On automating networked enterprise management

Ustun Yildiz, Olivier Perrin, Claude Godart

On Automating Networked Enterprise Management

Ustun Yildiz, Olivier Perrin and Claude Godart

LORIA-INRIA
BP.239 Campus Scientifique
54500 Vandœuvre-lès-Nancy, FRANCE
{yildiz, operrin, godart@loria.fr}

Abstract. With the new middleware IT technologies such as Web Services and peer-to-peer computing facilities, a Virtual Enterprise can be built easier achieving some problems of interoperability. Although existing standards deal with syntactic issues they are primitive for the maintenance of VE while the processes are fulfilled in the untrusted and dynamic environment of the Web. In this paper, we investigate the problems of maintenance and propose a generic model that can be used for the automation purposes of monitoring and management. The mechanism we propose models the process of the Virtual Enterprise and the perspective of process manager. Thus, it automates the maintenance according to predefined configurations.

1 Introduction

The Web services standards such as UDDI [8], WSDL [3] and SOAP [10] make inter-organizational interactions easier than in the past. Like previous generations of middleware supports, they aim to facilitate application integration. Although they can achieve some of important interoperability problems and make loosely-coupled collaborations easier, they are too primitive for the automated invocation and composition of services and for the maintenance of built compositions. To deal with these issues, existing WS standards need to be supported with additional languages, architectures, and related approaches. We notice various efforts in the research literature such as BPEL4WS [7], OWL-S [5], WSLA [4] that aim to complement existing standards with additional features providing formal specification for automated discovery, composition, and execution of WS based virtual collaborations.

However, the use of a set of services in tandem to achieve a precise goal with a number of constraints, goes beyond the use of common data formats and exchange mechanisms. It requires complex, dynamic, adaptive mechanisms that permit to monitor and manage WS collaborative processes with minimal problems of reliability, performance and cost. In this paper, we review some difficulties of the automated monitoring which is the core part of process management frameworks and we explain our initial ideas about our monitoring approach and framework.

The rest of paper is organized as follows, the next section details the problem statement while Section 3 explains the overview of our approach. Next, we detail the mechanism of our approach. Section 5 gives a case study where our approach can be deployed. Section 6 reviews similar works, then we conclude and make a discussion about our future work.

2 Problem Statement

A Web service based Virtual Enterprise(VE) gathers autonomous organizations that provide particular capabilities to enact a collaborative business process. As in traditional business activities, the virtual collaboration has a number of goals such as the respect of mutual commitments, the increase of benefits, the decrease of risks and the protection of privacy. In the highly dynamic and untrusted environment of the Web, it is unrealistic to expect the outcome of a complex process to be as planned initially. From a process management automation point of view, the execution of collaborative process should be supported by additional mechanisms that permit the monitoring of its behavior. The observation of process activities helps process managers to analyze the performance of contributors, to detect anomalous behavior that can cause lately unwanted situations or to review management alternatives that can increase benefits. The challenge of an *ideal* monitoring agent -human or software- is to fulfill its task in a real-time, accurate, reliable and autonomous manner. The limitations of existing middleware technologies and antagonist nature of different business features make the setup of an ideal monitoring support very difficult. For example, if a monitoring agent requires a key data which characterizes the state of an observed service and if this data is private to the service that holds it then the outcome of the monitoring can not be efficient, or there is always a network delay between the occurrence and detection time of an event in the observed system. The second important issue is indirectly related to similar limitations, the process monitor decides what characterizes an undesired or wanted behavior of an activity while there are no certain and precise facts. An execution can be considered as a normal behavior by a monitoring agent and as abnormal behavior by another. Most of the time, the monitoring analysis can have fuzzy nature. Classical monitoring solutions that we detailed some in the Related Work section, try to compute conventional process properties such as deadlines or costs with exception based analysis. First, this approach that does not provide any predictive indication concerning the future states of activities can not be an efficient support for the ad-hoc nature of Web services context where there are numerous management alternatives at the disposal. It lacks for rich semantics and support that can permit process monitor to express its interested features.

We focus on pro-active and predictive business management which goes one step further comparing to classical management approaches. We try to provide monitoring analysis as close as possible to realistic facts. Our aim is to provide a generic support for human process managers permitting to define, compute their interested process measures.

3 Overview of our approach

Our approach aims to provide a support composed by structures, concepts and algorithms to facilitate the automated monitoring of WS based virtual enterprises. In this context, we consider a VE as an organization that crosses the boundary between the virtual and physical world. Because an information system that provides a service is a part of an overall process that makes effects in the physical world.

3.1 Virtual enterprise process model

This section provides a brief introduction to process models of virtual enterprises. The first step of the process management is the selection of the services to compose. The process manager chooses services according to their functionalities they provide and process constraints, it defines the dependencies that exist among them. The process can be initially scheduled or the services can be dynamically chosen during the enactment. The process manager can compose services using computation patterns(e.g. Workflow Patterns [9]). Although there is a flexibility, the process has a number of global constraints that must be respected such as the global cost or a deadline. Another important point that concerns the composition is the dependencies among services. Among dependencies, there are different degrees that can tightly couple one service with another. For example, a service is supposed to begin its execution at a precise date and it needs the output of another service at this date. If the former fails then it will cause cascading impacts on service that follow it. As a matter of monitoring, a system designer puts emphasis on critical dependencies.

3.2 Service behavior

The service behavior is the characterization of the instantiated service operation such as the way it produces and consumes events, it responds to invocations or it operates on business metrics. In the previous section, we mentioned that the monitoring outcome can not be the result of an *ideal* analyze. To model the monitoring analysis of a monitoring agent, we consider two kinds of monitoring analyze:

1. *Exact output (EO)*: The exact output is the result of an justified analyze done by a monitoring mechanism. In such cases, the output of monitoring describes a situation that really happened or will certainly happen.
2. *Indicational output (IO)*: The second type consists of analysis that give precise information about process states and behavior.

The *EO* is specified using predicates in first order logic and do not depend on any subjective analyze. The predicates depict relationships among involving entities. If the predicates that illustrate the *healthy* execution hold then the process enacts or will enact as planned. If the predicates that illustrate *unwanted* executions hold then the service can fail or can not be enacted as planned. For example, a service that uploads a certain amount data over a network will precisely fail if the rest of data can not be uploaded before the deadline using the maximum bandwidth of the service consumer.

The *IO* consists of sophisticated analysis that takes as input broadest view of the involving entities and gives indications about run-time process evaluation as output. The indications can be behavior analysis such as compliance, deviation or violation of expected behavior, performance analysis such as low, normal or outstanding performance, or any analyze required by process manager.

4 Putting automation into practice

The key of our approach is to configure the monitoring policies of a VE at high level abstract processes. The configuration consists of the mining of metrics and events involv-

ing in the process. Precisely, we choose measurable illustrations to express the analysis in order to provide a good support for the comparison of uncomplete or undetermined behavior.

4.1 Definition of entities to compute

The process that will be enacted by different services has a number of constraints and has an initial execution plan. Each service guarantees a number of commitments such as the beginning date, end date, their capabilities and associated qualities to provide. The process manager has the unambiguous knowledge of the process constraints and the agreed behavior of services. The second set of information to compute is the run-time behavior of the services. The run-time behavior expresses the properties of the operations done by the service related to its agreement with service consumer. The behavior of the service can be examined in a single instantiation or in loops. Thus, its behavior is stored in process logs and it is compared to its agreed execution. Besides these two observable and objective phenomena, the process manager defines an expected(or desired) execution of the used service. The expected behavior of a service can be defined using the critical relationships, dependencies, previous use of the service or with the expertise of the process manager. For example, there is a critical dependency between two services such as a service needs the output of its preceding before a precise date to begin its execution, otherwise it fails and causes a damage to process manager. If the preceding service has the habit to fulfill its task far before the deadline, the process manager can be concerned when it does not as previously or to make a reliable composition the process manager may want the service to fulfill its task as soon as possible. Contrary to classical approaches, we do not rely the expected behavior of the service only to its former execution results. It can be calculated by the run-time environment of that time also. For example, if there are circulation problems, a delivery service can be expected to be later than usual. The goal of monitoring framework is to check the conformity of the run-time behavior of the service and its expected behavior.

Process description We call $\mathcal{P}$, the description of the process. It consists of process constraints, actors, objects, services to compose and the qualitative and quantitative properties of the process. The Process Manager plans the initial execution of the process defining the relationships among the entities involved in the process. Briefly, the set $\mathcal{P}$ describes what a process is supposed to provide when it is started.

Process behavior We call $\mathcal{C}$, the current state of the process. The process behavior consists of features that illustrate the current state of services such as received QoS, fulfilled steps, underlying network activities or any information that can illustrate the current state of observed services. The $\mathcal{C}$ characterizes what the process has been doing since its start.

Process expected behavior We call $\mathcal{Q}$, is the description of the expected execution of process described by $\mathcal{P}$. For example, let's suppose there is service that consists of a good delivery before a deadline. The good can be delivered at any time between the activation date of the service and its deadline. If the service requester is concerned by

the failure of this service, it can consider to take precautions before the deadline. In case where there are no alternatives to execute before *x* hours before the deadline then this precise date can be a critical point of the service that the service requester expects the fulfillment of the service. The expected execution of a service can defined by using various features such as the reputation of the service. For example, let's suppose the service delivered the good always *y* hours after its invocation in the previous invocations then its behavior can be its expected execution.

4.2 Monitoring engine

We introduce a monitoring function $\mathcal{M}$ that takes a process(or a party of a process) as input, uses disposed information to analyze its state and returns *EO* or *IO*.
Due to lack of space and facilitation of the lecture, we consider a set of sub-services that compose a complex service that can be observed independently. The technique that we use for sub-processes can be easily thought for the process in general. A sub-process, itself, can be considered as a basic service. For example, a service that delivers several goods with different properties in different contexts after the reception of an order. We can consider the deliverance of each good as a sub-process. The motivation of our approach is to not detail complex interactions or various outputs that can have place but to show how the behavior of a service can be modeled.

Definition 1 (Monitoring function). *The $\mathcal{M}$ is a function that takes as input a running service and returns either an information that depicts a precise and justified state(EO) or analysis(IO) about this service.*
Let $\mathcal{S}$ the set of monitored services that compose the process described by $\mathcal{P}$,

$$\mathcal{M}: \mathcal{S} \longrightarrow [0,1]$$

The return value of $\mathcal{M}$ has different meanings, the two return values of $\mathcal{M}$ correspond to *EO*:

Let $s \in \mathcal{S}$, s is sub-process(or a service) of an overall $\mathcal{P}$,
if $\mathcal{M}(s)$ = 1 then the service is provided or will be certainly provided as planned,
if $\mathcal{M}(s)$ = 0 then the service is failed or will not be provided as planned,

The intermediate values in $]0,1[$ characterize the run-time behavior of monitored service, the convergence toward 1 can illustrate the completion of a running service as planned, or an outstanding performance. The convergence toward 0 can illustrate the deviation of a service from its expected behavior or an irregularity.
The algorithm below depicts partially the intern mechanism of $\mathcal{M}$ function while it illustrates *IO*.

- Γ is the set of predicates that illustrate the planned fulfillment of observed service, they consist of relationships between $\mathcal{P}$ and $\mathcal{C}$. In case they hold, the service is fulfilled or will be certainly fulfilled as planned,
- Λ is the set of predicates that illustrate the failure of a service contrary to its planned fulfillment, in case they hold the service is failed or will certainly fail,

- $s_i \in \mathcal{P}$, is an output, effect, or step of a service, it is described by quantitative and qualitative features such as a service that uploads data.
- $c_i \in \mathcal{C}$, is a set of information associated with run-time s_i. It can be the amount transfered data, current time, underlying network activities related to transfer etc.
- c_i^j is one particular aspect that characterizes the context. For example, c_i^j can be the temporal context of c_i, and c_i^{j+1} can the fact that concerns only the received amount of data,
- $\Delta_{c_i}^{s_i}$ depicts the relation of s_i and its run-time state c_i, $\Delta_{c_i}^{p_i}$ is composed by $\{\Delta_{c_i^0}^{s_i}$, ... $\Delta_{c_i^j}^{s_i}$, $\Delta_{c_i^{j+1}}^{s_i}$ ...$\}$, the states that describe the current state depending on single aspects. For example, the completion of the service can be related to received amount of data then the $\Delta_{c_i^{j+1}}^{s_i}$ will characterize the current state of service using the rate or difference of received and total amount of data to transfer. Normally, $\Delta_{c_i^{j+1}}^{s_i}$ will converge to 1 while the received amount of data increases, or will be constant while there is no reception of data.
- Each $\Delta_{c_i^j}^{s_i}$ has a weight in the composition of $\Delta_{c_i}^{s_i}$, we call a weight υ_i^j, and υ_i, the sum all weights,

Algorithm 1 : Monitoring function computation algorithm

$\forall\lambda \in \Lambda, \forall\gamma \in \Gamma, \Delta_{c_i}^{s_i} \in$ `]0,1[`,

1: **while**($\neg\lambda$ **and** $\neg\gamma$),
2: **for**(all running s_i)
3: **for**(all associated c_i^j)
4: $\Delta_{c_i}^{s_i} \leftarrow \Delta_{c_i}^{s_i} + \frac{\upsilon_i^j}{\upsilon_i}\Delta_{c_i^j}^{s_i}$
5: **end for**
6: **end for**
7: **end while**

The above algorithm depicts partially the monitoring function that computes a service *s*. We chose the part that concerns the analyze of a running or invoked s_i. We do not depict the predicates that illustrate *EO* as they consist of conditional relationships. This algorithm iterates(1-7) as long as the predicates that illustrate the *EO* do not hold. For each observed s_i, a $\Delta_{c_i}^{s_i}$ is calculated(4) using all of the different $\Delta_{c_i^j}^{s_i}$ and their weights(3-5). As the algorithm iterates $\Delta_{c_i}^{s_i}$ can have different values over observation interval. Continuous values of $\Delta_{c_i}^{s_i}$ can show the changes of the service since its invocation, its completion, its irregular behavior etc. The analysis made by single s_is can be gathered to have the *IO* of the service *s* in order to make more general analysis.

The operation of the monitoring function is the subject of the process manager. It leaves much room for configuration, the process manager can rely the state of a service to the properties it desires. If the outcome of a service consists of an instantaneous feature, it is hard to model its behavior because one property that can be interpreted is the response time to an invocation. But its behavior can be modeled in multiple instantiations using the quality of the its operation after each invocation.

4.3 Process decision engine

The decision engine uses the output of the monitoring function, it consists of comparing the run-time behavior of the service to its expected behavior. If the running service is not likely to be fulfilled or gives a bad performance, it can be aborted and a better service can be chosen. If the running service gives bad signs and the alternatives are not desirable because of their cost or risk, the running service can be kept. In this paper, we are not concerned by how one can define the expected behavior of a service using dependencies or previous executions, we will study it in the next work. To facilitate the selection of relevant execution plans, we define quality properties for each management decision that can be done.

The set of alternatives The set $\mathcal{W}_i$=$\{w_i^1, w_i^2, ..., w_i^n, ...\}$ is a finite set of actions that can be done when s_i of a service is enacting. Each element w_i^n of $\mathcal{W}_i$ has:
an associated cost ct_i^n, an income g_i^n, a risk r_i^n, and can have an associated set fc_i^n that includes its forward choices. We do not mention the set of $\mathcal{W}$ that includes all sets $\mathcal{W}_i$ for all the whole process that uses the observed service. The management decision consists of choosing one of the alternatives of $\mathcal{W}_i$ including the kept of the existing enactment. As the new services are discovered continuously, or when the number of possible actions decreases over the execution, the cardinality of $\mathcal{W}_i$ can change. The critical points that we have mentioned in 4.1 are actually the points that the elements of $\mathcal{W}_i$ are very limited. In the critical points, the elements of $\mathcal{W}_i$ can be only canceling or keeping the execution. We define a decision function $\mathcal{D}$ that takes a service and the set of management alternatives that can be done during its execution as input, and returns a alternative to execute as output.

Definition 2 (Decision function). *The decision function $\mathcal{D}$, is a function that takes as input a service and alternatives that can be performed. It returns the best decision that corresponds to its inputs.*

$$\mathcal{D} : \mathcal{S} \times \mathcal{W} \longrightarrow \mathcal{W}$$

As we did in the definition of monitoring function, we define $\mathcal{D}$ partially. The algorithm 2 computes the monitoring result of a service s_i with its alternatives within $\mathcal{W}_i$ and gives the best decision corresponding to the state of s_i.

- $\Delta_{c_i}^{s_i}$ is the result of monitoring analyze done for s_i
- $q_{i_{c_i}}$ is the expected state of s_i in the context c_i. For example, if c_i^j consists of data to transfer then $q_{i_{c_i^j}}$ is the expected amount of data that is supposed to be transfered,
- $q_{i_{c_i}}$ is calculated like $\Delta_{c_i}^{s_i}$, different weights are taken into account, we call ϑ_i^j, the weight of each $q_{i_{c_i^j}}$ and ϑ_i, the sum of all weights,
- m_{s_i} is the depicts the deviation from expected execution,
- $\ominus$ is an operator that returns the difference of an expected execution and current execution,

Algorithm 2 : Management function computation algorithm

$\forall\lambda \in \Lambda, \forall\gamma \in \Gamma, q_{ic_i} \in$ `]0,1[`

1: **while**($\neg\lambda$ **and** $\neg\gamma$)
2: **for**(all running p_i)
3: **for**(all associated c_i^j)
4: $q_{ic_i} \leftarrow q_{ic_i} + \frac{\vartheta_i^j}{\vartheta_i} q_{ic_i^j}$
5: **end for**
6: $m_{s_i} \leftarrow \Delta_{c_i}^{s_i} \ominus q_{ic_i}$
7: **if** $m_{s_i} \geq 0$ **then** keep execution **else**
8: review alternatives **end if**
9: **end for**
10: **end while**

In the above algorithm, for each s_i an expected execution is calculated(3-5). The algorithm calls the monitoring function(6) and uses its return value $\Delta_{c_i}^{s_i}$ in order to compare to its expected value. The management policy we used in this algorithm is very basic, if the expected and current state are equal or there is an outstanding performance then the running service is kept(7), else the management alternatives(besides keeping the execution) are reviewed(8). This process can be resulted by the choice of an alternative which is better than keeping the running service. The alternatives are taken into account basic properties such as income, cost, risk and the alternatives that can follow them.

To compare the expected and current state of a running service, we use a special operator of comparison ($\ominus$) that expresses only concerned differences. The algorithm we used, can be extended easily to express the general behavior of a service, for example the record of differences between expected and current state of all s_is can be illustrated as its general quality over its use of its sub-processes.

5 A case study: Distributed software production

This section presents a simple example of collaborative software production process. According to [1], it has been estimated that the software development industry has an 85% failure rate in the development of software products. This means that most of the time, the outcome of a software production does not accomplish its planned results. When the development occurs over the Web, the production becomes more difficult. The processes and entities involving in them should be rigorously managed.

The scenario we use in this paper can be resumed as follows, the `software house(SH)`, a software development agent that produces software for end-users, has customers that place orders including the complete and unambiguous description of the software they require. The `SH` makes a production planning and dispatches different components and steps of production among its partners(`service`(s)) that provide corresponding services(functional core development, user interface development, delivery to end-user, payment service ...). The agreement of `SH` and its customer depicts the aspects related to quality of required software such as the number of time that the

software can crash in a precise period or the execution time of a operation and the features related to classical production process such as product delivery date or payment process. The SH plans an initial production and makes best effort to respect its agreement. During the production process, SH has initiatives. Basically, it can abort a service that does not show its expected performance, it can choose a cheaper service instead of an activated service canceling the latter, or for a risky dependency it can choose providers having good reputation. Due to lack of space, we explain one step of the complete production scenario. The figure 1 depicts three services used in the produc-

Service CD
Total size = 10000 KB
QoS = 30 crashes for 1000 op.
Completion time = 10 days

Table A

	Delivery over 10 days	Test results
1.day	200KB	8 crashes
2.day	2000KB	24 crashes
3.day	500KB	5 crashes
4.day	800KB	6 crashes
5.day	800KB	13 crashes
6.day	500KB	20 crashes
7.day	3500KB	12 crashes
8.day	1000KB	4 crashes
9.day	500KB	18 crashes
10.day	200KB	6 crashes

Table B

Fig. 1. An example of service composition

tion, `Core Development(CD)` service, `GUI Development(GD)` service, and `Delivery(DS)` service. There are serial dependencies among services, `GD` and `DS` use the outcome of the preceding services. The s_i to observe consists of the production of a software functional core composed by independent components(which has 10MB size), an expected quality(less than 30 crashes for 1000 operations invoked), all of the components must be delivered at most in 10 days. In this example, we suppose the separate components are delivered when their production are completed. The elements we mentioned above, describe the service outcome to observe. The system designer defines a monitoring and management policy for this service. The corresponding *IO* of this service can be defined by various ways. The completion of service can be associated to the amount received components proportionally to the total expected amount. In this case, the *IO* of the service converges toward 1 according to the relation of received and total expected amount of components to transfer. In the figure 2, graphic(1) illustrates the *IO* associated with this relation. As example of the delivery, we used the data of Table B in the figure 1. The implementation of corresponding $\mathcal{M}$ function is the proportion of received and total amount of data. If the system designer lies the *IO* to this relation then the default output of $\mathcal{M}$ is a certain value close to 0. We can not

consider 0 because, it corresponds a service which is not fulfilled as planned and also we can consider 1 as a return value if the service is fulfilled or certain to be fulfilled as planned. In the graphic(3), the system designer that is interested in temporal context associates the output to the relation of current time and deadline. Thus, he does not take the other metrics into account, he considers the service approaches to its failure as long as its is not fulfilled completely. In this interpretation, logically, the default state of the service is very close to 1 but its not equal either to 1 or to 0 as long as there is certain information about the fulfillment or failure of the service. In the same figure, the graphic(2) illustrates the state of service according to the number of crashes detected for each delivered component. As 30 crashes characterizes the failure of the service, the output is close to 1 when the number of crashes is not high. In the same figure, the

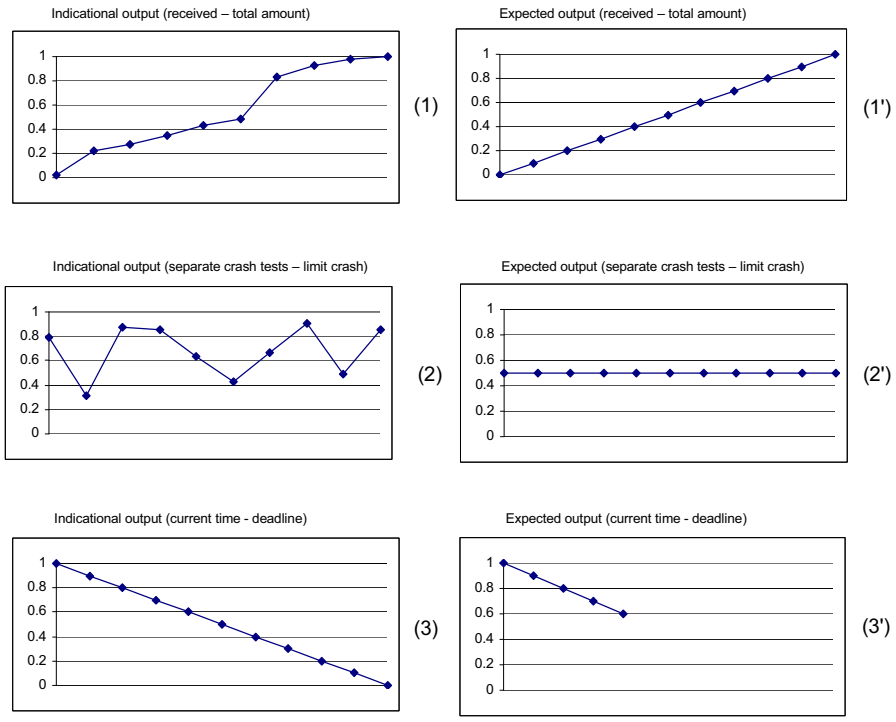

Fig. 2. Examples of monitoring output and corresponding expected states

graphic(1') depicts the expected state of the service according to the relation of (1). The intention of system designer consists of the expectation of regular component delivery that will begin with the invocation and end before the deadline. If we compare visually or arithmetically two illustrations, there is no important deviations. In the graphic(2'),

the expectation of the system designer consists of a constant quality for each delivered product. In this case, the comparison of the service behavior to expected behavior gives the signs of deviation from expectations. The expectation depicted in graphic(3') can be interpreted as follows, the service is expected to complete its fulfillment before the deadline. In our example, the service we considered ends its fulfillment just before its deadline. As result, at a certain date, the management function considers this service deviates from its expected execution.

The most efficient way of computation is to gather all of separate illustrations in a single illustration. Their importance in the composition can be characterized with the weights. Actually, the graphics(1, 2, 3) correspond to three $\Delta^{s_i}_{c_i^j}$, the graphic that will fusion them with their weights will be $\Delta^{s_i}_{c_i}$. The graphics(1', 2', 3') correspond to three $q_{i_{c_i^j}}$ and respectively their fusion will be the $q_{i_{c_i}}$ of the service.

In the graphics, we do not label the x line, the observation can occur over the time or single events.

6 Related Work

The work of Xu [11] is one of the closest works to ours. The authors consider the collaborative process as an e-contract. The parties have predefined mutual commitments composed by predefined actions series. Within the commitments, the actions are interconnected with temporal relationships that called constraints. The commitments are observed during the enactment, the monitoring party puts guards on constraints, the guards capture how far the commitments have progressed. If there is a non-compliance to prescribed scheme of actions, the concerned parties are notified.

Sayal et al. present in [6] a set of integrated tools that support business and IT users in managing process execution quality. The authors provide a data model and a generic architecture(HP Process Manager) to model and compute the execution of processes. Although the approach is interesting, it suffers heavy analyze of workflow log data.

Cardoso et al. [2] propose a model to manage workflow components from a QoS perspective. The QoS presented includes three dimensions: time, cost, and reliability. The QoS measures are applied separately to each workflow task and then they are automatically computed for the overall QoS of the workflow.

In the literature, there are many examples of Web service composition propositions that rely the behavior of a Web service to its network level activities such as response time to any invocation, availability, the probability of response to any invocation etc. In these approaches, the metrics and events that parameterize the behavior a service do not characterize the process manager perspective. In our work, we are mainly concerned by high level business properties of the processes.

7 Conclusion and outlook

In this paper, we examined the problem of VE maintenance within the context of WS. In the dynamic context of the Web, each process manager has an individual policy that governs the processes. We proposed a generic mechanism that can be used to express

monitoring and management policies computing basic semantic features of services. Our proposition includes a monitoring function that can compute basic relationships among process metrics, events, and objects. The monitoring is processed with measurable and continuous indications of how likely the process is enacted. The management mechanism we propose, uses a similar mechanism to deal with different alternatives. It permits the automated comparison of different alternatives according to their basic properties.

This paper depicts the preliminary concepts our future work. Our aim is to provide a framework that can monitor and verify the run-time behavior of the Web processes and also manage the composition of Web services according to the management policies. The model we proposed requires a complex data model that will express basic properties of processes and the features that characterize their key performance indicators. The technologies that can support our effort are data warehousing and mining techniques, complex event processing, workflow and business policy management approaches.

References

1. Jones Caper. Minimizing the risks of software development. *Cutter IT Journal*, 11(6), June 1998.
2. Jorge Cardoso, Amit P. Sheth, John A. Miller, Jonathan Arnold, and Krys Kochut. Quality of service for workflows and web service processes. *J. Web Sem.*, 1(3):281–308, 2004.
3. Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. *Web Services Description Language (WSDL) 1.1*. W3C, 1.1 edition, March 2001. URL: http://www.w3c.org/TR/wsdl.
4. Alexander Keller and Heiko Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *J. Network Syst. Manage.*, 11(1), 2003.
5. David Martin(editor), Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. *OWL-S: Semantic Markup for Web Services*, 2004. URL: http://www.daml.org/services/owl-s/1.1/.
6. Mehmet Sayal, Akhil Sahai, Vijay Machiraju, and Fabio Casati. Semantic analysis of e-business operations. *J. Network Syst. Manage.*, 11(1), 2003.
7. T.Andrews, F.Curbera, H.Dholakia, Y.Goland, J.Klein, F.Leymann, K.Liu, D.Roller, D.Smith, S.Thatte, I.Trickovic, and S.Weerawarana. Business process execution language for web services. BEA, IBM, Microsoft, SAP, Siebel, 2003.
8. UDDI. *Universal Description, Discovery, and Integration of Business for the Web*, October 2001. URL: http://www.uddi.org.
9. Wil M. P. van der Aalst, Boudewijn F. van Dongen, Joachim Herbst, Laura Maruster, Guido Schimm, and A. J. M. M. Weijters. Workflow mining: A survey of issues and approaches. *Data Knowl. Eng.*, 47(2):237–267, 2003.
10. W3C. *Simple Object Access Protocol (SOAP) 1.1*, 2000. URL: http://www.w3c.org/TR/SOAP.
11. Lai Xu and Manfred A. Jeusfeld. Detecting violators of multi-party contracts. In *CoopIS/DOA/ODBASE (1)*, pages 526–543, 2004.