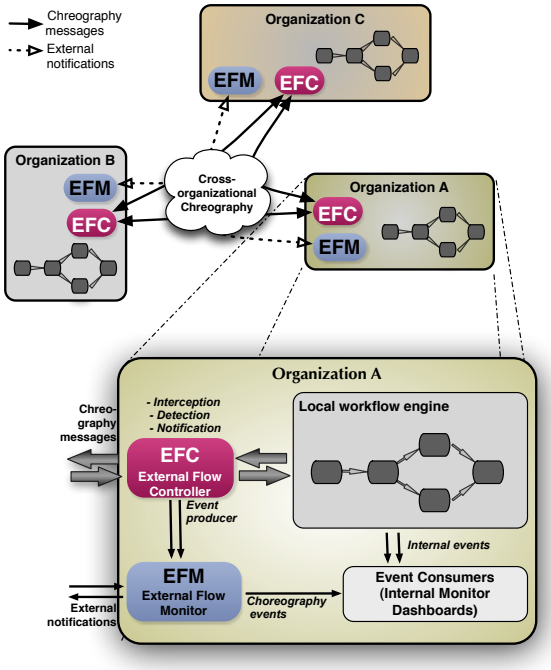

Figure 8. Conceptuel Reference Architecture.

Event correlation –: During the execution phase, we may have several instances of different defined choreography models. Obviously, we need to correlate notification events belonging to the same instance. The most common solution to deal with this issue is to define two identifiers that have to be contained on each message (e.g. included in the SOAP header): The choreography ID (a unique identifier for each choreography model) and the choreography instance ID (a unique identifier for each choreography instance). Since each choreography message passes through the EFC, this component may be adapted to include and read the identifier whenever a choreography message is exchanged.

Event timestamps –: Time is a critical issue in monitoring cross-organizational processes where a set of monitors communicate through message exchange and do not have access to a global clock. Nonetheless generated notification events have to be timestamped and a common time is required to analyze the acquired monitoring data. To deal with this issue, *Clock Synchronization Algorithms* [19] may be adopted. Without setting a global clock, an alternative solution is to send the time separating two consecutive events instead of sending time of occurrence of each event.

VII. Related Work

Run-time monitoring of services composition have been a subject of interest of several research efforts. In this section we want to outline some of the most relevant contributions with the aim to provide a distinction to our work.

Subramanian *et al.* [10] presented an approach for enhancing BPEL engines by proposing a new dedicated engine called "SelfHealBPEL" that implements additional facilities for runtime detection and handling of failures. This approach may have a negative impact on performance and agility since there is no separation between monitoring logic and the BPEL engine. Barbon *et al.* [11] proposed an architecture that separates the business logic of a web service from the monitoring functionality and defined a language that allows for specifying statistic and time-related properties. However, their approach focus on single BPEL orchestrations and do not deal with monitoring of choreographies in a cross-organizational setting. Ardissono *et al.* [20] presented a framework for supporting the monitoring of the progress of a choreography in order to ensure the early detection of faults and the notification of the affected participants. The approach consists on a central monitor which is notified by each participant whenever he sends or receives a message. Nevertheless, this approach is centralized and therefore not well adapted to inter-organizational choreographies specially when there is not a common trusted party among all participants.

In case of decentralized processes within the same organization (or within a circle of trust), Chafle *et al.* [4] have modeled a central entity as a status monitor which is implemented as a web service. On each partition, a local monitoring agent captures the local state of the composite service partition and periodically updates the centralized status monitor. The status monitor maintains the status of

all the activities of the global composite service. However, under high loads, maintenance of global state at one place can become a bottleneck. In [5], the authors introduce the concept of monitor-based messenger (MBM), which processes exchanged messages through a runtime monitor. Each local monitor stamps its outgoing messages with the current monitor state to prevent desynchronizations, provide a total ordering of messages, and offer protection against unreliable messaging.

Regarding event-centric perspectives, process monitoring solutions focus on intra-organizational processes and are mostly based on Business Activity Monitoring (BAM) technology [12]. To the best of our knowledge, only two event-centric approaches deal with monitoring cross-organizational choreographies. The first one [13] uses a common audit format which allows processing and correlating events across different BPEL engines. The second approach [14] introduces complex event specification and uses a choreography instance identifier (ciid) to deal with event correlation (which is not supported in [13]).

Our introduced event based framework allows for runtime decentralized monitoring of choreographies that are deployed across organizational boundaries. In contrast to the previously mentioned works, we rather focus on providing an automated and decentralized mechanism for exchanging notifications across the involved partners according to specific rules that can be automatically generated from the choreography specification.

VIII. Conclusions and future work

In this paper, we proposed an approach for monitoring global state change of cross-organizational choreographies. We introduced the notion of *External Flow Monitoring* and defined an hierarchical propagation model for exchanging external notifications between the collaborating parties in order to limit data exposure and avoid bandwidth overhead. The generation of notifications relies on state change of choreographies allowing to transparently observe the external behavior of each participant without providing information about the internal processes.

Future work will aim at extending our proposed architecture by a new component to manage the actions to be undertaken if an exception is detected (e.g. constraint violation, message/notification redundancy, deadlock, etc.). We also plan to address privacy issues using role based monitoring to avoid critical information disclosure.

References

[1] O. Moser, F. Rosenberg, and S. Dustdar, "Event driven monitoring for service composition infrastructures." in *WISE*. Springer, 2011, pp. 38–51.

[2] A. Francalanza, A. Gauci, and G. Pace, "Runtime monitoring of distributed systems (extended abstract)," University of Malta, Tech. Rep., 2010, wICT.

[3] A. Baouab, O. Perrin, and C. Godart, "An event-driven approach for runtime verification of inter-organizational choreographies," in *2011 IEEE International Conference on Services Computing (SCC)*, july 2011, pp. 640 –647.

[4] G. B. Chafle, S. Chandra, V. Mann, and M. G. Nanda, "Decentralized orchestration of composite web services," in *Proceedings of the 13th international World Wide Web conference*, ser. WWW. NY, USA: ACM, 2004, pp. 134–143.

[5] S. Halle and R. Villemaire, "Flexible and reliable messaging using runtime monitoring," in *Enterprise Distributed Object Computing Conference Workshops, 2009. EDOCW 2009. 13th*, sept. 2009, pp. 116 –125.

[6] A. Tost, "Planning and handling timeouts in service-oriented environments," *in IBM WebSphere Developer Technical Journal*, 2009.

[7] N. Lohmann and K. Wolf, *Realizability Is Controllability*, ser. Lecture Notes in Computer Science, C. Laneve and J. Su, Eds. Springer Berlin Heidelberg, 2010, vol. 6194.

[8] M. Montali, F. Maggi, F. Chesani, P. Mello, and W. van der Aalst, "Monitoring business constraints with the event calculus," in *Technical report, Universita degli Studi di Bologna*, 2011.

[9] A. Awad and M. Weske, "Visualization of compliance violation in business process models," in *Proc. BPI09*, 2009.

[10] S. Subramanian, P. Thiran, N. Narendra, G. Mostefaoui, and Z. Maamar, "On the enhancement of bpel engines for self-healing composite web services," in *Applications and the Internet. SAINT 2008.*, 2008, pp. 33 –39.

[11] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti, "Run-time monitoring of instances and classes of web service compositions," *Web Services, IEEE International Conference on*, vol. 0, pp. 63–71, 2006.

[12] A. Dahanayake, R. J. Welke, and G. Cavalheiro, "Improving the understanding of bam technology for real-time decision support," *Int. J. Bus. Inf. Syst.*, vol. 7, pp. 1–26, 2011.

[13] S. Kikuchi, H. Shimamura, and Y. Kanna, "Monitoring method of cross-sites' processes executed by multiple ws-bpel processors," in *CEC/EEE 2007*, 2007, pp. 55 –64.

[14] B. Wetzstein, D. Karastoyanova, O. Kopp, F. Leymann, and D. Zwink, "Cross-organizational process monitoring based on service choreographies," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, ser. SAC '10. New York, NY, USA: ACM, 2010, pp. 2485–2490.

[15] I. ul Haq, A. Huqqani, and E. Schikuta, "Aggregating hierarchical service level agreements in business value networks," in *Business Process Management*. Springer Berlin / Heidelberg, 2009, pp. 176–192.

[16] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. Web services choreography description language version 1.0. *W3C. Available from:*, 2005.

[17] P. Grefen, "Towards dynamic interorganizational business process management," *Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE*, 2006.

[18] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto, "Web services choreography description language version 1.0," *W3C. Available from:*, 2005.

[19] E. Anceaume and I. Puaut, "A Taxonomy Of Clock Synchronization Algorithms," 1997.

[20] L. Ardissono, R. Furnari, A. Goy, G. Petrone, and M. Segnan, "Monitoring choreographed services," *in Innovations and Advanced Techniques in Computer and Information Sciences and Engineering, CISSE 06, pp. 283288*, 2006.