論文 / 著書情報
Article / Book Information

Title	Reliability and Performance Estimation for Enriched WS-SAGAS
Author	Neila Ben Lakhal, Takashi Kobayashi, Haruo Yokota
Journal/Book name	IEEE Proc. of International Workshop on Challenges in Web Information Retrieval and Integration (WIRI2005), Vol. , No. , pp. 55-64
Issue date	2005, 4
DOI	10.1109/WIRI.2005.34
URL	http://www.ieee.org/index.html
Copyright	(c)2005 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other users, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works for resale or redistribution to servers or lists, or reuse of any copyrighted components of this work in other works.
Note	このファイルは著者（最終）版です。 This file is author (final) version.

Reliability and Performance Estimation for Enriched WS-SAGAS

Neila Ben Lakhal[†], Takashi Kobayashi[‡] and Haruo Yokota[†‡]
[†]Tokyo Institute of Technology, Department of Computer Science
[‡]Tokyo Institute of Technology, Global Scientific Information and Computing Center
2-12-1 Oh-Okayama, Meguro-ku Tokyo, 152-8552 Japan
neila@de.cs.titech.ac.jp, {tkobaya, yokota}@cs.titech.ac.jp

Abstract

Over the last couple of years, the web services compositions paradigm has been gathering a considerable momentum to grasp at the opportunity of becoming the most natural solution for autonomous and heterogeneous applications integration.

In this paper, we advocate that we have yet to be concerned about the reliability and performance levels that the web services compositions will exhibit upon execution, which is still an ongoing research problem finding notable interest. Focusing on this issue, we propose to estimate the reliability and performance of web services compositions. We concentrate on one important aspect that has received little attention so far: the consideration of the failures repercussions on the overall composition execution performance.

Specifically, our targets are twofold. Firstly, we propose to enrich WS-SAGAS with a new set of advanced aggregation patterns so that it fits with the inherent business processes complexity and thereby it allows to specify, as web services compositions, any business process no matter how it turns out to. Secondly, we propose to estimate the reliability and performance of each of these patterns, and this, in both, correct and faulty situations.

Our enriched WS-SAGAS model ameliorates considerably the chances to acquire more plausible estimations of the reliability and performance because of its failure awareness. Moreover, analyzing these estimations helps to investigate the reasons that may lay behind these failures and indeed to contribute noticeably to acquire more reliable compositions execution with high performance level.

1. INTRODUCTION

One of the most spectacular characteristics of the few past years is the ubiquitous spread of heterogeneous and autonomous applications cooperating in complex cross-organizational business processes.

Ensuring a high-performance-level of execution of such processes is a call to challenge the integration of disparate systems and surmount their inherent incompatibilities.

So far, the web services compositions paradigm rose to the occasion and its efficiency in facilitating such applications integration has been widely demonstrated. It is rapidly gathering considerable momentum to grasp at the opportunity of becoming the most natural solution for such applications integration. Behind the scenes, its key enablers are, building on a set of widely recognized standards, using ever-present protocols, and finally showing an extraordinary capability to glue any (disparate) systems together.

To date, many solutions were proposed to aggregate individual web services to derive from them new compositions, enabled even to encapsulate complex business processes underpinning logic. However, to enforce that such processes reliably serve their purpose, comparing the semantics of their interleaved services and checking their ports compatibility is no longer enough. We advocate that we have yet to be concerned about achieving a high-level Quality of Service(QoS) in general, and a high-level of performance, specially at the the composition execution level, which is still an ongoing research problem finding notable interest.

The notion of QoS is broad and is applied to many areas. Depending on its area of application, its definition varies. Some defines it as *"a set of user-perceivable attributes, which describes a service and the way it is perceived"* [1][2]. We are not concerned with this QoS since it has been widely addressed and was the subject of considerable research efforts in web services compositions. So far, several studies focused on the dynamic selection of the provider [3][4][5] and on semantic services descriptions to improve the dynamic selection[6]. These studies are classified under the umbrella of maximizing user's satisfaction.

A more appropriate definition of the QoS with which we deal in our present work is *"the system property which consists of a set of quality requirements on the collective behavior of one or more objects, as the information transfer rate, the latency, the system failure probability, etc."*[1][7].

Concerned with enhancing the reliability and availability dimensions of the web services compositions performance in particular, and its QoS in general, we have proposed in a previous work, *WS-SAGAS*, a *transaction model* for web services compositions reliable specification[8][9], and *THROWS (Transaction Hierarchy for Route Organization of Web Services)*, a *distributed architecture* for highly available execution of web services compositions[10][11].

In this paper, we focus on another issue related with the qualitative aspect of web services compositions. In particular, we propose to estimate the reliability and performance of web services compositions and to concentrate on one important issue that has received little attention so far, which is considering the failure repercussions on the composition execution performance estimations.

Specifically, our targets are twofold. First, we propose to enrich WS-SAGAS with a new set of advanced aggregation patterns so that it fits with the inherent business processes complexity and thereby allows to specify, as composition, any process no matter how it turns out to. Second, we propose to characterize, estimate and analyze the performance of each of these patterns,in terms of *Reliability* and *Execution Time*, in correct and faulty situations.

Contrary to other proposed approaches -also characterizing these two dimensions- where only the compositions "eternal" correct execution were considered and where the failure (detection and recovery) was not taken into consideration in none of the estimated metrics, we advocate that such performance estimations are too optimistic and far from the actual situation. This ensues from our belief that the correct execution does not reflect always the real situation of any system as reliable as it its said to be. Usually, failures (detection and recovery) contribute more or less in the effective performances of any system, specially in the web services context and their inherent tendency to fail rather easily (relatively to other computing components).

As for the reliability dimension characterization, we introduce a new concept, the *Reliability Tendency(RT)*, which builds mainly on the concepts of element's *State(S)*, *States Tendency Set(STS)*, and *Terminal States Set*. The *Reliability Tendency* upholds the idea that from one state to another, the reliability contributions vary. The *State Tendency* would allow to approximate the failures locations (e.g., the element' with *State Tendency Set* comprising the *Failed* state).

In estimating the *Execution Time* dimension, we introduce the notion of *Probable Execution Time* and *Optimistic Execution Time*, where the former considers all the possible system states (i.e., correct/faulty/recoverable/none recoverable execution), and where the latter is only limited to the correct execution of WS-SAGAS. Distinguishing between these two variants of the time dimension allows us to provide more accurate estimations.

Finally, since a web services composition is a collaboration of elements, we propose to provide performance estimations of each element. Later, we derive the performance of the overall composition, following the newly defined aggregation patterns in the Enriched WS-SAGAS.

This paper is structured as follow. Section 2 describes our motivations and our contributions. Section 3 discusses some related work. Finally, section 4 concludes our present work and gives and outlook of our future directions.

2. ENRICHED WS-SAGAS

2.1. PRELIMINARIES

To cope with web services compositions reliability and availability issue, we have proposed in a previous work, *WS-SAGAS* and *THROWS* architecture[8]-[11]. In a nutshell, our previous proposal most prominent features are:

- WS-SAGAS transaction model specifies the web services composition as a hierarchy of arbitrary-nested transactions. These transactions execution is provided with state capturing, execution retrial and compensation mechanisms;
- In WS-SAGAS, we defined a web services composition (WSC) as an orchestration of n *elements* (E_i^v) noted $\{E_1^v, E_2^v ... E_n^v\}$. An *element* has a *state* (S_i) and a *vitality degree* (noted E_i^v for *vital* and $E_i^{\bar{v}}$ for *not vital*). According to the considered nesting level, the same element could be either assimilated to an atomic element or to a composition of elements;
- The state of an element (*vital* or *not-vital* independently noted E_i), noted S_i, is exclusively one of the followings: *Waiting, Executing, Failed, Aborted, Committed or Compensated*;
- THROWS architecture applies a peer-to-peer execution pattern where the composition execution control is distributed among dynamically discovered engines. An engine (noted e_i) is attached to an involved web service and is allocated to an element E_i at runtime. Once allocated, the engine is in charge of the web service invocation, its execution context communication/update(.e., input and output variables), its execution control delegation or completion etc. On each engine, the *Current Execution Progress (CEP)* and the *Candidate Engines List (CEL)* are locally stored.
- $CEP(WSC_c)$ keeps track of the web services composition WSC_c execution progress. When an element is executed by an engine, every change in that element's state is reflected on *CEP*;
- $CEL(E_i)$contains all the candidate engines potentially enabled to execute E_i. Every engine, after committing the execution of its allocated element, it generates the CEL of its direct successor(s).

2.2. MOTIVATIONS AND CONTRIBUTIONS

In this paper, we propose in our model to map the performance of web services compositions, depicted as *Enriched WS-SAGAS*, to estimations of two dimensions the *Reliability* and *Execution Time*, the more likely performance aspects to be influenceable by failures. In this work, we limited our model to only two performance-related dimensions. However, our model can be extended in the future with other aspects.

The estimation of reliability and execution time issue, in web services context, is with little previous work. Part of the proposed work is oriented toward quality measurements of web services compositions, but without any failure consideration.

Considering the *Execution Time* aspect, [12][13] defined the execution time taken by a single web service invocation with three constituents:

- `Service Time` is to perform the service task;
- `Message Delay Time` is the time taken by the SOAP messages to be sent/received and
- `Waiting Time` is the web service invocation delay.

Still, considering only these three constituents in defining an element execution time is not enough since this definition considers only the case of mapping one web service-to-one element. Moreover, it is rather optimistic since it doesn't take into account any eventual failure (information and recovery time). Indeed, we assume that an effective execution of any element by any web service in actual situation is inherently error-prone.

Our *Enriched WS-SAGAS model* builds on the following definitions to estimate the reliability and performance:

A- RELIABILITY-RELATED DEFINITIONS

DEFINITION A.1 Let E_i and E_j two elements from a composition WSC_c. If E_i execution can be more *times* attempted than E_j, then E_j is said to *tend to be more reliable* than E_j, since it has more chances to be retried and potentially reach the *Committed* state. We define this notion as the element's *Retriability*. Each element's *retriability* is function of its CEL cardinality (noted $|(CEL(E_i))|$).

DEFINITION A.2 Each element E_i has a *Terminal State(noted TS)* with which each invocation is terminated. After m invocations, for each element, a *Terminal States Set*(noted $TSS(E_i)$) is formed. The cardinality of $TSS(E_i)$ verifies Equation(1).

$$1 \leq |TSS(E_i)| \leq 5 \quad (1)$$

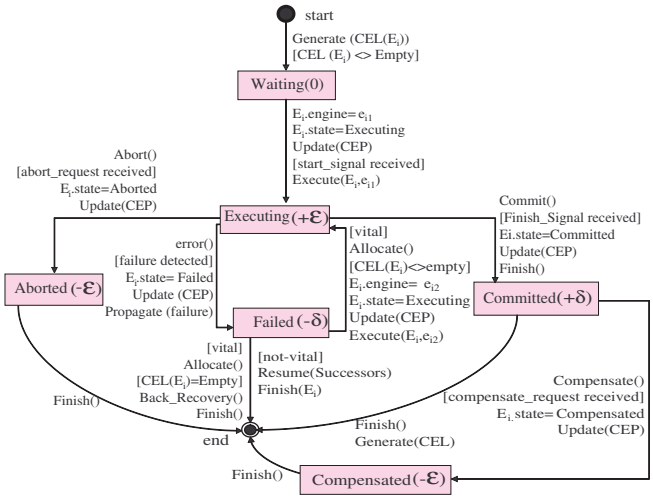

Figure 1. State Transition Diagram with States Reliability Contributions

DEFINITION A.3 After m invocations of E_i, at least one state, among the five different states {*Waiting, Failed, Aborted, Committed,Compensated*} tends to have the biggest occurrence number as a *Terminal State*. We introduce the notion of *State Tendency Set*(noted $STS(E_i)$), as the state(s) that has(have) the biggest occurrence number after m invocations. $STS(E_i)$ contains the set of state(s) that are included within the *TSS*, and which have the biggest number of occurrences after an element is executed m *times*:

$$STS(E_i) = \max_{State \in TSS} \{\sum_{1}^{m} State(occu(State))\} \quad (2)$$

With: $STS(E_i) \subseteq TSS(E_i)$ $occ(State)$: the number of *times* (after *m* invocations) E_i execution was terminated with the state $State$.

DEFINITION A.4 We define the *State Reliability Contribution*(noted SRC) as how the reliability of an element is influenced by its TS. From one state to another, the state's contribution in the reliability differs.

For instance, the state *Failed* would decrease the reliability contrary to the *Committed* state which contributes positively in increasing the reliability. Similarly, the *Waiting* state is considered as neutral since it doesn't have any effect on the reliability. As for the *Executing* state, since it is one step toward the fulfillment of an element, its reliability level is in between the *Committed* and *Waiting* state. Finally, the *Compensated* and *Aborted* states are performed only as a result of failure handling. Thus they have the same negative contribution, in between the contributions of *Waiting* state and the *Failed* state.

Indeed, we assume that a transition from one state to another state makes the *State Reliability Contribution* stronger if is toward reaching the $Committed$ state and it contributes negatively and makes the *State Reliability Contribution* weaker if it is toward the $Failed$ state.

(Figure.1) depicts the state transition diagram that we have defined in WS-SAGAS execution following THROWS architecture. The diagram is enriched with each element's *SRC*, written between () after each state. Initially, each *SRC* can be allocated values based on the designers judgement.

DEFINITION A.5 The notion of *Reliability Tendency* for an element E_i (noted $RT(E_i)$) is derived from the previous definitions as in Equation(3). It defines the reliability rate of each element, after m invocations.

$$RT(E_i) = \frac{\sum_{\text{State}\in\ STS(E_i)} (ST(State) * SC(State))}{|STS(E_i)|} \quad (3)$$

We emphasize here that we introduced this appellation of *Reliability Tendency* (equivalent to *Reliability* in other work) because we are deeply convinced that in the web services context, fully precise reliability measurements are very difficult to acquire, in view of its dynamism. Estimate the rate to which the reliability will tend is more plausible.

DEFINITION A.6 The *Reliability Tendency* of a web services composition WSC_c, comprising n elements, is derived from its elements respective *RT* as follows:

$$RT(WSC_c) = \frac{\sum_{0\leq i\leq n} (RT(E_i))}{n} \quad (4)$$

B- EXECUTION-TIME-RELATED DEFINITIONS

DEFINITION B.1 We define the *Optimistic Execution Time* of an element E_i from a composition WSC_c (noted $T(E_i)_{opt}$) as the execution time of the *dynamically-mapped* web service at runtime to E_i. This definition considers only the best case where E_i is mapped to a service which succeeds in its execution.

We remind here that any E_i can be mapped at runtime to more than one web service, at most the cardinality of $CEL(E_i)$. We assume that E_i was executed by q engines ($q \leq |CEL(E_i)|$) from $CEL(E_i)$. Among these executions, we assume that $(q-1)$ executions were finished with failures. Indeed, E_i was retried $q\ times$ and the q^{th} execution delegated to engine e_{iq} (controlling the web service ws_{iq}) was successful. Building on [12][13], we define $T(E_i)_{opt}$ by Equation(5), with $T(E_i, e_{iq}, ws_{iq})$ is the time to execute ws_{iq} ranked in position q in $CEL(E_i)$.

$$\begin{aligned} T(E_i)_{opt} &= T(E_i, e_{iq}, ws_{iq}) \\ &= S(ws_{iq}) + M(ws_{iq}) + W(ws_{iq}) \\ &\quad \text{with: } 1 \leq q \leq |CEL(E_i)| \end{aligned} \quad (5)$$

DEFINITION B.2 We define the *Probable Execution Time*(noted $T(E_i)_{prob}$), as the estimation of the execution time of E_i. As formulated by Equation(6), it takes into account the allocated web service failure: the time necessary to inform about it and to recover from it (forward and backward recovery are considered).

$$T(E_i)_{prob} = T(E_i)_{opt} + I(E_i) + R(E_i) \quad (6)$$

DEFINITION B.3 We define the *Failure Information Time* (noted $I(E_i)$), as the time taken by the different SOAP messages to be sent/received between web services-engine and peer-engines as notifications of failure.

$$I(E_i) = \sum_{k=1}^{q-1} I(E_i, e_{ik}, ws_{ik}) \quad (7)$$

Since any element E_i might be subject to as many failures as the number of times it was reattempted (at most q times), $I(E_i)$), defined by Equation(7), is the sum of all the elapsed periods of time to notify about each of the allocated web services' failure.

For more details about the different messages sent, depending on the occurred failure type/location refer to [10].

DEFINITION B.4 We define the *Failure Recovery time*(noted $R(E_i)$) as the required time to recover from E_i failure. Since we deal with transactions, any execution must terminate in a consistent state.

$$R(E_i) = For(E_i) + Back(E_i) \quad (8)$$

With:
$$For(E_i) = \sum_{k=1}^{q-1} T(E_i, e_{ik}, ws_{ik}) \quad (9)$$

$$Back(E_i) = xor(Comp(E_i), Abort(E_i)) \quad (10)$$

As described in Equation(8), $R(E_i)$ can be of two kinds:

1. $For(E_i)$: Time necessary to perform a *Forward Recovery*. It is equal to the execution retrial with the different web services that have failed, that is, as we described it in Equation(9), the sum of the time necessary to execute the $(q-1)$ web services that have failed;
2. $Back(E_i)$: We define it as the *Backward Recovery Time*: the time necessary to trigger a backward recovery mechanism by aborting all the still-executing elements and compensating all the already-committed ones, in case a failure cannot be forwardly recoverable. This induces that $Back(E_i)$ is either equal to the web service *Compensation Time* $Comp(E_i)$ or to the web service *Aborting time* $Abort(E_i)$ as in Equation(10).

2.3. ADVANCED AGGREGATION PATTERNS OF WS-SAGAS

When we firstly proposed WS-SAGAS, only basic aggregations of web services compositions were defined (parallel, sequential and recursive). However, considering how business processes tend to involve more complex patterns and to be orchestrated in more different ways, we propose to enrich WS-SAGAS with a set of aggregation patterns compiled in [14].

Here, we put the accent on what follow, first, we propose to add new aggregation patterns to WS-SAGAS to broaden our model potential scope of application since we target to make WS-SAGAS rich enough to sustain any business process no matter how complex it turns out. Second, As we already specified it when we first proposed WS-SAGAS, we are bearing in mind that our model must be an all-encompassing model to support any information integration situations whether on web or not. These two reasons were mainly behind our motivation to introduce new aggregation patterns in WS-SAGAS.

The aggregation patterns proposed in [14] are greatly inspired from an analysis of existing workflow languages. These patterns capture typical control flow dependencies encountered in workflow modeling and they arguably apply as well for web services composition, since the situations they capture are also relevant in this domain.

In what follows, we propose to build on[14] and [15] work to define our set of WS-SAGAS advanced patterns. For each of our defined pattern, we derive also the equation for the execution time. However, we do not specify the reliability aspect since it is not altered with the aggregation pattern that the composition follows.

2.3.1. Sequence of WS-SAGAS : An element E_j in a composition WSC_c is enabled after the successful completion of its direct predecessor, the element E_i, having i, j in $[0..n]$ and $i < j$ (Figure.2). The estimation of the *Probable Execution Time* of WSC_c is described in Equation (11).

$$\mathrm{T(WSC_c)_{prob}} = \mathrm{T(E_i)_{prob}} + \mathrm{T(E_j)_{prob}} \qquad (11)$$

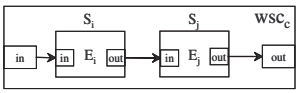

Figure 2. Sequence of WS-SAGAS

2.3.2. Parallel WS-SAGAS : Say we have a composition WSC_c where an element E_i, after its commitment, it has to dispatch the execution control to its direct successors: elements E_j to E_k ($j \ll k$)(Figure.3), which are executed in parallel (noted $[E_j; ...; E_k]$).

Considering the case where these elements are run in *parallel* and *simultaneously*, the *Probable Execution Time* aggregation is defined by Equation(12).

Since we consider the case of a transactional execution, and following THROWS definitions, the order of the different elements should be known beforehand.

We assume also that every element execution have to be terminated necessary in a consistent state either with triggering the following element(s) execution, with terminating the overall composition execution (we are at the end of the WS-SAGAS), or it should trigger a failure recovery in case any failure has happened.

If they are executed in any other order the estimation of the probable execution time of WSC_c would be, in the worst case as described in the aggregation Equation (11).

$$\begin{aligned}\mathrm{T(WSC_c)_{prob}} &= \mathrm{T(E_i)_{prob}} \\ &\quad + \max[\mathrm{T(E_j)_{prob}}; ...; \mathrm{T(E_k)_{prob}}]\end{aligned} \qquad (12)$$

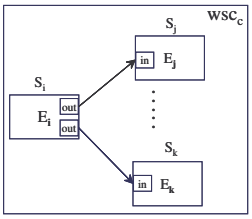

Figure 3. Parallel WS-SAGAS

2.3.3. Selection of WS-SAGAS : This pattern is similar to the previous one. The difference is that element E_i has to operate a choice among the following elements and at *least one* and *at most all* the following branches could be selected. In other words, say we have $\mathcal{Q}(E_i)_{\text{succ}}$, the set of all the *direct successive* elements of E_i(Figure.4). E_i has to choose among $\mathcal{Q}(E_i)_{\text{succ}} = \{E_j, ..., E_k\}$).

Building on the same assumption as previously that, equal probabilities among all choices holds, E_i can choose among $\mathcal{P}(\mathcal{Q}(E_i)_{\text{succ}})$, the power set of $\mathcal{Q}(E_i)_{\text{succ}}$, which is the set of all subsets of $\mathcal{Q}(E_i)_{\text{succ}}$. We define $\mathcal{S}(E_i)_{\text{succ}}$, as a subset chosen from $\mathcal{P}(\mathcal{Q}(E_i)_{\text{succ}})$, with $\mathcal{S}(E_i)_{\text{succ}}$ verifies: $\mathcal{S}(E_i)_{\text{succ}} \subseteq \mathcal{P}(\mathcal{Q}(E_i)_{\text{succ}})$ and $\mathcal{S}(E_i)_{\text{succ}} \neq \emptyset$.

$$\mathrm{T(WSC_c)_{prob}} = \mathrm{T(E_i)_{prob}} + \mathrm{T}(\mathcal{S}(\mathrm{E_i})_{\text{succ}})_{\text{prob}} \qquad (13)$$

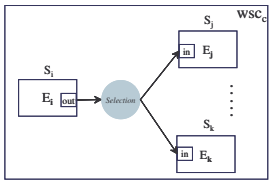

Figure 4. Selection of WS-SAGAS

In the execution time described in Equation (13), by $T(\mathcal{S}(E_i)_{\text{succ}})_{prob}$, we denote the probable execution time of the different elements of this subset. If these elements are to be executed in parallel, apply the equation of pattern *Parallel WS-SAGAS*, otherwise, if it is a sequential execution apply the equation of pattern *Sequence of WS-SAGAS*.

2.3.4. Switch of WS-SAGAS : This pattern is actually a specialization of the pattern **Parallel WS-SAGAS**. The difference is that at least AND at most only one element can be chosen. It is a point in the composition, where based on an obtained result from a previous element E_i execution, a choice need to be operated among several elements $[E_j|...|E_k]$(Figure.5). We assume here that all elements have the same probability to be chosen. The estimation of $T(WSC_c)_{prob}$ depicted as follows is:

$$\mathrm{T(WSC_c)_{prob}} = \mathrm{T(E_i)_{prob}} + [\mathrm{T(E_j)_{prob}xor...xorT(E_k)_{prob}}] \quad (14)$$

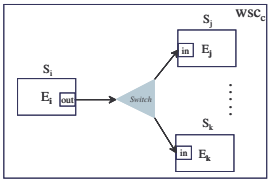

Figure 5. Switch of WS-SAGAS

2.3.5. Rendezvous of WS-SAGAS : This pattern considers the case where there exists a rendezvous point in the composition where multiple parallel elements converge into one single thread of control, thus synchronizing multiple threads(Figure.6). Say we have E_j to E_k elements which are executed in parallel ($[E_j; ...; E_k]$), their direct common successor is an element E_l, which actually cannot start its execution unless all the elements ($[E_j; ...; E_k]$) has terminated successfully. The $T(WSC_c)_{prob}$ expression is:

$$\mathrm{T(WSC_c)_{prob}} = \max[\mathrm{T(E_j)_{prob}}; ...; \mathrm{T(E_k)_{prob}}] + \mathrm{T(E_l)_{prob}} \quad (15)$$

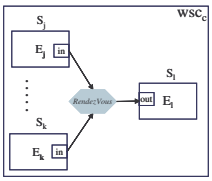

Figure 6. Rendezvous of WS-SAGAS

2.3.6. Selective Merge of WS-SAGAS : It is a point in the web services composition where two or more elements' executions reconverge BUT without synchronization.(Figure.7).

Contrary to the pattern *Exclusive Merge of WS-SAGAS*, where only one element is executed, and its execution success would trigger the direct successor only once, this pattern assumes that more than one element from $\mathcal{Q}(E_l)_{\mathsf{pre}} = \{E_j, ..., E_k\}$ at a time can get activated, possibly concurrently, and the element E_l is started for every activation of every incoming branch.

Let $\mathcal{S}(E_l)_{\mathsf{pre}}$, the subset chosen from $\mathcal{P}(\mathcal{Q}(E_l)_{\mathsf{pre}})$ and let $\lambda = |\mathcal{S}(E_l)_{\mathsf{pre}}|$. At most $\lambda = |\mathcal{Q}(E_l)_{\mathsf{pre}}|$. The upper bound of $T(\mathcal{S}(E_l)_{\mathsf{pre}})_{prob}$ is reached when all the elements from $\mathcal{S}(E_l)_{\mathsf{pre}}$ are executed and in a sequential order.

$$\mathrm{T(WSC_c)_{prob}} = \mathrm{T(\mathcal{S}(E_l)_{pre})_{prob}} + \lambda \mathrm{T(E_l)_{prob}} \quad (16)$$

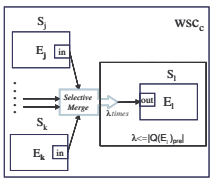

Figure 7. Selective Merge of WS-SAGAS

2.3.7. Exclusive Merge of WS-SAGAS : We consider the case where TWO OR MORE alternative elements come together BUT without synchronization. We assume that none of the elements is ever executed in parallel(Figure.8).

Let $\mathcal{Q}(E_l)_{\mathsf{pre}} = \{E_j, ..., E_k\}$, the set of elements which are the direct common successor of element E_l. Actually, E_l cannot start its execution unless either of the elements of $\mathcal{Q}(E_l)_{\mathsf{pre}}$ has terminated successfully. In other words, E_l execution cannot be triggered unless any of its previous elements has terminated successfully its execution. We must make sure that only one of the branches is triggered.

Assume we have a set $Singleton(E_l)_{\mathsf{pre}}$, which contains one of the possible elements to choose among $\mathcal{Q}(E_l)_{\mathsf{pre}}$. Any of the elements of $\mathcal{Q}(E_l)_{\mathsf{pre}}$ have the same probability to trigger the execution of E_l and thus to be in $Singleton(E_l)_{\mathsf{pre}}$. We assume that in case $Singleton(E_l)_{\mathsf{pre}} = \emptyset$, a failure has for sure occurred and that a recovery should be triggered. The probable execution of such a composition is:

$$\mathrm{T(WSC_c)_{prob}} = \mathrm{T(Singleton(E_l)_{pre})_{prob}} + \mathrm{T(E_l)_{prob}} \quad (17)$$

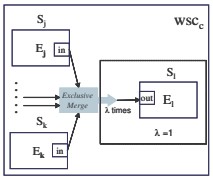

Figure 8. Exclusive Merge of WS-SAGAS

2.3.8. Iterative WS-SAGAS It is a point in the web services composition execution where a particular element E_j execution needs to be repeated λ times (Figure.9). The number of iteration is determined upon the decision of a direct predecessor E_i which has finished executing and needs to delegate the control.

$$\mathrm{T(WSC_c)_{prob}} = \mathrm{T(E_i)_{prob}} + \lambda \mathrm{T(E_j)_{prob}} \quad (18)$$

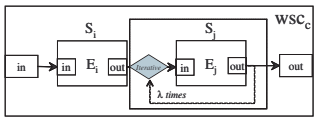

Figure 9. Iterative WS-SAGAS

2.4. CASE STUDY

We consider the case study of an online shopping web site. A customer accesses the online shopping web site. After selecting the list of items he wants to order and placing them in his shopping basket, to place his order, he needs either to login if he has already signed up, otherwise, he needs to register as a new customer. Once he either logged in successfully or registered successfully as a new customer, he needs to choose simultaneously the mode of delivery and payment. Once it is done, he needs to finalize his order.

Using only our WS-SAGAS former notation to describe this case study is not possible because the exclusive choice pattern was not yet defined. Trying to depict such a process shows how valuable was adding the newly introduced aggregation patterns in this paper. (Figure.10) follows the enriched WS-SAGAS notation to depict this scenario.

2.4.1. Situation 1. We consider the scenario where a customer put three items in his basket ($E_{1.1} \mapsto ws_{11}$), logged in ($E_{1.2} \mapsto ws_{21}$), did not change any of his registered information ($E_{1.4} \mapsto ws_{41}$, chose rapid post as delivery mode ($E_{1.5} \mapsto ws_{51}$) and credit card as payment mode ($E_{1.6} \mapsto ws_{61}$), and finally validated his order ($E_{1.7} \mapsto ws_{71}$).

We suppose that all the different web services terminated successfully. We suppose also that $E_{1.5}$ requires longer execution time than $E_{1.6}$ and that we have: $\mathcal{Q}(E_{1.4})_{\text{pre}} = \{E_{1.2}, E_{1.3}\}$ since the customer logged in, so $Singleton(E_{1.4})_{\text{pre}} = \{E_{1.2}\}$. The probable execution time of the composition WSC_1 is described in Equation(19):

$$\begin{aligned}\mathrm{T(WSC_1)_{prob}} &= \mathrm{T(E_{1.1})_{prob}} + \mathrm{T(Singleton(E_{1.4})_{pre})_{prob}} \\ &\quad + \mathrm{T(E_{1.4})_{prob}} \\ &\quad + \max(\mathrm{T(E_{1.5})_{prob}}, \mathrm{T(E_{1.6})_{prob}}) \\ &\quad + \mathrm{T(E_{1.7})_{prob}} \\ \mathrm{T(WSC_1)_{prob}} &= \mathrm{T(E_{1.1})_{prob}} + \mathrm{T(E_{1.2})_{prob}} \\ &\quad + \mathrm{T(E_{1.4})_{prob}} + \mathrm{T(E_{1.5})_{prob}} + \mathrm{T(E_{1.7})_{prob}}\end{aligned} \tag{19}$$

With:

$$\begin{aligned}\mathrm{T(E_{1.1})_{prob}} &= \mathrm{T(E_{1.1})_{opt}} = \mathrm{T(E_{1.1}, e_{11}, ws_{11})} \\ \mathrm{T(E_{1.2})_{prob}} &= \mathrm{T(E_{1.2})_{opt}} = \mathrm{T(E_{1.2}, e_{21}, ws_{21})} \\ \mathrm{T(E_{1.4})_{prob}} &= \mathrm{T(E_{1.4})_{opt}} = \mathrm{T(E_{1.4}, e_{41}, ws_{41})} \\ \mathrm{T(E_{1.5})_{prob}} &= \mathrm{T(E_{1.5})_{opt}} = \mathrm{T(E_{1.5}, e_{51}, ws_{51})} \\ \mathrm{T(E_{1.6})_{prob}} &= \mathrm{T(E_{1.6})_{opt}} = \mathrm{T(E_{1.6}, e_{61}, ws_{61})} \\ \mathrm{T(E_{1.7})_{prob}} &= \mathrm{T(E_{1.7})_{opt}} = \mathrm{T(E_{1.7}, e_{71}, ws_{71})}\end{aligned}$$

2.4.2. Situation 2. Throughout the overall composition execution, we suppose that $E_{1.5}$ which was allocated for execution to (e_{51}, ws_{51}) failed. Fortunately, its execution retrial is possible since we suppose that another engine/web service couple (e_{52}, ws_{52}) does exist in its CEL. Thus a forward recovery is performed and Equation(19) becomes:

$$\begin{aligned}\mathrm{T}(WSC_1)_{\text{prob}} &= \mathrm{T(E_{1.1})_{opt}} + \mathrm{T(E_{1.2})_{opt}} + \mathrm{T(E_{1.4})_{opt}} \\ &\quad + \max(\mathrm{T(E_{1.5})_{prob}}, \mathrm{T(E_{1.6})_{opt}}) \\ &\quad + \mathrm{T(E_{1.7})_{opt}}\end{aligned} \tag{20}$$

With:

$$\begin{aligned}\mathrm{T(E_{1.5})_{prob}} &= \mathrm{T(E_{1.5})_{opt}} + \mathrm{I(E_{1.5})} + \mathrm{R(E_{1.5})} \\ \mathrm{I(E_{1.5})} &= \mathrm{I(E_{1.5}, e_{51}, ws_{51})} \\ \mathrm{T(E_{1.5})_{opt}} &= \mathrm{T(E_{1.5}, e_{52}, ws_{52})} \\ \mathrm{R(E_{1.5})} &= \mathrm{For(E_{1.5})} = \mathrm{T(E_{1.5}, e_{51}, ws_{51})}\end{aligned}$$

2.4.3. Situation 3. In this case, we suppose that $E_{1.5}$ was reattempted by (e_{52}, ws_{52}), failed, and that $CEL(E_{1.5}) = \emptyset$. Thus, a backward recovery needs to be triggered.

It entails undoing all the successfully terminated components with compensation mechanisms and aborting all the still executing components. Equation(20) becomes:

$$\begin{aligned}\mathrm{T(WSC_1)_{prob}} &= \mathrm{T(E_{1.1})_{opt}} + \mathrm{R(E_{1.1})} \\ &\quad + \mathrm{T(E_{1.2})_{opt}} + \mathrm{R(E_{1.2})} \\ &\quad + \mathrm{T(E_{1.4})_{opt}} + \mathrm{R(E_{1.4})} \\ &\quad + \max(\mathrm{T(E_{1.5})_{prob}}, \mathrm{T(E_{1.6})_{prob}})\end{aligned} \tag{21}$$

With:

$$\begin{aligned}\mathrm{R(E_{1.1})} &= \mathrm{Back(E_{1.1})} = \mathrm{Comp(E_{1.1})} \\ \mathrm{R(E_{1.2})} &= \mathrm{Back(E_{1.2})} = \mathrm{Comp(E_{1.2})} \\ \mathrm{R(E_{1.4})} &= \mathrm{Back(E_{1.4})} = \mathrm{Comp(E_{1.4})} \\ \mathrm{T(E_{1.5})_{prob}} &= \mathrm{I(E_{1.5})} + \mathrm{R(E_{1.5})} \\ \mathrm{T(E_{1.6})_{prob}} &= \mathrm{T(E_{1.6})opt} + \mathrm{R(E_{1.6})} \\ \mathrm{R(E_{1.5})} &= \mathrm{For(E_{1.5})} \\ &= \mathrm{T(E_{1.5}, e_{51}, ws_{51})} + \mathrm{T(E_{1.5}, e_{52}, ws_{52})} \\ \mathrm{R(E_{1.6})} &= \mathrm{Back(E_{1.6})} = \mathrm{Abort(E_{1.6})} \\ \mathrm{I(E_{1.5})} &= \mathrm{I(E_{1.5}, e_{51}, ws_{51})} + \mathrm{I(E_{1.5}, e_{52}, ws_{52})}\end{aligned}$$

2.4.4. Discussion. This case study allowed us to show that our proposal of estimating the overall composition *Execution Time*, starting from the very atomic level, that is the element level(with the mapped web-service/engine couple), and taking into consideration the failures (recovery and information time) is very promising.

In fact, major part of the proposed work addressing the same issue, they only compute the overall execution time of the overall process[16] or composition[17].

Moreover, they do not provide any distinction between faulty/correct executions. They only give a global idea about the cost in time.

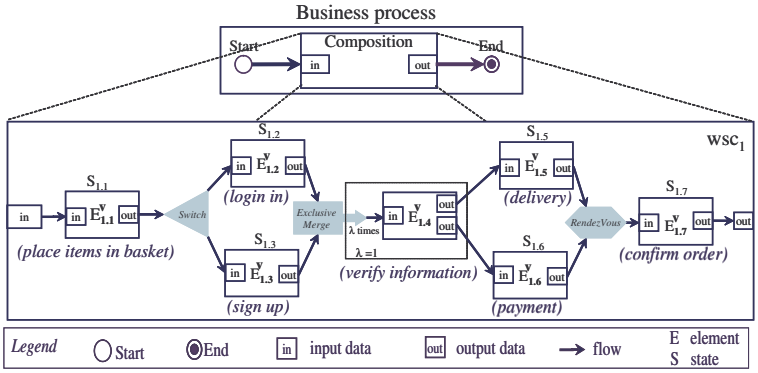

Figure 10. Enriched WS-SAGAS

However, our proposal, with estimating the *Execution Time* at the element level, and moreover, with considering all the possible execution cases (going from faulty-situations to the successfully-committed executions), is much more detailed.

Indeed, with only the time execution estimation, the more error-prone element(s) can be located, their failure reasons might be more easily investigated (i.e., CEL empty) ad eventually, the composition overall structure can be altered, if required, to improve the performances.

We emphasize here that our defined model for reliability and performance level estimations can apply to any other business process whether depicted as a transaction model, a Workflow or a flowchart or any other possible notation. In fact, in defining the new set of advanced patterns to enrich WS-SAGAS, one of our main concern was to keep them all-encompassing so that they can apply as well to other cases.

3. RELATED WORK

Over the last few years, there has been a relatively large amount of work underway in the area of aggregating web services into compositions. A logical next step to that is to provide the means to reckon the qualitative aspects of the specified web services compositions.

Specifically, taking the QoS into account for web services discovery and selection in order to give the consumers some confidence has been a very active area of research. The AgFlow middleware[17][18] has proposed a service quality model to evaluate the overall quality of composite Web services and two service selection approaches for composite service execution. Based on their QoS model, a global service selection approach that used linear programming techniques to compute optimal execution plans for composite services has been described.

In AgFlow, the potential failure reflections on the global execution time has not been considered. The reliability was mapped directly to each of the web service individual reliability. It was defined as the probability that a request to a particular web service is correctly responded within a maximum expected time frame (which is published in the Web service description). The value of the reliability was computed from historical data about past invocations.

Moreover, in AgFlow the underlying service composition model does support only parallelism and branching. However, in practice, business processes are more complex and considering other kind of components orchestrations is required.

[14] considered only relevant patterns that address the structure of a workflow in the process modeling phase and that no execution pattern is considered. Moreover, neither the failure was not taken nor its recovery was considered. Furthermore, we are assuming the case of a transactional execution of web services compositions, thus the assumption that [14] did on the fact that components are supposed to be working independently does not hold any more.

Other work in the area were oriented toward integrating the QoS criteria in the web services description to help once again users to acquire better results in their web service discovery. We cite here DAML-S [6] which supports the semantic description of web services based on a generic ontology in which both functional and QoS aspects of services are expressed as rule-based preconditions and post-conditions on service operations.

The DAML-S specification has semantically described processes. It has included constructs which specify several QoS parameters (e.g., the quality rating, the quality guarantees, the degree of quality). However, the QoS model adopted by DAML-S does not address the issue of dynamic service composition and composite services QoS.

Besides, it has not supplied any functional solution for its users. It has not allowed for a precise characterization of the different dimensions.

[19] proposed a web services discovery model in which the functional and nonfunctional requirements are considered for service discovery. It has proposed an extension to UDDIs data structure types. However, it did not establish any metrics to quantify each QoS category proposed.

While major part of the work done in web services compositions QoS is classified under the umbrella of maximizing user/customer satisfaction, very limited work is focusing on analyzing, estimating, and monitoring web services composition QoS. Indeed this was so far done in the very similar area of workflow.

The Crossflow project[20][21] and the METEOR project[12][13][22][23][24] have made major contributions. Specifically, the METEOR project has investigated four QoS dimensions, namely the time, the cost, the reliability, and the fidelity. However, the Crossflow project has not considered in any way the services dynamical composition. It has focused mainly on analyzing, predicting, and monitoring the QoS of workflow processes.

In our work, we dealt at the same time with, first, estimating the web services compositions performance and second, on considering the dynamic composition of web services. Furthermore, we consider a distributed transactional execution where state capturing contribute in reliability estimation.

As for the failure repercussions consideration in the reliability and time execution, and more generally QoS estimations, to the best of our knowledge, no work addressing the qualitative aspect of web services compositions has taken into consideration this issue.

In our model definition, we were somehow inspired from the most notable work in a very related area, which is the popular discrete-time stable reliability models proposed in[25] and shown below:

$$R(t) = 1-(\textit{system failure rate}+\textit{process failure rate}) \quad (22)$$

The system failure is defined as the ratio between the numbers of time a task did not perform and the number of times the task was called for execution. The process failure rate provides information concerning the relationship between the number of times the state done/committed is reached and the number of times the failed/aborted state is reached after the execution of a task. It is calculated using the formula #(failed or aborted)/(#(failed or aborted) + #(done or commit)). This model of reliability rate computation considers the coexistence of both transactional and none transactional tasks: in a workflow, task structure[26] has an *initial* state, an *execution* state, and two distinct terminating states. For none transactional tasks, one of the terminating states indicates that a task has *failed*, while the other state indicates that a task is *done*. For transactional tasks, the terminating states are *aborted* and *committed*.

This way of modeling the tasks is limitative since if other states do exist the reliability equation will be hardly extensible.

Finally, as it is described in the above equation, no matter the state is, the contribution in the reliability is always the same (coefficient always equal to one).

In our model, we consider the fact that not all the states contribute in the same way in the reliability computation. In fact, the *failed* state contributes negatively in the reliability. However, if the *committed* state is reached, a positive contribution is affected.

the Web-Flow architecture [4] presented an approach to dynamic exception handling in web-service-based processes that supports the specification of quality constraints for services, in addition to conditions a service may offer itself. A rule-based approach is used to handle exceptions such as the violation of constraints or other events (e.g. service faults) occurring during process execution. What was proposed in Web-Flow architecture can be classified under the same umbrella as what we did in THROWS and WS-SAGAS. In fact, they considered how to increase to quality with providing faults recovery. However, the estimation of the QoS dimensions was not addressed.

Finally our model can help the composition builder to acquire better estimations, and analyze them. On the base of these estimations, failures reasons might be more smoothly investigated, composition structure could even be altered, to improve the performances.

4. CONCLUSIONS

In this paper, we proposed to estimate the reliability and performance of web services compositions and to concentrate on one important aspect that has received little attention so far, which is the consideration of the failure repercussions on the overall web service composition execution performance.

In reaching this target, we enriched WS-SAGAS with a new set of advanced aggregation patterns so that it fits with the inherent business processes complexity.

Since when firstly we proposed WS-SAGAS, we only defined a set of basic aggregation patterns: sequential, parallel and recursive compositions, in this paper, we enriched WS-SAGAS with other advanced aggregation patterns, namely, Rendezvous of WS-SAGAS, Switch of WS-SAGAS, Selection of WS-SAGAS and so forth, to support the business process inherent complexity. For each of the defined patterns, the *Execution Time* estimation equation was derived. Our enriched WS-SAGAS model estimated two dimensions: the *Reliability* and *Execution time*, the most-easily influenced performance dimensions with failures. Extending our model with other dimensions is possible. In the Reliability dimension characterization, we introduced the *Reliability Tendency*, which builds mainly on the concepts of component's *State* and *States Tendency Set*.

The *Reliability Tendency* upheld the idea that from one state to another, the reliability contributions vary.

On the base of these estimations, potential executions reliability is ameliorated since failures detection is increased, error-prone components are more smoothly detectable, with failure-aware reliability estimations providing.

The reasons that lay behind the failures can be investigated and approximated. Finally, the chances to acquire more plausible estimations of the performance are notably ameliorated. Considering the time dimension, we introduced the notion of *probable execution time* and *optimistic execution time*, where the former considers all the possible system states and where the latter is only limited to the correct execution. Distinguishing between these two variants of the time dimension allowed us to provide more accurate time estimations.

Ongoing work comprises conducting experiments using the implemented simulation system of WS-SAGAS and THROWS architecture [11] to evaluate that our proposed model is indeed effective. We intend mainly to add the necessary module to collect the history of the past executions of the web services compositions, to automatize the performance estimation of each element, and of each composition execution instances.

Acknowledgment

A part of this research was supported by a Grant-in-Aid for Scientific Research of MEXT Japan(#16016232), by CREST of JST, and by the TokyoTech 21COE Program "Framework for Systematization and Application of Large-Scale Knowledge Resources".

References

[1] Telematica Institut Fundamental. Towards an adaptable qos aware middleware for distributed objects a.enschede, the netherlands, 2003 ctit phd.-thesis series number 02-46.

[2] ITU/ISO. Open distributed processing reference model, part 2: Foundations. International Standard 10746-2 ITU-T Recommendation X.902, 1995.

[3] *Peer-to-Peer Process Execution with Osiris*, volume 2910 of *Lecture Notes in Computer Science*. Springer, 2003.

[4] U.Greiner and E.Rahm. Quality-oriented handling of exceptions in web-service-based cooperative processes. In *EAI2004*.

[5] M.Keidl, S.Seltzsam, and A.Kemper. Reliable web service execution and deployment in dynamic environments. In *TES*, pages 104–118, 2003.

[6] A. Ankolekar. Daml-s: Web service description for the semantic web, 2002.

[7] B. Meyer. Applying design by contract. *IEEE Computer (Special Issue on Inheritance and Classification)*, 25(10):40–52, October 1992.

[8] N. Benlakhal T. Kobayashi and H. Yokota. Distributed architecture for reliable execution of web services. Technical report, IEICE, DBWS2003 2B, 2003.

[9] N. Benlakhal T. Kobayashi and H. Yokota. Ws-sagas: transaction model for reliable web-services-composition specification and execution. *DBSJ letters*, 2(2):17–20, Oct. 2003.

[10] N. Benlakhal T. Kobayashi and H. Yokota. Throws: An architecture for highly available distributed execution of web services compositions. In *RIDE WS-ECEG'2004*, pages pp.103–110, Boston, USA, March 2004. IEEE.

[11] N. Benlakhal T. Kobayashi and H. Yokota. A simulation system of throws architecture for ws-sagas. Technical Report 7-B-4, 14th IEICE Data Eng. Workshop, March 2004.

[12] J.Cardoso and A.Sheth. Semantic e-workflow composition. *Journal of Intelligent Information Systems*, 2003.

[13] G. Silver I. B. Arpinar S.Chadrasekaran, J. A. Miller and A. Sheth. Composition, performance analysis and simulation of web services. *Electronic Markets: The International Journal of Electronic Commerce and Business Media*, 2003.

[14] Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Mühl. Qos aggregation for service composition using workflow patterns. In *Proc. of the 8th Int. Enterprise Distributed Object Computing Conf.(EDOC2004)*, pages 149–159, Monterey, California, USA, September 2004. IEEE CS Press.

[15] P.Wohed, W.Aalst, M.Dumas, and A.H.M.Hofstede. Pattern based analysis of bpel4ws. Technical Report FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.

[16] J.Miller J.Arnold J.Cardoso, A.Sheth and K.Kochut. Quality of service for workflows and web service processes. *Journal of Web Semantics*, 2004.

[17] M.Dumas J.Kalagnanam L.Zeng, B.Benatallah and Q.Z.Sheng. Quality driven web services composition. In *the twelfth inter. conf. on World Wide Web*, pages 411 – 421, Budapest, Hungary, 2003. ACM Press.

[18] L.Zeng, B.Benatallah, A.Ngu, M.Dumas, J.Kalagnanam, and H.Chang. Qos-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5):311 – 327, May 2004.

[19] Shuping Ran. A model for web services discovery with qos. *ACM SIGecom Exchanges*, 4(1):1–10, Spring 2003.

[20] Paul W. P. J. Grefen, Karl Aberer, Heiko Ludwig, and Yigal Hoffner. Crossflow: Cross-organizational workflow management for service outsourcing in dynamic virtual enterprises. *IEEE Data Eng.Bulletin*, 24(1):52–57, 2001.

[21] J.Wasch J.Klingemann and K.Aberer. Deriving service models in crossorganizational workflows. In *RIDE-Information Tech.for Virtual Enterprises*, Australia, March 1999.

[22] J. Cardoso. *Quality of Service and Semantic Composition of Workflows*. Ph.d. dissertation, Department of Computer Science, University of Georgia, Athens, GA., 2002.

[23] A.Sheth J.Cardoso, J.Miller and J.Arnold. Modeling quality of service for workflows and web service processes. *technical report*, 2002.

[24] J. Cardoso J. Miller Sheth, A. and K. kochut. Service-oriented middleware. In *6th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, FL, 2002.

[25] E.C.Nelson. A statistical basis for software reliability assessment. Technical report, TRW Systems Report, March 1973.

[26] N.Krishnakumar and A.Sheth. Managing heterogeneous multi-system tasks to support enterprise-wide operations. *Distributed and Parallel Databases*, 3(2), 1995.