Ensuring time in service composition

E. del Val, M. Navarro, V. Julian and M. Rebollo
Departamento de sistemas informáticos y computación
Universidad Politécnica de Valencia
Camino de Vera S/N
46022 Valencia (Spain)
{edelval,mnavarro,vinglada,mrebollo}@dsic.upv.es

Abstract

Time is an important non-functional parameter to consider in service compositions, especially in environments where a service must be provided before a deadline. This paper presents a framework that deals with service compositions taking into account the service execution time. To enhance this composition it is important to provide service execution times with reliability, bearing in mind the workload and availability of the service.

1. Introduction

Service Oriented Architectures (SOA) are composed of groups of independent services that communicate or interact with each other. Services can be considered auto-contained pieces of autonomous code that provide their 'clients' with basic functionalities. SOA [1] is the base of current computation models such as grid [2] and cloud computing [3].

One of the main problems of SOA is how to create added-value services dynamically by composing elemental services. Services can be seen as elemental instructions and they are commonly used by human developers to create bigger systems. Semantic annotations help machines to deal with services, but service discovery and composition are complex tasks that need extra intelligence doses to achieve proper results, especially in open and dynamic environments where services are not always available.

Expressive languages [4][5][6][7] have been used to describe services and to deal with complex services composition. A drawback common to all standard service description languages is the impossibility of modeling how time pass in the system. Some approaches have been done to include temporal service information to enrich semantic descriptions [8][9][10]. This information is important to consider due to some services could be completely useless if they are not provided on time. For this reason, the temporal cost associated to the service execution should be known and service descriptions should contain it. It is possible to define a *real-time service* as a service with temporal restrictions in its execution. A real-time service must be executed over a service provider with the capability to execute real-time tasks in order to guarantee the fulfillment of the temporal restrictions.

In this way, compositions of real-time services must be realized taking into account temporal restrictions established by the client. But, this is not enough to guarantee that these compositions would be fulfilled on a estimated time. It is necessary to consider a service provider workload at the moment when a client makes a request. It could be possible that if a service provider is already executing several services, probably it does not have enough time to attend a new request and therefore the service composition could not be possible. So, in order to evaluate if an initially suitable service composition is feasible, it is necessary that each service provider informs about its availability to attend the request.

This paper presents SAES (Search And Execution Services) framework which allows to compose services and to ensure their fulfillment on time. To do that, the service considers the current workload and the availability of the real-time service providers.

The sections of the paper are structured as follows: In Section 2, a general description of the SAES framework is presented. In Section 3, a module in charge of making the composition and a planning technique for dealing with the service composition is presented. Furthermore, a temporal extension for OWL-S descriptions is shown. Next, in Section 4, the module in charge of consulting the execution time of the services is described. Section 5 presents the Real-Time Service Provider which is in charge of executing real-time services. Section 6 shows an example of the SAES being used. Finally, conclusions and final remarks are presented in Section 7.

2. SAES Framework

To ensure the correct service composition and also that the goal expected by the client is satisfied within the deadline the SAES (Search And Execution Services) framework has been developed. SAES is a framework

978-0-7695-3708-5/09 $25.00 © 2009 IEEE
DOI 10.1109/SERVICES-I.2009.91

which allows a client to search complex services that satisfy its requirements and to guarantee its execution in a maximum time (deadline) established by the client. To fulfill these functions, the SAES is composed of two modules (Figure 1):

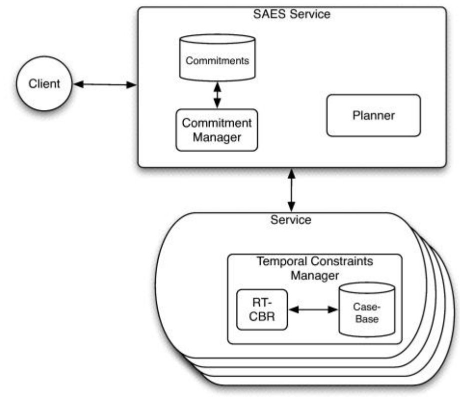

Figure 1. SAES Framework

- **Service Composer Module**: The Service Composer Module (SC) has as objective to get service compositions which fulfill a set of constrains. These compositions are based on theoretical information without taking services current situation into account (workload, shared resources availability).
- **Commitment Manager Module**: This module analyzes the service composition and the current situation of services that compose it. The Commitment Manager Module (CM) checks if a service composition is feasible and whether it can be executed on time. Moreover, this module monitors service execution in order to take it into account the observed performance for similar future situations.

Furthermore, service providers must have mechanisms to analyze whether tasks associated with the service can be executed within the deadline. Besides that, providers should be able to detect missed deadlines. In other words, providers should be able to offer temporal-bounded services. So, in our system, the concept of Real-Time Service Provider (RTSP) is introduced as a provider with mechanisms that allow the execution and control of temporal-bounded services. The SAES works as follows:

1. Initially a client sends a query to determine if there is a set of services which could satisfy a goal before a deadline established by the client.
2. The request is received by the SC, which starts the search process. The aim of this process is to find service compositions which fulfill the client's goals taking into account the temporal restriction established by client. To consider service execution time in this process, service descriptions are extended with service durations as a non-functional parameter.
3. When a service composition is obtained, the SC sends it to the CM and continues searching for more alternatives. This search will continue until the SC receives a message from the CM to finish the search process. The search process can also finish when more compositions are not found.
4. After that, the CM queries each RTSP involved in the service composition asking for the provider's availability to execute the service.
5. Each RTSP analyzes the service execution time according to its current workload and returns it to the CM.
6. The CM establishes pre-commitments with the RTSPs involved in order to reserve the slack time for the service during a period of time. The CM sends the resulting service composition to the client with a success probability.
7. If the client agrees with the service composition, the CM confirms commitments with the RTSP involved. Otherwise, the client sends a message rejecting the service composition and the CM breaks the established pre-commitments with the RTSPs.
8. Once service executions start, the CM monitors the fulfillment of the committed real-time services. When a service ends its execution, the CM stores the execution time in order to take it into account in future queries. Should the RTSP not fulfill its commitment, it will be penalized.

The following sections describe the modules that form the proposed SAES framework and the Real-Time Service Provider in more detail.

3. Service Composer Module

If the client requirement cannot be solved only with a service, then a possible solution is to automatically compose several services in a sequence. However, dynamic service composition is a complex problem and it is not entirely clear which techniques are the best. There are several proposals that use techniques such as hypergraphs [11][12], modelchecking [13][14] or planning [15][16][17][18][19] to deal with this problem.

Service composition as planning describes a service as a process in terms of inputs, outputs, preconditions and effects. Using the metaphor of an action, composition can be viewed as a planning problem. An important benefit of the planning approach is the use of knowledge that has been accumulated over years of research in the field of planning. Therefore, well know planning algorithms, techniques and tools can be used to take advantage of efficient and seamless service composition. The desired outcome of the service is described as a goal state, while simple services play the role of planning operators or actions. The planner is then responsible for finding an appropriate plan (sequence of services) to achieve the goal state.

In SAES, the SC provides a service composition (or a set of service compositions) which satisfies the goal and the time deadline established by the client. To do this, the SC considers the execution time of services and employs *Artificial Intelligence planning* techniques to automate this process. Basically, the idea is to translate OWL-S service descriptions temporally annotated to PDDL durative actions in order to generate a plan.

The SC is composed by three components:

- **Service Translator Service**: Responsible for translating OWL-S service descriptions extended with a duration non-functional parameter into PDDL 2.1 durative actions [20].
- **Problem Translator Service**: Responsible for translating the client query into a PDDL 2.1 problem description.
- **Composition Service**: Takes as input a PDDL 2.1 problem and provides a set of plans which represent service compositions that satisfy the client's request and temporal constraints.

Before explaining how the SC works, the OWL-S service description extension to consider service estimated execution time is presented.

3.1 Temporal Extension in OWL-S Service Descriptions

A common drawback to standard service description languages is the impossibility of modeling how time pass in the system. There are some proposals that try to include temporal service information to enrich semantic descriptions using non-functional parameters [8] or techniques based on Interval Temporal Logic (ITL) [9][10]. Some services could be completely useless if they are not provided on time. For doing that, time is an important factor for services and might be taken into account.

The execution time of a service should be expressed in its service description. Service descriptions in SAES are expressed in terms of OWL-S, which has been extended to include temporal information. This extension consists of a new non-functional parameter which represents service time execution. Besides that, preconditions and effects are time-stamped annotated.

Duration. Service duration is represented by a non-functional parameter called *duration* which represents the service time execution. This parameter is a PDDXML expression that contains a value assignment to the variable duration. PDDXML [21] is a XML dialect of PDDL that simplifies parsing, reading, and communication PDDL descriptions using SOAP.

```
<Duration_param:hasLocal>
 <duration:Duration-Expression rdf:ID="PDDXML-Duration">
  <expr:expressionBody
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
   <and><equals>
    <variable><var type="object">?duration</var></variable>
    <constant><const type="int">8</const></constant>
   </equals></and>
  </expr:expressionBody>
 </duration:Duration-Expression>
</Duration_param:hasLocal>
...
<process:hasPrecondition>
 <pddxml:PDDXML-Condition rdf:ID="PDDXML-Precondition">
  <expr:expressionBody
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
   <and><atStart><not>
     <pred name="agentHasKnowledgeAbout">
      <param>?http://.../Packing/GetItems.owl#FinishEvent</param>
     </pred>
   </not></atStart></and>
  </expr:expressionBody>
 </pddxml:PDDXML-Condition>
</process:hasPrecondition>
```

Figure 2. Non-functional parameter duration and temporal precondition

Preconditions. The annotation of a precondition makes it explicit whether the associated proposition must hold:

- *at the start* of the interval (the point at which the service is applied)
- *at the end* of the interval (the point at which the final effects of the service are asserted)
- *over* the interval from the start to the end (invariant over the duration of the service)

The preconditions in the OWL-S document are also PDDXML expressions (Figure 2).

Effects. The annotation of an effect makes it explicit whether the effect is immediate (it happens at the start of the interval) or delayed (it happens at the end of the interval). No other time points are accessible, so all discrete activity takes place at the identified start and end points of the service. Service effects are PDDXML expressions in the OWL-S document.

Preconditions, effects and duration parameters that appear in the OWL-S description will be translated into preconditions, effects and duration in a PDDL 2.1 action. The *Composition Service* will consider them in the composition process.

3.2 Composing Temporal Services

Basically, the SC works as follows. When a service is registered in the system, the SC takes the OWL-S service description and sends it to the Service Translator. The

Service Translator is responsible for translating OWL-S descriptions into PDDL 2.1 durative actions. Once the *Composition Service* has the services modeled as actions, it generates a PDDL domain file which contains the definition of the actions structured as a planning problem. This process is an extension of a converter presented by Klusch and Gerber [21] which is limited to dealing with services that are not time-stamped annotated.

When a query arrives at the SC the composition process starts. The client query is an OWL document that contains the information related to the inputs of the desired service and the goals (outputs) to be achieved. This file is sent to the *Problem Translator* to be translated. First, the *Problem Translator* translates the OWL file into an equivalent one in PDDL 2.1 language.

Once the service composition problem has been translated into a planning problem by the *Problem Translator*, the domain and problem files in PDDL 2.1 are sent to the *Composition Service*. The *Composition Service* is a planner which deals with PDDL 2.1 language and it have to consider time as a parameter to optimize the plans. In this proposal the planner is only used as a tool for obtaining service compositions. It is not a goal of this work to study in-depth the use of planning for service composition. This planner obtains a plan composed of actions (services) with their initially estimated duration and the total estimated time of the plan. The plan represents a service composition sequence that satisfies the goal considering temporal annotations. The *Composition Service* will continue searching for plans until the CM sends the SC a message to finish the search process because a previous plan has been accepted. Besides this, the *Composition Service* stops if no more plans are found.

4. Commitment Manager Module

The CM has two main functions: (i) to check if the set of services offered as a solution by the SC will be available to fulfill the request; and (ii) when the client select a service composition, the CM must establish a commitment relationship with the RTSPs that the selected services provide. Besides, a RTSP must control that its own services will be executed correctly (achieve its commitments).

To fulfill the first function, the CM must communicate with all of the RTSPs that offer the services involved in the composed service. Each RTSP analyzes when it can complete the service for the CM and it returns the result to CM. The result consists of a tuple $<T_{start}, T_{duration}, SP>$ where T_{start} indicates the moment when the service can start its execution, $T_{duration}$ indicates the necessary time to complete the service and *SP* is the probability of a successful execution. Moreover, a pre-commitment between the RTSP and the CM is established.

When all RTSPs have responded to the CM, it must calculate the success probability associated to the whole service composition. For doing that, the CM uses the success probability sent for all RTSPs weighted with the information of previous executions of similar services. The service composition success probability is calculated as follows:

$$SP_{compositioin} = \prod_{i=0}^{N} SP_i * \omega_i$$

where $\omega_i \in [0,1]$ is the weight associated to the service *i*. This weight is related to the previously fulfilled commitments; A RTSP who has many unfulfilled commitments will have a low weight.

Once the CM calculates the service composition success probability, it sends the client the composed service and its probability $SP_{composition}$. The client analyzes if it is a suitable composition. If the client agrees with the service composition, the client communicates to CM that the service executions can start. When this is the case, the pre-commitments established with the RTSPs are confirmed by the CM. If the client does not agree with the service composition, the CM breaks the pre-commitments, freeing the slack reserved by the RTSPs.

The CM is also in charge of ensuring that the acquired commitments are fulfilled. In case where a commitment cannot be fulfilled, the CM penalizes the RTSP which provides the service. This penalty is captured through the weights applied when the CM calculates the service composition success probability.

5. Real-Time Service Provider

As previously pointed out, the RTSP is in charge of executing the real-time services. Besides this, the RTSP analyzes when a service can be executed without exceeding the maximum time proposed by the client. In Figure 3, the architecture of the Real-Time Service Provider is shown. In order to guarantee the correct execution of services offered by RTSP, it is necessary that the RTSP runs over a Real-Time Operating System (RTOS). Otherwise, the system behavior becomes unpredictable and the RTSP loses the ability to analyze whether it can commit to perform a service within a deadline.

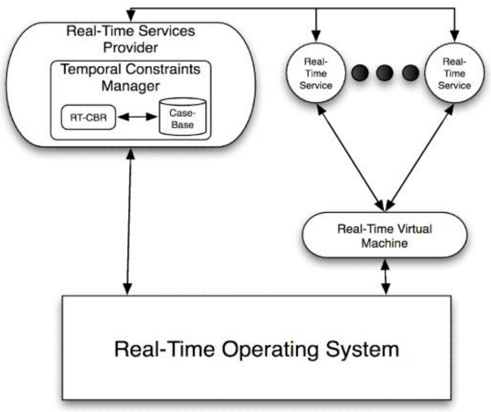
Real-Time Services Provider
Temporal Constraints Manager
RT-CBR
Case-Base
Real-Time Service
Real-Time Service
Real-Time Virtual Machine
Real-Time Operating System

Figure 3. Real-time service provider architecture

To determine whether a service can be executed on time, it is necessary to know the execution time for each service. In some cases, the execution time of the service is known and limited. In these cases to determine the necessary tasks to fulfill the service and the maximum time needed to perform it is relatively easy using well-known scheduling techniques [22] [23].

Otherwise, there are services for which to calculate the needed execution time is not possible. In this type of services, a time estimation is the unique measure that can be made. In order to do this estimation, each provider that offers real-time services incorporates a module, called Temporal Constraint Manager (TCM). This module, using previous experiences, is able to make an accurate prediction of whether a service will be completed within the time specified by the CM.

5.1 Temporal Constraint Manager

The Temporal Constraint Manager (TCM) is a module inside the RTSP that must decide if it can commit to perform a specific real-time service. A possible way of performing such decision-making functionality is to use a Case-Based Reasoning (CBR) approach, which adapts previous problem solving cases to cope with current similar problems [24]. Therefore, in the absence of unpredictable circumstances, we assume that an agent can commit itself to perform a service within certain time if it has already succeeded in doing so in a similar situation (CPU utilization, resources and service availability). To carry out the decision-making about contracting or not a commitment, the TCM has been enhanced with a RT-CBR (Real-Time CBR), following a soft Real-Time approach. This RT-CBR incorporates a temporal-bounded decision process that estimates the time that a service performance could entail. This task is carried out using the time spent in performing similar services.

The classical CBR cycle consists of four steps: *Retrieve, Reuse*, *Revise* and *Retain*. A CBR *Retrieves* similar experiences from a *case-base*, *reuses* the knowledge acquired in them, *revises* such knowledge to fit the current situation and, finally, *retains* the knowledge learnt from this problem-solving process. In our framework, the CBR phases must observe soft real-time constraints and thus, its execution time must be bounded. Otherwise, the RT-CBR could provide the TCM with useless time estimations about services whose deadline have already expired.

To bound the response time of the TCM, the RT-CBR case-base must have an structure that eases the case retrieval. Anyway, independently of the choice made about the indexation, the temporal cost of most retrieval (and reuse) algorithms depend on size of the case-base. This entails to specify a maximum number of cases that can be stored in the case-base and to perform a constant maintenance and updating of the information stored. Each agent that offers real-time services must have a specific implementation of its RT-CBR taking into account its application domain.

Figure 4 shows the execution phases of the TCM. The module is launched when the agent begins its execution. At the beginning, the TCM controls if a new service request has arrived (Figure 5). If the new request is a service request where the service execution time is not known, the TCM must estimate the time required to execute that service. It is necessary to determine if the service can be completed before the deadline specified in the request. When the estimated time is obtained and the provider confirms that it is possible to execute the service, the necessary tasks to perform the service must be analyzed at low-level using a real-time scheduler. The worst-case execution time of each phase of the TCM is known and, therefore, the phases are temporal bounded. This feature is crucial to allow the TCM to have a precise time control of each execution phase. As it can be seen in Figure 4, the TCM execution is cyclical. When there is no request, the manager can employ its idle time to perform the *revision* and *retention* phases in order to learn about experiences.

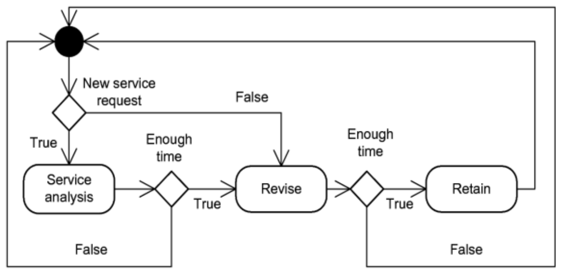

Figure 4. Temporal constraints manager algorithm

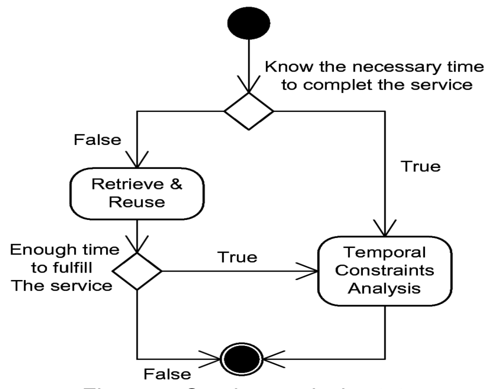
Know the necessary time to complet the service
False
True
Retrieve & Reuse
Enough time to fulfill The service
True
Temporal Constraints Analysis
False

Figure 5. Service analysis state

6. Example: Packing Cell

In order to illustrate this proposal, an example in a manufacturing environment is presented. Our proposal is suitable for manufacturing systems as they have been facing a continuous change over the last few years. Rapid static and hierarchical manufacturing systems will give way to systems that are more adaptable to rapid changes. Moreover, the diversity of customers' demands is increasing. This situation makes the system change the configuration of the manufacturing cell in a dynamic way with the arrival of new product demands. All of these factors result in the requirement for manufacturing to be more efficient and time-critical in order to bring in new products at the right time.

6.1 System Description

The scenario in which to apply the presented proposal is a *packing cell*. The *packing cell* supplies gift boxes with a set of products inside [25]. The possible actions in this cell are presented as services offered by entities. The aim of this cell is: to find the least time consuming service composition in order to respond to the arrival of a rush order.

The packing cell is composed of five types of RTSPs: *DockingStation*, *Robot*, *Order*, *Storage* and *Wrapper*. Each RTSP has associated services (Table 1) that represent the tasks that the entity can carry out. The configuration of the different entities available in the cell depends on the client product demand order. If the order received is a *rush* order with temporal constraints, it is possible that not all of the services provided by the RTSPs will be able to accomplish their activities before a deadline. In other situations, a service offered by an RTSP may not be available due to it being busy attending other client orders.

In this scenario, each RTSP is executed over a real-time operating system (Suse Linux Enterprise Real-Time 10) in independent machines. To execute services is necessary a real-time virtual machine (Sun Java Real-Time system) due to the fact the services are implemented using a RT-Java (Real-Time Java Specification version 1.1).

Provider Entity	Service	Inputs	Outputs	Estimated Duration
Docking	LockShuttle	ShuttleEvent OrderEvent	LockshuttleFlag NotificationEvent	3
	UnLockShuttle	FinishEvent LockShuttleFlag	UnLockshuttleFlag	2
Robot.	GetItemsOp	NotificationEvent Material Stock List of Item Types	FinishEvent	5
	GetItems	NotificationEvent Material Stock List of Item Types	FinishEvent	8
Order	GetOrder	NotificationEvent	List of Item Types OrderCode	4
	SendOrder	FinishEvent OrderCode	PackageCode	3
Storage	QueryCarriersAndSt.	List of Item Types	MaterialStock	8
	QueryStorage	List of Item Types	MaterialStock	4
Wrapper	GiftWrapper	Type of Wrapper Package Dimensions	WrapEvent	6
	PackageWrapper	Package Dimensions	WrapEvent	4

Table 1. Available services in the *PackingCell* system

In order to illustrate how the SAES works a trace of this *packing cell* system is presented. First of all, the registered services are translated into PDDL durative actions by the *Service Translator*. Once the services are modeled as actions, the *Service Translator* generates a PDDL domain file which contains the actions, structured as a planning problem. In this example, the SAES client is the manufacturing manager which controls the cell configuration. When a new configuration is needed, due to a new product demand, the manager sends a request to the SAES. The request contains the I/O's that the service should have. In this example a possible service request could be described with: (i) inputs: *ShuttleEvent* and *OrderEvent* and (ii) output: *PackageCode*. Furthermore, the manager could establish a temporal restriction, requiring the service to be provided in 30 time units.

6.2 Packing Cell System Trace.

Considering the previous situation, the sequence of the next steps are described. First, the manager request is translated by the SC into a PDDL problem description. At that point, the domain and problem descriptions are available. This information would be sent to the *Composer Service*. Then, the *Composer Service* starts to find service compositions (plans) that fulfill the client request and time constraints.

The first plan found is the plan presented in Figure 6. The solution is a possible configuration of six services to fulfill the client request. Each service is associated to its estimated execution time. The estimated execution time for the composition is 23 time units.

This solution would be sent to the CM. The *Composer Service* would continue searching for plans until no more possible plans exist or until the CM sends the SC a message to finish the search process. This situation occurs when a previous service composition has been accepted by the client.

Time:<**ACTION**, *PARAMETERS*> [action duration; action cost]
0.0003: (**LOCKSHUTTLESERVICE** *ARRIVAL ORDER LOCKSHUTTLEFLAG NOTIFICATIONEVENT*) [3]
3.0005: (**GETORDERSERVICE** *NOTIFICATIONEVENT ITEMTYPELIST ORDERCODE*) [4]
7.0008: (**QUERYCARRIERSANDSTORAGESERVICE** *ITEMTYPELIST MATERIALSTOCK*) [8]
15.0010: (**GETITEMSSERVICE** *MATERIALSTOCK ITEMTYPELIST NOTIFICATIONEVENT FINISHEVENT*) [8]
23.0012: (**SENDORDERSERVICE** *ORDERCODE FINISHEVENT PACKAGECODE)* [3]
23.0015: (**UNLOCKSHUTTLESERVICE** *LOCKSHUTTLE FLAG FINISHEVENT UNLOCKSHUTTLEFLAG*) [2]
Actions: 6 Execution cost: 6.00 Duration:23.000 Plan quality:23.000

Figure 6. Sequence of services of the first plan

After the composition is sent to the CM, the CM gets through to the providers that appear in the composition and consults them about: their execution time, the start time and the success probability. With this information the CM checks if the composition satisfies the client temporal constraint. One of the possible situations that could arise is that a service could be busy executing previous requests. To illustrate this situation, the service "*querycarrierandstorageservice*" is supposed to have a high workload as it is busy executing previous requests. Therefore, the service needs more time units to deal with the new request. So, the number of time units needed to execute the composition is greater than the assigned time to complete the client goal.

The CM calculates the total execution time taking into account the current service workload. This time is higher than the client temporal constraint so the service composition is dismissed. The CM communicates to SC that it needs another composition. The SC gives CM a second service composition solution that consists of six services and the total estimated execution time is 19 (Figure 7).

Time:<**ACTION**, *PARAMETERS*> [action duration; action cost]
0.0003: (**LOCKSHUTTLESERVICE** *ARRIVAL ORDER LOCKSHUTTLEFLAG NOTIFICATIONEVENT*) [3]
3.0005: (**GETORDERSERVICE** *NOTIFICATIONEVENT ITEMTYPELIST ORDERCODE*) [4]
7.0008: (**QUERYSTORAGESERVICE** *ITEMTYPELIST MATERIALSTOCK)*[4]
11.0010: (**GETITEMSOPSERVICE** *NOTIFICATIONEVENT MATERIALSTOCK ITEMTYPELIST FINISHEVENT*) [5]
16.0012:(**SENDORDERSERVICE** *ORDERCODE FINISHEVENT PACKAGECODE*)[3]
16.0015: (**UNLOCKSHUTTLESERVICE** *LOCKSHUTTLEFLAG FINISHEVENT UNLOCKSHUTTLEFLAG*) [2]
Actions: 6 Execution cost: 6.00 Duration:19.000 Plan quality:19.000

Figure 7. Sequence of services of the second plan

The CM analyzes this new service composition. Supposing that the service "*querystorageservice*" now has a lower workload and needs 6 time units to complete his task, the fulfillment of the goal can be accomplished on time. Then, the CM establishes the pre-commitments with the providers and analyzes the success probability, as discussed in previous sections.

Finally, the CM consults the manager if it agrees with the success probability. In that case, the CM formalizes the commitments with providers and monitors service executions.

6.3 Test and Results.

Several simulation experiments have been carried out to monitor the behavior of system and evaluate the incorporation of the CM into the SAES framework in order to provide the client with feasible plans. The experiments basically consist of launching a set of client requests to the SAES, including the CM or not.

The points to be analyzed from the results obtained after executing the tests are: (i) the number of client requests that can be attended with a positive answer (service composition) (ii) once the SAES provides a suitable service composition, to check if the composition provided has been carried out successfully.

In Figure 8 the behavior of the SAES, including the CM or not, is compared. On this graph it can be seen that SAES with the CM provides a lower number of answers (service compositions) than the SAES without the CM. The reason is that the SAES with CM rejects some of the compositions provided by the *Composer Service* because this configuration takes into account not only the composition suitability but also the service provider availability.

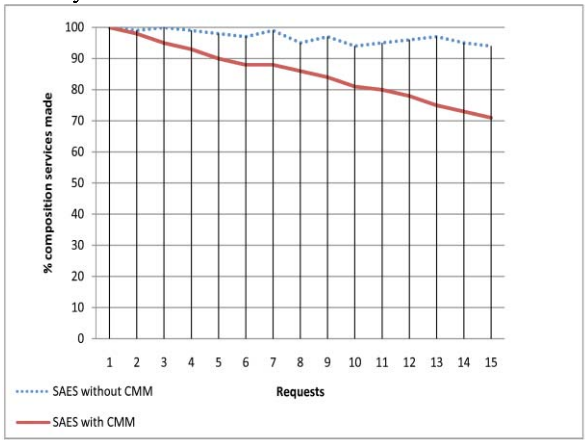

Figure 8. Number of service compositions provided by SAES including the CM or not.

In Figure 9 the percentage of successfully accepted plans, in both SAES configurations (with CM and without it), is shown. This graph reflects that once the client has accepted a service composition provided by SAES with

CM, the probability of service composition success is higher than the SAES without the CM. Obviously, this is because the CM first checks the providers' current availability.

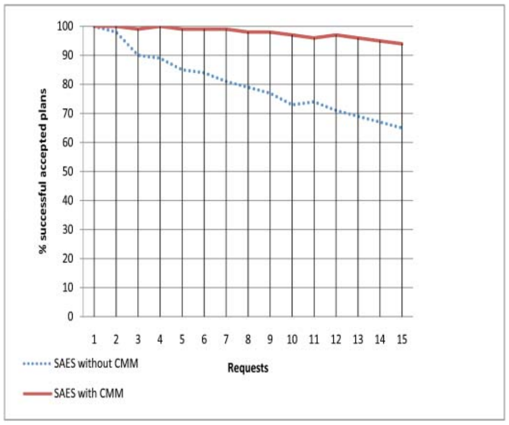

Figure 9. Execution success of service compositions provided by SAES including the CM or not.

7. Conclusions

Time in service execution is an important parameter to consider in service composition, mainly in environments where there are temporal constraints, such as manufacturing systems. Execution time in many situations is established by the providers in normal conditions.

This time estimation may not be realistic as it does not consider workload or service availability at the moment when a client request arrives. In this situation, taking a non-functional time parameter in service descriptions in to account is only useful for providing an initial service compositions. If a service composition to be reliable this information should be updated, bearing in mind the current services conditions. It is necessary to contact service providers and to query their availability and workload at that moment. With this information, the service compositions obtained are more accurate and their probability of success is higher, so the quality improves. In this paper a SAES (Search And Execution Services) framework has been presented to deal with service compositions and to ensure their fulfillment on time. The different modules that compound SAES and their functionality have been described in detail. Finally, this work has been tested and evaluated by using a simulated manufacturing scenario. The results obtained reflect the benefits of the SAES framework in different configurations.

8. References

[1] Erl, T.: SOA: Principles of Service Design. (2007)
[2] Fraser, R., Rankine, T., Woodcock, R.: Service oriented grid architecture for geosciences community. In: ACSW '07: Proceedings of the fifth Australasian symposium on ACSW frontiers, Darlinghurst, Australia, Australia, Australian Computer Society, Inc. (2007) 19—23
[3] Hayes, B.: Cloud computing. Commun. ACM 51(7) (2008)
[4] Consortium, W.W.W.: Owl-s: Semantic markup for web services. *http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/*
[5] Consortium, W.W.W.: Web service modeling ontology (wsmo). *http://www.w3.org/Submission/WSMO*
[6] Consortium, W.W.W.: Web service semantics - wsdl-s. *http://www.w3.org/Submission/WSDL-S*
[7] WSDL Working~Group, S.A.: Semantic annotations for wsdl and xml schema. http://www.w3.org/TR/sawsdl
[8] Naseri, M., Towhidi, A. :Qos-aware automatic composition of web services using ai planners. In: ICIW '07, (2007) ~29
[9] Solanki, M.: Tesco-s: A framework for defining temporal semantics in owl enabled services. In: W3C Workshop on Frameworks for Semantics in Web Services. (2005)
[10] Solanki, M., Cau, A., Zedan, H.: Augmenting semantic web service description with compositional specification. In: Proceedings of the 13th international World Wide Web conference (WWW 2004). (2004) 544--52
[11] Benatallah, B., Hacid, M.S., Rey, C., Toumani, F.: Request rewriting-based web service discovery. In: International Semantic Web Conference. (2003) 242--257
[12] Hashemian, S., Mavaddat, F.: A graph-based approach to web services composition. In IEEE Computer Society (2005)
[13] Gao, C., Liu, R., Song, Y., Chen, H.: A model checking tool embedded into services composition environment. In: Proceedings of the Fifth International Conference on Grid and Cooperative Computing, IEEE Computer Society (2006)
[14] Walton, C.: Model checking multi-agent web services. In: Proceedings of the 2004 Spring Symposium on Semantic Web Services, Stanford, CA, USA. (2004)
[15] Vukovic, M., Robinson, P.: Adaptive, planning based, web service composition for context awareness. In: Advances in Pervasive Computing. (2004) 247--252
[16] Carman, M., Serafini, L., PaoloTraverso: Web service composition as planning. In: CAPS'03
[17] Rao, J., Su, X.: A survey of automated web service composition methods (2005)
[18] Vukovic, M., Vukovic, C.M.: Context aware service composition. Technical report (2006)
[19] Oh, S.C., Lee, D., Kumara, S.R.T.: A comparative illustration of ai planning-based web services composition. SIGecom Exch. 5(5) (2006) 1--10
[20] Fox, M., Long, D.: Pddl2.1: An extension to pddl for expressing temporal planning domains. (JAIR) 20 (2003)
[21] Klusch, M., Gerber, A. :Semantic web service composition planning with owls-xplan. In Proceedings of the 1st Int. AAAI Fall Symposium on Agents and the Semantic Web. (2005)
[22] Liu, C.L., Layland, J.W.: scheduling algorithms for multiprogramming in a hard-real-time environment. ACM 20(1) (1973) 46--61
[23] Burns, A., Wellings, A.: Advanced fixed priority scheduling. J. Mathai (Ed.), Real-Time Systems (1996) 32--65
[24] Aamodt, A., Plaza, E.: Case-based reasoning; foundational issues, methodological variations, and system approaches. AI Comm.7(1) (1994) 39—59
[25] Marik, V., Vrba, P., Fletcher, M.: Agent-based simulation: Mast case study. Emerging Solutions for Future Manufacturing Systems (159) (2005) 61--72