A Bounded-Time Service Composition Algorithm for Distributed Real-Time Systems

M. García-Valls, R. Castro-Fernández, I. Estévez-Ayres, P. Basanta-Val, I. Rodríguez-López

Distributed Real-Time Systems Laboratory
Department of Telematics Engineering
Universidad Carlos III de Madrid
Leganés (Madrid), Spain
{mvalls, rcfernan, ayres, pbasanta, irlopez}@it.uc3m.es

Abstract—Handling the dynamics of future service-based distributed systems in real-time is a complex problem; a number of state transitions or reconfigurations take place that must be handled in real-time; this requires to impose some bounds to the structure of the system to ensure timely operation. We present a model for real-time reconfiguration based on a service model using the concept of service implementations that are actual versions of a specific functionality or service. Over this model, we present an algorithm for service composition that provides a feasible solution compliant with the provided application quality of service (QoS) criteria that consists of a set of service implementations. This algorithm executes in linear time by drawing a clear separation between the composition algorithm and the real-time analysis of the service implementation paths; this is a key idea that allows to have the linear time service-based composition algorithm as a simple straight forward graph search guided by some values or heuristics related to the application QoS. Our solution targets real-time systems being, therefore, appropriate for timely reconfiguration. The proposed solution is evaluated using an profiling tool. We provide experimental results of this tool showing the suitability of the approach and the proposed concepts.

Keywords- real-time, service composition, middleware, distributed systems, reconfiguration, SOA.

I. INTRODUCTION

Service Oriented Architectures (SOA) are today an active research field with significant progress in web-service (WS) based technologies. Distributed systems based on services increase the decoupling of components and the flexibility of the system as a whole in a way that new components and functionality pieces are easy to integrate compared to the monolithic deployments.

In such a scenario, services are self contained functionality units that interoperate with other services by exchanging messages. They can be composed to construct enhanced functionality units called applications. In such a context, applications can suffer transformations by simply replacing one service by a different one or by a different implementation of the same service. In modern systems, this is a powerful characteristic that enhances the flexibility of systems at the cost of introducing dynamic behavior. We name these transformations as reconfigurations, i.e., the process by which a system that is in a specific state evolves to a target state.

Service-based composition mechanisms have been studied mainly in the context of web-services that are typically silent about timeliness.

We target distributed real-time systems, and, in such a context, it is necessary to provide mechanisms for time-deterministic execution of the applications not only during normal operation but also during the reconfiguration process.

In this paper, we present an approach for providing time-bounded reconfiguration of distributed service-based applications, that has been carried out in the iLAND project [1]. In iLAND, we define applications as graph-based structures where nodes are the services, and the links are the messages exchanged between them. Each service may have different service implementations, that are different versions of the same service providing the same functionality but with different output qualities and, as a consequence, different resource demands. A running application is a path of the graph of service implementations. As a consequence, reconfiguring an application consists of replacing one path by another.

One of the main problems is how to deal with complex graphs when real-time reconfiguration is needed. Searching through the complete space of solutions can be unmanageable specially at run-time. Even with medium complexity graphs, it is desirable to reduce the space of solutions to search from in a graph. This issue is left outside of the scope of the work presented in this paper; here, we focus on a different but also important problem of providing a composition algorithm that outputs a result in bounded time. For this purpose, it is also required to provide the reconfiguration framework for the distributed real-time systems.

In this paper, we present a real-time reconfiguration framework by working at two levels. Initially, the structure of the reconfiguration process is logically broken down into
several phases that need be executed also in bounded-time. After, we provide a solution to deal with the timely composition of services through a linear-time algorithm; this is one of the most important phases in real-time reconfiguration. Our approach is validated showing the execution in a profiling tool developed in Java; we show the behavior of our real-time algorithm, and we demonstrate that the composition time can be estimated due to the linear time behavior of the process.

This paper is organized as follows. We present the related work in section 2. Section 3 describes the main characteristics of the reconfiguration in distributed real-time systems. Section 4 presents our approach to achieve a bounded time service-based composition algorithm. Section 5 presents the validation of our approach. Section 6 concludes the paper and presents future work.

II. RELATED WORK

Service-based environments as described in iLAND [1] come from the traditional structuring of applications into functionality pieces with well-defined interfaces. Research on service-based paradigms has traditionally been applied to web-services [2][3], that is only a special part of service-oriented computing. Web environments are typically not sensitive to real-time deadlines, but they are being used in projects that target industrial systems such as SOCRADES [4] where web services were applied to manufacturing and SIRENA and OASIS [5] where DPWS (Device Profile Web Services) middleware enables the interaction with the embedded nodes present at the factory floor. In this context, timeliness is not handled from a real-time scheduling perspective but from efficient low-level programming that yields acceptable performance results for embedded domains. Other work as [6] merged web services solutions for production environments.

From a real-time perspective, some projects applied resource management in the context of distributed systems using SOA for interoperability among nodes. This is the case of [7], being limited about the schedulability of the complete system not considering the network effects. It does not aim at reconfiguration but rather it uses SOA for interoperability among nodes where a certain level of resource management and enforcement takes place. Also, project RTLlama [8] proposed a similar approach based on resource reservations and estimations of the response time of services; a centralized algorithm for service composition is used based on the calculation of the shortest path.

Other approaches as [9] enhance the SOA model to provide a composition algorithm with the goal of supporting real-time applications. This approach does not consider the network effects at it uses a light real-time characterization which is not realistic in the domain.

Other real-time SOA efforts as [10] provide data transformation services that must be time-bounded. Their assumptions are the a-priori knowledge of the execution times of services, but the schedulability of the complete system is not considered, this contradicts the real-time guarantees.

Therefore, reconfiguration is not a new concern. It has been studied in different domains but rather limited to centralized systems as [11][12][13] (in which a set of time-bounded steps were defined); or in the distributed systems domain, they were silent about timeliness as in [14][15]. Other domains that are not silent about timing issues as [16] do not fully aim at obtaining results in real-time, but rather at providing analysis of precision and accuracy of service composition in soft real-time environments; others strictly focus at characterizing the QoS properties as [17]. To summarize, we can argue that the majority of the analyzed related work proposes interesting solutions to web service composition for the general computing environments. However, they lack the adequate modeling of the real-time properties of services, the considerations of the schedulability of the system as a whole, the timely reconfiguration solutions, and the timely execution and communication guarantees for services.

In this paper, we present an approach that identifies the service composition as an essential building block of the reconfiguration of a system by presenting a Bounded Time Service-Composition Algorithm for timely selection of service sets; these service sets are aggregated in a subsequent step that is out of the scope of this paper to conform the full timely reconfiguration. Previously presented composition algorithms are based on uninformed information search techniques, and they use information that is internally calculated to optimize the search process. The problem is that internal calculations can become complex, and they can easily challenge the timeliness of the response. Our composition algorithm requires additional information based on external calculations that can be done in parallel; this is compatible with our system model where the real-time profiling of services must be done off-line and prior to execution.

III. DISTRIBUTED REAL-TIME RECONFIGURATION

A. Real-time service model and related components

Timely reconfiguration in distributed real-time systems is not a trivial task. For open distributed systems with lack of restrictions it is, in fact, not solvable with the available computing paradigms, i.e., a set of bounds and limitations to the structure of systems need to be imposed. This is a way of reducing the complexity of the problem in order to obtain a set of phases for the complete reconfiguration process that need to be bounded. We restrict the magnitude of the problem in order to achieve a bounded time solution for reconfiguration, that implies a modification in at least one service implementation.

In our work, a reconfiguration is the process of transitioning from the current state of the system to the target state. A system state is a set of service implementations \( S_t \) that are active and running.
Therefore, in a reconfiguration at least one service implementation needs to be either stopped, replaced by another one, or launched. A service implementation $S_i$ is specified by its functionality, its timing parameters, and the list of dependencies in the following way: \( \{F, C, T, D, P, Q, \Delta\} \) where $F$ is the functionality, $C$ is the computation time or processor cycles it requires to complete its function, $T$ is the release period since in our real-time computation model (see [18]) all tasks are approximated by periodic, $D$ is the deadline to complete its function, $P$ is the priority that indicates the relative importance of the service implementations, $Q$ is the output quality delivered by $S_i$, and $\Delta$ is the dependency list with respect to other service implementations.

The replacement of a service implementation then implies a readjustment in the resource assignment since different service implementations have different $C_i$ and $T_i$ values that determine their processor requirements. Reconfigurations will then only be allowed if the system has enough free resources for all service implementations of the target system state. As a consequence, prior to initiating the transition from the current state to the target state, an admission test will be executed to determine if there are enough spare resources to execute the service implementations of the target system state.

Reconfiguration triggers are of two main types. The system performs a continuous monitoring activity to detect whether some service implementation is consuming more than its assigned budget, some application or service implementation is not meeting deadlines, or the system is at risk of overload or failure. When these situations are detected, an internal reconfiguration event is triggered. On the other hand, functional reconfiguration events are triggered generally by users/operators or by some programmed/timed event at application level; they initiate the replacement of modification of the current functionality of the system (e.g., change the resolution of some incoming video signal or store to disk the incoming video instead of displaying it) that requires that different service implementations be activated.

At architectural level, we have designed the components shown in figure 1, implemented inside the iLAND middleware.

CFL manages the structure of applications, contains the service composition logic (CL component), and the logic for the overall coordination of the system reconfiguration (CM). Reconfiguration triggers are handled by the CM component that initiates and controls the reconfiguration process invoking the required operations from the CLC to obtain the new application graphs, and from the underlying layer that performs the admission control.

CBL performs the lower-level communication and resource management functionality. Inside this layer, the component that is related to the reconfiguration is the QoSRM (Quality of Service Resource Manager) that is in charge of monitoring the resource consumption of the service implementations in order to enforce their contracted resource assignments and it also executes the admission logic based on a real-time schedulability test.

**B. Service-based reconfiguration through graphs**

Figure 2 presents the overview of the reconfiguration protocol. It has four phases based on the evolution of the service and service implementation graphs until a path is selected and executed in the system.

Achieving real-time reconfiguration in a complete open system is an NP-hard problem at the present state of science and technology. However, we have enabled it by imposing a set of limitations and bounds. We define two main phases in the execution of the system:

- **Initialization phase**: we perform an a priori study of the system and its service graph and derived service implementation graphs. We analyze the schedulability of the graphs, their complexity, and perform a fine tuning of the system in order to bound the sources of unpredictability. This phase also calculates the reconfiguration time that is application dependent.

- **Mission phase**: this is the phase when the system is in execution. Once the graphs have been analyzed and the size of the system is assured to be of medium-low
complexity, the reconfiguration then will take place in bounded time.

We define the real-time reconfiguration protocol having four phases:

1. **Application graph elaboration.** When a reconfiguration event is triggered, the Control Manager is notified of it (by the application or by the QoSRM). Then, it takes the current execution graph (EG) that contains the running service implementations of the current state and it determines it the target state will share the same services. Then, it creates the new application graph (AG) that contains the set of services of the target state and builds the Expanded Graph (XG) that only contains service implementations; for each service of the AG, the XG replaces it by its service implementations that can run in the target state.

2. **Expanded graph pruning.** To decrease the complexity of the XG and to increase efficiency, the non-schedulable paths of the XG are removed from it. The CM sends the XG paths to the QoSRM that executes the schedulability test to determine which paths to remove. As result of this phase, the Scheduled Expanded Graph (SXG) is obtained that contains the paths that are schedulable.

3. **Execution of the composition logic.** The composition algorithm's goal is to decide the application to be run (i.e., the target EG) according to a QoS parameter value. For example, in a video application, the QoS value can be the end-to-end time that determines the frequency of frames displayed on screen. The composition logic then receives as input two things: the system QoS parameter value and the SXG from which to select the target EG. It then explores the space of possibilities of the SXG to select one single path that is closest to the QoS parameter.

4. **Execute the new application.** The new application is the target EG. This phase implies replacing old service implementation by new ones and/or changing some execution parameters. The CM coordinates the mode change; there are different possibilities in the literature for this depending on the nature of the applications [12]. In our case, we use immediate mode change (i.e., instant replacement of the old service implementations and instant start up of new ones).

**C. Real-time reconfiguration scheme**

The time to reconfigure is application dependent, i.e., graph dependent since it may be required to consider important aspects as the nature of the processed data and sensitivity to timing aspects, among others. To alleviate this problem, we have worked at the levels of:

- Identifying the sub-steps of the reconfiguration and making them time-bounded,
- Reducing the complexity of graphs to limit the complexity and time taken to search through them, and
- Designing a bounded-time service composition based on graph search techniques.

Following, we present the breakdown of the reconfiguration protocol into the steps that ought to be followed to enable a state transition. Firstly, we define the reconfiguration time slot $b^*$ according the terminology defined in [18]. In this slot, the phases shown in figure 3 must take place in bounded time.

![Figure 3. Reconfiguration phases and timing](image)

Following, we describe how all these steps have been bounded in time with some limitations to the system model. The control manager, CM, is in charge of controlling these steps:

- **Trigger and initial processing ($t^\pi$).** The detection of the reconfiguration event is performed in the first place, and it is initially processed to determine if a functional or an internal transition must take place.
- **Graph working time ($t^\delta$).** The current application graph is obtained and, in case of a functional reconfiguration, it is updated with the new services. Services in the AG are expanded with their service implementations, and the XG is obtained.
- **Admission control time ($t^\psi$).** The XG contains the possibilities of the new state of the system. The CM interacts with the QoSRM for this matter. The CM extracts individual paths from the CM and sends them to the QoSRM that determines whether the path is schedulable or not (i.e., whether it passes the real-time schedulability test).
- **Mode change time ($t^\omega$).** After the new EG has been selected, the service implementations it contains must be executed. Also, any old service implementation must be stopped. This is in practice a mode change. There are a number of time-bounded mode change algorithms in the literature (e.g. [12]) that determine the sequence of steps to change to the new state (e.g. first stop old service implementations and then start new ones or do the change progressively). The type of mode change depends on the applications and the nature of the data they process. Some
applications can afford to lose some data for the benefit of having a fast transition; others do not.

The most difficult step to bound in time is the graph transformation time or real-time pruning ($t^p$). There may be different ways to extract a representative sub-graph from a graph in order to reduce its complexity depending on the reduction percentage. In real-time, we are limited by the time slot assigned to this process ($t^p$). If we define $t^m$ as the interaction time between CM and QoSRM in such a way that $t^m = t^s - t^p - t^o - t^c - t^t$.

The remaining time, $t^p$, is assigned to the real-time pruning; however, by limiting this time slot, it is possible that some solutions will be lost in the graph summary, but the process can be time-bounded.

The next step consists of choosing a suitable criterion in the real-time pruning that increases the chances of finding all the potential solutions within the $t^p$ time slot, hence respecting as much as possible the bounded-error capability. In the next section it is explained some procedures to minimize the fact of limiting the computation time for the extraction of the Scheduled Expanded Graph.

IV. SERVICE COMPOSITION IN THE RECONFIGURATION PROCESS

Guaranteeing that the reconfiguration steps will be achieved in bounded time requires to analyze the interactions among components (done in the previous sections) and study the type of operations performed by each step. In the reconfiguration, one of the most important steps is the service composition since timeliness of the reconfiguration is achieved through time-bounded service composition. In this section, we describe the latter to prove the time deterministic nature of the system dynamic reconfiguration.

1) Reconfiguration Event Detection and Initial Processing.

When a reconfiguration event arrives, the first operation to perform is to search for the application that should be affected; its identifier is obtained and the AG is retrieved. These are two read operations and their timing is easy to obtain and fit in the $t^s$ time.

2) AG Expansion.

This must be done in the graph working time slot consisting of expanding the AG into the XG (time-bounded by the a priori study in the initialization phase). In the XG, an initial prune is made by removing the service implementations that are not eligible to run, i.e., they can be in use by other running applications or they can be down. The three initial phases, shown in figure 4, are efficient in execution time since only a few operations are done within the slot defined by $t^o$, and these operations are time-bounded.

![Figure 4. Time-bounded reconfiguration for steps 1 through 3](image)

3) Graph Pruning Logic ($t^p$).

In this step, the XG is transformed into a smaller graph, SXG, containing only some selected solutions. Each combination of the SXG is sent to the QoSRM that checks its schedulability. The internal structure of this logic is one of the most important steps of the reconfiguration logic, and its efficient design is mandatory to achieve timelines. This logic has been later achieved in the iLAND project, but its explanation is currently outside the scope of this paper.

4) Service Composition Process.

The composition logic is sketched in figure 5.

![Figure 5. Bounded-time composition algorithm in the context of iLAND middleware](image)

We have defined the Bounded Composition Algorithm that searches through the SXG to select the EG to be run; it executes in linear time with respect to the size of the graph, and it selects the path that is closest to the specified application QoS criteria. In our approach, the application QoS criteria is the target/desired application end-to-end time.

The algorithm relies on the usage of heuristics, i.e., specific information items or values related to a given composition criterion. Different heuristics can be processed by the logic to maximize or minimize different values of the system. The ones used in our approach aim at:

- Fulfilling end-to-end response time of applications.
- Reducing the node utilization factor (incur in low C/T values for the service implementations).

5) Mode Change.

Finally, the service implementations of the Execution Graph are started issuing system calls for the creation of the threads in the nodes where the service implementations are located. An immediate mode change algorithm is used since the time it takes is linear with respect to the number of service implementations concerned in the state transition.
V. BOUNDED-TIME SERVICE COMPOSITION ALGORITHM

In essence, the proposed time-deterministic service composition algorithm strictly separates the real-time schedulability analysis technique from the graph-search method; the schedulability analysis is performed in the previous step transforming the XG into the SXG.

The scheduling technique requires then the usage of certain calculated data values of the graph that we call *heuristics*. We identify two main approaches for working with heuristics in a service composition algorithm:

- **External heuristics** are the additional information that is calculated outside of the composition logic. This information is fed to the algorithm as data values.
- **Internal heuristics** are information required that is computed inside the composition logic. Since they are calculated as additional tasks to be done inside the composition algorithm, they have an effect over the complexity of the algorithm.

Performance and complexity are mostly the deciding variables to determine whether heuristics are calculated inside or outside the composition logic. Also, it should be considered that external heuristics present some advantages as the following:

- Complex heuristics can be worked on in parallel to improve their calculation time.
- It is a more flexible technique since it is simpler to update the internal algorithmic calculations of the composition logic.
- Recalculation of heuristics is avoided; using internal heuristics requires that the computation of heuristics is recalculated every time that the composition logic is invoked.

The following assumptions are made with respect to the application graphs: no cycles are contained, they are forward-only directed graphs, and the output quality delivered by a service is directly proportional to the amount of resources (processor time) it consumes.

Let us consider that application $a^i$ has services $\{s_{i1}, \ldots, s_{in}\}$ and each service $s_{j}$ has one or more service implementations in the form $\{s'_{j1}, \ldots, s'_{jm}\}$. The set of paths of the AG of application $a^i$ are named as $\pi^i$. Each path contains a set of service implementations, i.e., one service implementation per service; therefore, $\pi^i_j$ is the $j^{th}$ path of the application $a^i$. The following heuristics are calculated within the graph:

- **Individual service heuristics**: each service has a value $H_j$, resulting from the calculation of the average QoS value of all its service implementations in the following form: $H_j = \text{avg}(s'_{j1}, \ldots, s'_{jm})$.
- **Graph heuristics**: they are calculated for the whole graph: $\pi^m$ is the path of service implementations that have the minimum QoS value; $\pi^M$ is the path of service implementations with the maximum QoS value; $\pi^i$ is the path of service implementations with the average QoS value of $\pi^M$ and $\pi^m$.

Initially, the desired end-to-end time for the application is checked against the value of $\pi^M$ and $\pi^m$. If the end-to-end deadline is higher than $\pi^M$ then the optimal solution is the maximum path; if the end-to-end deadline is smaller than the $\pi^m$, there is no feasible solution. If the deadline is larger than the minimum path, there is a solution that is contained within the graph.

We establish four *regions* in a graph as presented in figure 6 by adding two extra control values that will guide the search, $B^M$ and $B^m$. The number of regions of the algorithm influences the level of accuracy of the selected final solution.

![Figure 6. Graph heuristics and related regions](image)

$B^M$ and $B^m$ are the intermediate control points between the minimum and maximum path values, respectively.

$$B^M = (\pi^m + \pi^a) / 2$$
$$B^m = (\pi^a + \pi^M) / 2$$

(1)

Each value computed by the search algorithm can be compared to any of the five control points to know the current distance from the desired end-to-end QoS value.

The desired end-to-end deadline is assigned to the appropriate region that is defined by the two values that are calculated in order to create the bounds for the algorithm to search from. The upper bound will be the application QoS value (e.g., deadline) $Q$; the lower bound will be $A$.

We define a *quality margin* $QM$ to check if a service implementation is inside $Q$, i.e, inside the application QoS limits as

$$QM = LC^p - LB^p$$

(2)

where

$$LM^p = \frac{\chi_{\text{lim}}}{\pi^a}$$
$$LB^p = A / \pi^a$$

(3)

$LCP$ gives an optimal theoretical path, so the search process must try to find a path which is as similar as possible to it. Therefore, $LCP$ gives a theoretical path to search within the graph. This theoretical path is variable and adjusted based on the best possible quality, which is that one given by the $\chi_{\text{lim}}$ or the limiting criterion. Hence, it is
easy to see that the path given by the algorithm must be worse or equal (in the best case) than that of the theoretical path.

Let $LLV$ be the lower labeled value in the regions above, and $C_A$ be the set of solutions that the algorithm can obtain. Then $\Lambda$ is defined as:

$$A = LLV \mid A < C_A < \chi^\text{lim}$$  \hfill (4)

A linear search over the graph is done that uses the above calculations. According to the previous region values, the algorithm explores every service implementation deciding whether the service implementation will be part of the final selected path. The algorithm never goes back, so, the complexity of our service composition algorithm is linear with respect to the size of the graph.

For each service, the search process takes all the service implementations and sorts them by QoS. Then, the algorithm calculates the $IB^p$, which is the corresponding $LC^p$. $LC^p$ give a general guiding value at graph level, and $IB^p$ gives a guiding value at each level.

With the reference value calculated ($IB^p$), the algorithm searches for the service implementation that is closest to it and pre-selects it. Taking the decision on whether the pre-selected service implementation is going to be the chosen requires that the previous service implementation does not exceed the maximum quality allowed according to the $QM$. The $QM$ is in charge of controlling if the QoS of the candidate service implementation is close enough to the selected value. If the $QM$ is lower than the candidate QoS, the service implementation is not valid, and we need another with worst QoS.

The search process is iterative, and ends when the algorithm has explored all the services in the graph.

VI. VALIDATION

We have validated our approach by implementing the algorithms that reside inside the iLAND middleware [1][19][20]. We have used a profiling tool developed in Java SDK 1.6 that allows to study, analyze, and validate the proposed algorithm in a platform independent environment. Applications are modeled following our service model as described in our previous work [17] and [1] which is an evolution to the service model of distributed interactions and component frameworks as [23], [11], and [12]. The bounded time composition algorithm has been studied with respect to its cost (execution time to deliver a decision, e.g., an EG) and the quality of the delivered result with respect to the theoretical optimum. Also, iLAND middleware containing this algorithm has been validated in a distributed real-time video streaming application. We have used an Ubuntu 9.10 Linux distribution running on a hardware based on Intel processor with 2 cores of 2.2 GHz and 4 GB memory. The implementation has been done in C and a GCC 4.4.1 compiler.

Figure 7 shows the behavior of the algorithm. It can be seen that it executes in less than linear time even in the worst case.

Most of current composition time algorithms based on graph search may present exponential execution time. It is then possible to estimate the execution time of the algorithm at run-time since the function shape is known.

Figure 7. Execution time of the service composition algorithm

Figure 8 shows the quality test for assessing the results output by our composition approach. The average expected quality of the algorithm is measured with respect to the specified application QoS value entered. We take as optimum result the path that has an end-to-end value closest to the application QoS value.

Figure 8. Quality analysis results

The quality measure used for this experiment is given by the following expression: \( Q/t \), where $Q$ is the desired application QoS and $t$ is the QoS of the path selected by the algorithm. In the best case, the optimum solution will be reached and then the expression will be 1. In the worst case, the difference between the optimum and the algorithm result is the largest possible. In the proposed solution, it can be seen that the worst quality is close to 70%. We show the mean quality values for 8, 16, 32, 64,128 and 256 nodes. It is worth noting that values range from 86.5% to 90%; so,
the number of nodes does not affect the quality offered by the algorithm.

VII. CONCLUSIONS

Handling the dynamics next generation distributed real-time systems is a complex problem that requires to impose some bounds to the structure of the system to really achieve timely response. Systems will be distributed made of different pieces of functionality that will composed on-line to adapt to the changing world. Timely composition of them will be required in order to build responsive systems. In this paper, we have described a bounded-time composition algorithm that is part of a real-time reconfiguration scheme that identifies the different phases to be performed in a transition. By leaving out of the composition the real-time calculations, the service composition algorithm is kept very simple just as a straight forward graph search guided by some reasonable margins and heuristics that are pre-computed. The outcome of the algorithm is the system execution graph that is the path closest to the optimum or the application QoS that is specified. We have shown the results obtained from its implementation that validate the presented algorithm, showing its suitability of the approach and the proposed concepts.

ACKNOWLEDGEMENT

This work has been partly supported by the iLAND project (ARTEMIS-JU 100026) funded by the ARTEMIS JTU Call 1 and the Spanish Ministry of Industry, Commerce, and Tourism (www.iland-artemis.org), ARTISTDesign NoE (IST-2007-214373) of the EU 7th Framework Programme, and by the Spanish national project REM4VSS (TIN 2011-28339).

REFERENCES