A real-time perspective of service composition: key concepts and some contributions

M. García Valls, P. Basanta Val

PII: S1383-7621(13)00123-9
DOI: http://dx.doi.org/10.1016/j.sysarc.2013.06.008
Reference: SYSARC 1148

To appear in: Journal of Systems Architecture

Please cite this article as: M. García Valls, P. Basanta Val, A real-time perspective of service composition: key concepts and some contributions, Journal of Systems Architecture (2013), doi: http://dx.doi.org/10.1016/j.sysarc.2013.06.008

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.
A real-time perspective of service composition: key concepts and some contributions

Authors:

M. García Valls, P. Basanta Val
Distributed Real-Time Systems Lab.
Dept. Ingeniería Telemática
Universidad Carlos III de Madrid
Avda de la Universidad 30, 28911 Leganés, Madrid, Spain
Phone number (+34) 91 6248783
e-mail: mvalls@it.uc3m.es

Last review:
Monday, 15 July 2013
A real-time perspective of service composition: key concepts and some contributions

Abstract.- Timing predictability of service oriented architectures is challenged by their dynamic nature. Systems have to reconfigure their service-based structure to adapt to the changing environmental requirements. The development of dynamic systems that have timing constraints is currently not possible without imposing some bounds and limitations to the structure and operation of the system. This paper identifies the key factors for achieving time-bounded service-based reconfiguration from a system perspective. The key contribution is to bypass the possible complexity of the used task model and associated schedulability analysis algorithm is the provided architectural design that separates the composition from the schedulability. The paper also extends a previous service composition algorithm that provided a feasible solution compliant with the application quality of service criteria (QoS). Due to the design based on the separation of concerns, the algorithm is a simple straight forward graph search guided by values related to the application QoS. The generalized algorithm offers a search mechanism guided through $n$ regions going beyond the four regions model of the previous contribution. Results from the previous algorithm and the $n$ regions version are shown to illustrate the advantages and finer grain results of the latter.

Keywords: service composition, real time systems, reconfiguration, service oriented architecture, SOA, middleware, distributed systems.

1. Introduction

It is nowadays widely accepted that service oriented architectures (SOAs) are one of the powerful tools for constructing highly decoupled distributed systems. SOAs are an architectural style and a programming model based on the concept of services. The main principles of SOAs are loose coupling, abstraction, reusability, and composition. SOAs increase the flexibility of the system as a whole in a way that new components and functionality pieces can be integrated easily, especially if compared to the traditional monolithic deployments. 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 may undergo structural transformations by simply replacing one service by another or by a different implementation of the same service that adjusts to the same provided interfaces. For this reason, service composition algorithms are required to adequately select and connect services to construct different applications. Service composition is, then, a powerful tool to enhance the flexibility of systems. In this work, the composition algorithms are observed in the context of the overall process of transformation of applications. We refer to such transformations of the structure of applications as reconfigurations, i.e., the process by which a system is in a specific state evolves to a target state by some modification in the set of constituent services.

Although the advantages of supporting reconfigurations in real-time systems are significant, reconfigurations come at the cost of increasing the temporal unpredictability of systems. Service-based composition mechanisms have been studied mainly in the context of web-services that are typically silent about timeliness. In this work, we focus on 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 can have a number of service implementations, that are different versions of the same service offering the same functionality, i.e., adjusting to the same provided interface. Different service implementations output different qualities and, as a consequence, can have different computational resource requirements.

In iLAND, an application in execution is a path of the graph of service implementations. As a consequence, reconfiguring an application consists of replacing one path by another. Two of the phases of the overall reconfiguration process are the management and reduction of complex graphs and the real-time schedulability...
model. On the one hand, dealing with complex graphs when real-time reconfiguration is needed is a mandatory activity. Searching through the complete graph of possibilities can be unmanageable specially at run-time. Even with medium complexity graphs, it is desirable to reduce the space of solutions to search from. As a consequence, some strategy to reduce the complexity of the graphs must be part of the reconfiguration process. On the other hand, the temporal parameters of services must be carefully established and adequately adjusted to a suitable real-time model in order not to incur in an unmanageable determination of the execution feasibility of a service set. Therefore, the selected real-time schedulability model is of outmost importance in a real-time service model. Both phases, i.e., reduction of complex graphs and real-time schedulability model, are left outside of the scope of the work presented in this paper. Here, we focus on the service composition phase that outputs a result in bounded time to be suitable for real-time systems.

This paper also provides a description of the reconfiguration framework for the target distributed real-time systems that works at two levels. Initially, the structure of the reconfiguration process is logically broken down into several phases that need be all executed in bounded-time. Then, we describe a linear-time algorithm for service-based composition which is one of the real-time reconfiguration phases. Our approach is validated showing its simulated execution; 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.

The current paper extends the work presented in [2] based on the definition of five search regions for timely selection of service sets, and we provide a generalized algorithm for service composition based on \( n \) regions search. Moreover, this paper extends the previous work presented in [2] elaborating the key idea of separation of concerns between schedulability analysis and the service composition. This is a fundamental concept to achieve the time-bounded property of the composition algorithm. As a result, complex real-time task models can be used, and they will not directly affect the complexity of the composition algorithm. Another positive result from the separation of concerns is that we do not require to integrate calculations of temporal behavior inside the search process that can easily challenge the timeliness of the composition. Our composition requires relies on external calculations (off-line) that can be done in parallel to improve the efficiency and response time of the algorithm; this is compatible with our system model where the real-time profiling of services must be done off-line and prior to execution. Also, we improve the state of the art including recent related work and we provide an experimental comparison of our new \( n \) regions service composition algorithm by the previous algorithm presented in [2].

The paper is organized as follows. We present the related work in section 2. Section 3 describes the real-time model for achieving real-time reconfiguration. Section 4 elaborates the principle of separation of concerns to achieve time-bounded service composition; also, it describes the key phases of real-time service-based reconfiguration presenting key ideas for performing them in bounded time. Section 5 presents the generalized bounded-time composition algorithm based on \( n \) regions search. Section 6 presents the validation of our approach compared to the previous one showing that finer search is performed and cost is improved. Section 6 concludes the paper.

2. Background

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 [3][4][5], that is only a special part of service-oriented computing. For instance, Dynamic SOA (D-SOA) adds dynamic behaviour to SOA in the sense that it provides dynamic availability and dynamic properties modification. Dynamic availability [6] [7] refers to the ability of the services to be available or unavailable at any moment; whereas, dynamic properties modification designates the fact that service properties can be modified at run time. Dynamic availability allows systems to evolve without downtime and dynamic properties modification may be useful in dynamic context adaptation or negotiation. In both cases, the service consumer must be notified of the context changes.

Web environments are typically not sensitive to real-time deadlines, but they are being used in projects that target industrial systems such as SOCRADES [8] where web services were applied to manufacturing and SIRENA and OASIS [9] where DPWS (Device Profile Web Services) middleware enables the interaction with the embedded systems 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 [10] 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 across nodes such as [11] that is, nevertheless, limited about the schedulability of the complete system not considering the network effects. Also, it does not aim at reconfiguration, but it rather uses the SOA paradigm for interoperability across nodes where a certain level of resource management and enforcement takes place. The project RTLlama [12] 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 such as [13] 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 [14] 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 which contradicts the real-time guarantees. Other approaches rely on a previously specified composition strategy and they provide a real-time schedulability framework to support real-time replacement of functionality such as [15]; however, it is based on component technology.

Therefore, one may claim that reconfiguration is not a new concern in distributed systems that follow a service oriented paradigm. In fact, it has been studied in different domains but rather limited to centralized systems as described in the work of [15], [17], and [18] in which a set of time-bounded steps were defined. Resource management architectures with the ultimate goal of supporting real-time dynamic behavior in mobile operating systems such as [19] or in Java-based component frameworks such as [20] have also appeared. Other work focusing on the distributed systems domain such as [21]-[22] are silent about timeliness. Other efforts consider the temporal behavior of the services at run-time and other propose contributions to supporting failover rather for component-based systems [23]. Other proposed service composition algorithms such as [24] were first initiated on the context of assessment of software architectures of embedded systems and their real-time properties; however, they do not aim at obtaining resulting application graph in real-time, but rather at providing mechanisms for deciding improvements in the real-time properties based on the software architecture. Other approaches such as [25] strictly focus at characterizing the QoS properties of services in relation to the applications they belong to or for specific modeling languages as MARTE (see [26]).

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 further elaborate on the work presented in [2]. We focus on service-based composition as an essential building block or phase of the reconfiguration process of a SOA based distributed system. We take as basic reference the work of [2] that described a Bounded Time Service-Composition Algorithm based on the definition of five search regions for timely selection of service sets and we generalize proposing a finer grain solution based on n regions search. As part of the reconfiguration process, these service sets are later aggregated in a subsequent phase that is out of the scope of this paper to conform the full timely reconfiguration. Previously presented composition algorithms were based on uninformed information search techniques, and they used information that is internally calculated to optimize the search process. Their main problem is that internal calculations can become complex, and they can easily challenge the timeliness of the response. Our composition algorithm keeps the characteristic of requiring additional information based on external calculations that can be done in parallel to improve the efficiency and response time of the algorithm; this is compatible with our system model where the real-time profiling of services must be done off-line and prior to execution. In the current work, we formalize the real-time service model, and we establish a clear relation between the selected service implementations set and the quality of service of applications.

**3. Real-time reconfiguration in distributed systems**

**3.1 Real-time service model and required coordination 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 identify and define the 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. In our view, 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, so a reconfiguration occurs when a modification in, at least, one service implementation is required.

Therefore, in a reconfiguration at least one service implementation needs to be either stopped, replaced by another one, or launched. A service implementation $s_{(t)}$ is specified by its functionality, its temporal parameters, and the list of dependencies in the following way: \( \{ F, C, T, D, P, Q, \} \) 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 [28]) 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 \( s_{i,k} \), and \( \mathcal{S} \) is the dependency list with respect to other service implementations.

\[
\text{Application } a_i \text{ is } \{a_{i,j} : j=1,2,...\}
\]

\( a_i \) has a number of services represented by the set \( S_i = \{s_j : j=1,2,...\} \)

\text{Service } s_i \text{ has a number of implementations } \{s_{i,k} : k=1,2,...\}

Therefore, \( a_{i,j} \{ s_{i,k} : k=1,2,... \text{;} \ l=1,2,... \} \). It can also be represented by \( S_{i,j} \) that is a set of service implementations. Therefore, application \( a_{i,j} \) or service implementations sets \( S_{i,j} \) are used interchangeably.

In the end, \( a_i = \{ i_1, 2, ..., n \} \)

In a reconfiguration, current \( a_{i,j} \) is replaced by \( a_{i,k} \), such that \( a_{i,j} \prec a_{i,k} \) or \( S_{i,j}^{\text{init}} \leftrightarrow S_{i,k}^{\text{target}} \)

**Figure 1.** Formalization overview of the real-time reconfiguration model

The different internal structure possibilities of a given application \( a_i \) are \( a_{i,j} \). Reconfiguration refers to a change in the structure of the active software units that are part of an application. In general, a service can contain a number of threads or tasks; in our model, we consider single-task services. A reconfigurable system or application \( a_i \) is, therefore, a superset of \( n \) service implementations; in the end, \( a_i = \{ i_1, 2, ..., n \} \).

Different service implementations have different \( C_i \) and \( T_i \) values that determine their processor requirements. Then, the mere replacement of a service implementation implies a readjustment in the resource assignment. In a real-time system, the transition to a new system state is only allowed if the system has enough free resources for all service implementations of the target system state, \( S_{i,k}^{\text{target}} \). 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 available computational resources to execute the service implementations of \( S_{i,k}^{\text{target}} \).

State transitions are triggered for two main situations. On the one hand, the system performs a continuous monitoring of its actual resource consumption, i.e., it carries out and active tracing of the resource consumption of the service implementations. If it detects that some implementation is consuming more than its assigned budget, this can put at risk the deadline fulfillment of some other service implementation, i.e., the system is at risk of overload or failure. If such situation is detected, an internal reconfiguration event is triggered. On the other hand, functional changes can be issued or triggered by users/operators or by some programmed/timed event at application level. Functional changes consist of the replacement of modification of the current functionality of the system, e.g., a change the resolution of some incoming video signal or store to disk the incoming video instead of displaying it. Such changes require that different service implementations be executed and, therefore, resource demands may also vary.

At architectural level, we have designed iLAND middleware to include the necessary components to adjust to the behavior explained above. Figure 2 presents a summarized overview of the iLAND architecture that has two main layers: CFL or the Core Functionality Layer and CBL that is the Communication Backbone and Resource Manager Layer.

**Figure 2.** Components involved in the reconfiguration process

CFL manages the structure of applications, containing 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 orchestrating the invocation of the required operations from the CL component 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; it also executes the admission logic based on a real-time schedulability test.

Therefore, an overview of the orchestration of the invocations is shown in numbers in figure 2. Firstly, application structure of \( a_i \) is fed to the system through the CM component that stores all relevant information about
the application structure, services it contains, service implementations for each service, the real-time parameters of the implementations, and the quality values of applications. Secondly, CM invokes the schedulability analysis of the admission test that resides in the QoS Resource Manager. Thirdly, after the CM has a set of schedulable \( a_{ij} \), it invokes the execution of the service-based composition algorithm or CL. Lastly, the CL component will select a specific \( a_{ij} \) that will be the target system state \( a_{ij} = S_{t, target} \).

### 3.2 Service-based reconfiguration through graphs

The reconfiguration of an application requires some technique to store the current system structure and the space of solutions for selection of the target configuration. One of the most widely used techniques for storing and analyzing complex structures are graphs. Figure 3 presents the overview of the reconfiguration protocol that has four phases based on the evolution of graphs of applications containing services and service implementations. It shows the evolution until a path is selected as the target state to be executed in the system.

![Reconfiguration sequence through graph evolution](Image)

Figure 3. Reconfiguration sequence through graph evolution

Achieving real-time reconfiguration in a complete open system is an NP-hard problem at the present state of science and technology. Nevertheless, by imposing a set of limitations and bounds some practical limited solution can be achieved as shown in [28]. We define two main phases in the execution of the system. Firstly, we define an initialization phase where we perform an a priori study of the system by studying its service graph and mostly its service implementation graph. We analyze the schedulability of the graphs, their complexity, and perform fine tuning of the system in order to bound the sources of unpredictability. This phase may yield the calculation of the reconfiguration time that is application dependant. The second one is the mission phase that consists of the system in execution.

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

*Application graph elaboration.* When a reconfiguration event is triggered either by an application-programmed event or by the internal monitoring activity of the QoSRM, the CM component detects it and starts the reconfiguration process. First, the CM takes the current execution graph (EG) that contains the service implementations that are running in the present state, and it determines if the target state will share the same services. Then, it creates the new application graph (AG) that contains the set of services and implementations of the target state to build the Expanded Graph (XG) containing only service implementations. To build the XG, each node or service of the AG is replaced by the set of implementations that it can have in the target state.

*Expanded graph pruning.* To decrease the complexity of the XG and to increase efficiency, the non-schedulable paths (i.e., a set of \( a_{ij} \)) 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 only the paths that are schedulable.

*Execution of the composition logic.* The primary objective of the composition algorithm is to determine 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 data pieces: 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, i.e., the selected \( a_{ij} \) that corresponds to the \( S_{t, target} \). We define two main phases in the execution of the system. Firstly, we define an initialization phase where we perform an a priori study of the system by studying its service graph and mostly its service implementation graph. We analyze the schedulability of the graphs, their complexity, and perform fine tuning of the system in order to bound the sources of unpredictability. This phase may yield the calculation of the reconfiguration time that is application dependant. The second one is the mission phase that consists of the system in execution.

*Start the new application.* The new application (i.e., \( a_{ij} \) or \( S_{t, target} \)) is the target EG. This phase implies replacing old service implementations 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 [17]. In our case, we use immediate mode change, i.e., instant replacement of the old service implementations and instant start up of new ones.

### 3.3 Definition of temporal bounds

The reconfiguration time has application semantics, and it is, therefore, calculated considering application requirements. Details such as the nature, type, and characteristics of the data processed by the application, or its sensitivity to temporal behavior have a direct influence on the maximum allowed reconfiguration time. Also, the application structure and the complexity of the graph of service implementations affects the duration of the state transition. In order to time bound the reconfiguration, we have worked on the following essential levels:

Identification of the phases of the reconfiguration process with the goal of making them time-bounded,
Reduction of the complexity of the application graphs (i.e., service implementation graphs) to limit the complexity and time taken to search through them, and

Design of a bounded-time service-based composition algorithm based on simple graph search techniques.

Initially, we present the breakdown of the reconfiguration process into the different phases that ought to be followed to enable a state transition. Firstly, we define the duration of the reconfiguration time slot, i.e., \( b^c \) according to the model defined in [29]. In this slot, the different phases (shown in figure 4) must take place in bounded time.

![Figure 4. Reconfiguration phases and timing](Image)

Following, we describe the reconfiguration phases and how they have been bounded in time by imposing some limitations to the system model. The control manager, CM, is in charge of controlling these steps:

- **Trigger and initial processing (\( t^a \)).** 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 (\( t^gw \)).** 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 (\( t^a_c \)).** 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. An important part of this is the graph transformation time (or real-time pruning (trp)). It is very important to reduce the number of interactions between CM and QoSRM in order to decrease the number of schedulability test executions. Therefore, the XG is summarized, i.e., it is transformed into a reduced complexity graph that does not contain all paths, but only those that are representative of the complete XG.

- **Service composition time (\( t^s_c \)).** This process results in the selection of the new application to run, i.e., the new EG or execution graph. The selection is made by the composition logic (CL component) that decides among the different possible paths of the XG the one which is closer to the quality of service parameter (Q).

- **Mode change time (\( t^m_c \)).** After the new EG has been selected, the service implementations that it contains must be executed. Old service implementations are stopped and new ones are run. This is in practice a mode change. 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 require a progressive change. There are a number of time-bounded mode change algorithms in the literature that determine the sequence of steps to change to the new state such as [17]; examples of proposed mode changes are immediate that stop old service implementations and then start new ones or progressive that do the change smoothly.

The hardest phase to bound in time is the graph transformation 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 applications, there is a limitation on the duration of the time slot assigned to this process (\( t^p \)). If we define \( t^{int} \) as the interaction time between CM and QoSRM in such a way that \( t^p = t^{int} + t^p \), then \( t^p = b^c - t^a - t^gw - t^a_c - t^{int} \). 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.

4. **Separation of concerns: composition and schedulability**

In the reconfiguration of real-time service oriented systems, it is necessary to determine or find the target configuration of the system that is made of a set of service implementations. Two fundamental activities must take place in such a transition from the initial to the target state: (1) composition of services and (2) schedulability analysis. On the one hand, the target configuration must be schedulable, i.e., a schedulability analysis algorithm must be executed over the set of target service implementations to ensure that they will fulfill their execution deadlines at run-time. On the other hand, the selection of the set of service implementations of the target
configuration must be done by a service-based composition logic. Based on application-dependant criteria, such an algorithm will choose the set of service implementations that are closer to the application objective values. Criteria may depend on the specific application types, e.g., a value relative to the compression format or image quality for multimedia applications.

It must be considered that the whole process can be very complex if depending on the specific real-time schedulability model. This is clearly a tightly coupled approach that can yield a complex and time consuming reconfiguration protocol. Typically, service composition is silent about timeliness; however, in the very few cases when real-time has been contemplated, the schedulability analysis is tightly coupled inside the graph search process. This produces highly complex service-based composition algorithms that cannot be time-bounded.

In this paper, we propose a separation of concerns between the service-based composition model and the schedulability analysis. Therefore, we are able to produce a bounded-time service composition algorithm. Figure 5 presents the basic idea of our approach that was initially sketched in [27]. Here, we have further elaborated the context and motivation of the separation of concerns that was proposed for embedded real-time multimedia execution in [16], [30], and [31].

![Figure 5](image)

Figure 5. Bounded-time composition algorithm in the context of iLAND middleware

The sequence, types, and complexity of the interactions between components affect directly the temporal cost of the reconfiguration process. To guarantee that the reconfiguration phases will be performed in bounded time, the set of required operations must be simple and well indentified. This section briefly analyzes these interactions and studies the type of operations performed by each step especially focusing on the service composition phase. Two of the fundamental steps are: (1) schedulability analysis and (2) service composition.

One of the key ideas developed in this work is the separation of concerns between the schedulability analysis of the admission control protocol and the service composition algorithm. Such separation decouples the complexity of both activities allowing the system to use different real-time schedulability analysis algorithms of various complexity levels. For instance, a utilization based analysis can be executed in linear time, whereas an exact response time analysis test can be very complex and may require fine tuning of the temporal parameters of the services to be executed at run-time.

In a broad sense, a reconfiguration protocol requires the phases described below to be undertaken:

1. **Detection and initial processing of the triggering event.**

   For the sake of stability, only those events that trigger a reconfiguration must be detected. Others, must be filtered to avoid over-reaction. Therefore, upon arrival of a trigger event, the application that requires reconfiguration must be identified. The logic that the system requires to execute to discriminate those events that indicate a transient peak of load that are not sign of a permanent reconfiguration must be filtered out. Logic and algorithms for this purpose was proposed by [16], [34], and [35].

   Information relative to applications, services, implementations and temporal parameters is stored in the internal information repository managed by the QoSRM entity. Applications are discriminated by their unique identifier and they are stored in the form of application graphs or AG. Both the identifier and the AG are two read operations. As a consequence, their temporal cost can be determined a priori, and it will be executed within the \( t \text{ev} \) time slot.

2. **AG expansion**

   Our reconfiguration model that is defined in [28] and later used in [2] must finish execution in the predefined time slot for expanding the AG. This time slot named the graph working time \( t\text{gh} \) slot transforms the AG into the XG; this transformation is time-bounded by the a priori study in the initialization phase.

   Once the XG is obtained, an initial prune is made in it by removing the service implementations that are not eligible to run, e.g. they may have failed execution or are down. The initial prune work is an operation that is easy to perform in bounded time since only a sequential check of the status of current service implementations is done.
3. Reduction of the complexity of the solution space

The XG will contain the target system configuration (or target solution). However, the XG can be a very complex graph depending on the number of services and service implementations per service. The time required to search for the target solution can, as a consequence, be unaffordable. It is required that the complexity of the XG is checked a priori to determine the size of the solution space and schedulability of their solutions. Also, it is needed that the XG be reduced in size.

In this phase, the XG is transformed into a smaller graph named SXG, containing only some selected solutions. The associated time slot is named \( t' \). Each combination or possible solution of the SXG is sent to the QoSRM that will check 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 context including the required information is sketched in figure 6. We have defined the Bounded Composition Algorithm that searches through the SXG to select the target system configuration or EG that is the set of service implementations to be run. The service based composition algorithm 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. However, this approach is flexible enough so that the QoS criteria may vary depending on the specific application domain.

As shown in figure 6, the bounded time service composition 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 depending on the application domain. The ones used in our approach aim at:

- Fulfilling end-to-end response time of applications.
- Reducing the node utilization factor, i.e., incur in low individual processor utilization values for the service implementations.

5. Mode Change

The last part of the reconfiguration process consists of the launching of the execution graph (EG), i.e., of the service implementations that it contains by issuing system calls for the creation of the threads in the nodes where the service implementations are located. An immediate mode change algorithm as proposed in [17] is used that executes in linear time with respect to the number of service implementations concerned in the state transition.

5. Bounded time service composition based on \( n \) regions search

In this section, we revisit the approach proposed in [2] and we generalize it for supporting a finer grain search by defining \( n \) regions in the scheduled expanded graph (SXG). As explained in the previous section, the property of temporal determinism of the service composition algorithm is achieved by the strict separation of 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.

For the selection of the eventual service implementations that will be part of the execution graph, we rely on the usage of a set of data values that are calculated for the specific graph named heuristics. These values can be calculated in two different ways in the service composition algorithm, that yields to the classification of heuristics in two types:
*External heuristics* are the additional information that is pre-calculated outside of the composition logic to guide the selection process of the composition algorithm. This information is fed to the algorithm as data values that refer to indicators of the application quality criteria.

*Internal heuristics* are data values calculated along the process of execution of the composition logic, i.e., they are part of the service 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 ones:

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 \( a \) has services \( \{s_j,...,s_n\} \) and each service has one or more service implementations in the form \( s_{j,l},...,s_{n,m} \), etc. The set of paths of the AG of application \( a \) are named as \( i \). Each path contains a set of service implementations, i.e., one service implementation per service; therefore, \( i \) is the \( j \)th path of the application \( a \). The following heuristics are calculated within the graph:

- **Individual service heuristics**: each service has a value \( H_i \) resulting from the calculation of the average QoS value of all its service implementations in the following form: \( H_{ij} = \text{avg}(s_{j,l},...,s_{n,m}) \).

- **Graph heuristics**: they are calculated for the considering the whole graph: \( m \) is the graph path containing the service implementations that have the minimum QoS value; \( M \) is the path of service implementations with the maximum QoS value; \( a \) is the path of service implementations with the average QoS value of \( M \) and \( m \): \( a = (M + m)/2 \).

Initially, the desired end-to-end time for the application is checked against the value of \( M \) and \( m \). If the end-to-end deadline is higher than \( M \), then the optimal solution is the maximum path; if the end-to-end deadline is smaller than the \( m \), 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 \( n \) regions in the application graph as presented in figure 7 by adding extra control values that will guide the search, \( B^1 \) to \( B^n \). The number of regions, \( n \), of the algorithm influences the level of accuracy of the selected final solution.

![Figure 7. Reference regions and associated heuristics](image-url)

As indicated in figure 7, the set of values \( \{B^i: i>0 \text{ and } i<n\} \) are the intermediate control points established between the minimum and maximum path values, respectively. For reducing the complexity of calculations, we establish that regions are homogeneous in size \( (h) \), and the number of divisions \( (d) \) of the graph is an odd value so that branch and bound search can be applied, i.e., there is always a control point in the middle of the analyzed
spectrum. It must be considered that it is always true that the number of regions is an even value since: \( n = d+1 \). As a result, the set intermediate control values is calculated as equal size divisions of the graph as indicated in (1):

\[
m + \sum_{i=1}^{d} (h \cdot i) = B
\]  

(1)

Each value calculated by the search algorithm can be compared to any of the \( n+1 \) control points to know the current distance from the desired end-to-end QoS value. This draws a search path through the different regions that have been established in the graph.

The desired end-to-end deadline is assigned to the appropriate region that is defined by the two values that are calculated and that determine the region bounds for the algorithm to search from. The upper bound will be the application QoS value (e.g., deadline in our case) \( Q \); the lower bound will be \( B \) that is the limit of the region that contains \( Q \).

We define two guiding values or heuristics to guide the search process both at graph level (\( opt \)) and at service level (\( si \) level). \( opt \) gives a general guiding value at graph level, and \( si \) level gives a guiding value at each service level.

For each service, the search process takes all the service implementations and sorts them by QoS. Then, the algorithm calculates the \( si \) level, which is the corresponding \( opt \). \( opt \) gives a theoretical path to search within the graph. This theoretical path is variable, and it is adjusted by the search process based on the best possible quality, which is that one given by the \( lim \) or the upper limit for selection of the EG. Eventually, the path selected by the algorithm must be worse or equal (in the best case) than that of the theoretical path.

Let \( B \) be the lower bound of the selected region above, and \( C \) be the set of solutions contained within the bounds of the selected regions at each step of the algorithm. Then, the next lower bound \( B \) is defined as:

\[
sl < C < lim
\]

(4)

A linear search through 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.

With the reference value calculated (\( sl \)), 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 the algorithm will need to select another one with lower QoS.

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

6. Experiments and validation

The algorithm has been validated by implementation inside the the iLAND middleware and also extending 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 the proposed service model as described in
our previous work [25] and [1] which is an evolution to the service model of distributed interactions and component frameworks as [30], [32], and [33]. The bounded time composition algorithm has been studied with respect to its cost (execution time to obtain a decision, e.g., an EG) and the quality of the delivered result with respect to the theoretical optimum. The platform setting is comprised of 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 8 shows the behavior of the algorithm comparing the previous version (5 regions) versus the generalized version of this paper ($n$ regions). In this specific case, $n$ is set to equal 7. It can be seen that it executes in less than linear time even in the worst case. The logical observation occurs since the search through the graph is of finer grain for a larger number of regions. 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 8. Comparison of two execution of the service composition algorithm for 5 and 7 regions compared also to the linear behavior.](image)

Figure 9 shows the quality test for assessing the results output by the composition algorithm. 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 9. Comparison of quality values of two executions with various values of $n$.](image)

The quality measure used for this experiment is given by the following expression: $Q^a/t$, where $Q^a$ 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. The separation of concerns and a priori study of the graphs allows us to achieve an interesting property as a higher number of regions is used; in such a case, the higher the number of regions, the shorter time that the search will take.

In a video surveillance scenario, a distributed deployment with a medium size number of nodes has been synthesized as shown in table 1.

<table>
<thead>
<tr>
<th>Table 1. Distributed video surveillance application</th>
</tr>
</thead>
</table>
To ensure a fair quality, our system has to guarantee an output rate of 20 images per second. This implies that the service processing pipeline must ensure that a frame is generated every 50 ms. This value also sets the activation period of the services in the pipeline as well as the deadlines that are considered to be equal to periods which is compliant to both models, utilization or response time. If solutions were selected based on the timing analysis, application level quality parameters would not be taken into account. In the example above, every service implementation is assigned a quality value (q) that has application level semantics; it represents the value of each service implementation as perceived by the application as exposed in [36]. Therefore, the service composition algorithm will select the implementation that has a quality value closer to the specified application Q. For example, in the case of table 1, for an application with \( Q=5 \), the \( EG \) will contain the set \( \{ s_{\text{cap},1}, s_{\text{cap},2}, s_{\text{cap},3}, s_{\text{cap},4} \} \). If solutions were selected based on the timing analysis, the service composition algorithm is kept very simple just as a straightforward graph search guided by some reasonable margins and heuristics that are pre-computed. This is, in fact, the essence of the idea of separation of concerns. The benefit of the proposed generalized algorithm over our previous version proposed in [2] is the fine grain with which it will search near the optimum values. That benefit is obvious for large applications with large service implementation sets.

### 7. Conclusions

Handling the dynamics in the next generation distributed service-oriented 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. These new systems will require to be composed on the fly in a timely manner to build responsive systems. In this paper, we 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 establishing a clean separation of concerns (already developed in [16] and [34] as the germ of the idea proposed in this paper) between composition and timing analysis, the service composition algorithm is kept very simple just as a straightforward graph search guided by some reasonable margins and heuristics that are pre-computed. This is, in fact, the essence of the idea of separation of concerns. This paper enhances the contribution presented in [2] significantly. On the one hand, it performs an analysis task presenting a detailed description of the implications of service based composition from the perspective of real-time systems. On the other hand, it shows how a time-bounded solution is achievable by separating the real-time schedulability analysis algorithm from the service composition mechanism. Moreover, the paper provides a generalized solution based on the definition of \( n \) regions for finer grain graph searches. It has been validated and implemented inside the iLAND middleware and in synthetic scenarios. The proposed algorithm can perform a time-based search of service implementation combinations to compose an application, but it also allows the system to search for a service implementation set that is closest to an application-level quality parameter. Therefore, this algorithm supports application quality semantics.

### 8. References


ACKNOWLEDGEMENTS

This work has been partly funded by the ARTEMIS Call1 project iLAND (ARTEMIS-JU-1-100026) funded by the ARTEMIS JTI and the Spanish Ministry of Industry, Commerce, and Tourism. Also, this work has been partly funded by the ARTISTDesign NoE (IST-2007-214373) of the EU 7th Framework Programme, by the Spanish national projects REM4VSS (TIN-2011-28339), and by the Spanish Ministry of Education for International Research Stays (PRX12/00252). Authors would like to thank the members of the UC3M implementation team iLAND middleware, specially to Raul Fernández and also Iago López, L. Fernández, Patricia Uriol, and Felipe Ibáñez.
Prof. Marisol García Valls was born in Castellón de la Plana (Spain). She obtained her Computer Science Engineering degree from Universitat Jaume I de Castellón in 1996 and her PhD from the Technical University of Madrid in 2001. Currently, she is a professor at Universidad Carlos III de Madrid in the Telematics Engineering Department.

Since 2002, she is the head of the Distributed Real-Time Systems Lab of the GAST Group. Her research interests focus on resource management and quality of service for distributed embedded real-time systems, real-time middleware, operating systems, and service oriented architectures. She is reviewer of a number of SCI/JCR journals in these fields; also, she is member of the PC of a number of conferences in these areas. She has been enrolled in a number of European, national, and regional projects as the EU projects ARTIST and ARTIST2, and the European Network of Excellence ARTISTDesign. Currently, she is the scientific and technical coordinator of the EU project iLAND (ARTEMIS-1-100026) and the principal investigator and coordinator of the Spanish national project REM4VSS (TIN-2011-28339). On July 2012, she became IEEE Senior Member.

Pablo Basanta Val was born in Lugo (Spain). He obtained his Electrical Engineering Degree from University of Vigo in 2002, and the PhD degree from Universidad Carlos III de Madrid in 2007. Currently, he is assistant professor in the Telematics Engineering Department of Universidad Carlos III de Madrid, and he is a member of the Distributed Real-Time Systems Lab (Drequiem Lab) of GAST group. He has been member of national and European projects as REM4VSS. His main research interest is the Real-Time Specification for Java.