Reducing the space of solutions for real-time reconfiguration of complex data-intensive service-oriented applications

M. García-Valls, Patricia Uriol-Resuela, F. Ibáñez-Vázquez, P. Basanta-Val
Distributed Real-Time Systems Group
Department of Telematics Engineering, Universidad Carlos III de Madrid
Leganés (Madrid), Spain
{mvalls,pbasanta}@it.uc3m.es, {fibanez,puriol}@pa.uc3m.es

Abstract—Next generation distributed real-time systems will be complex high-performance environments containing applications with a flexible structure, integrating a large number of nodes of heterogeneous nature characterized by multiple and decoupled software units scattered all over the distributed environment; they are expected to offer data-intensive capabilities through merging the processing power of large numbers of nodes. These systems will have increased dynamic behavior by suffering frequent reconfigurations or state transitions resulting, among others, from the changing nature of the processed data. Handling the dynamics of these systems in real-time is a complex problem that requires to impose some bounds to the structure of the system to really achieve timely response not only during normal operation but also in the event of reconfigurations. In this paper, we present an approach to achieve real-time reconfiguration in distributed real-time service-based systems modeled as graphs. A reconfiguration requires to search for a new schedulable/valid solution or state from a complete system graph that contains all tentative solutions; each of these solutions will have undergo a schedulability analysis to determine if it is a valid solution; if the system graph is too complex, the overall time required for the schedulability check can be exponential with respect to size of services and service implementations; this may lead to an unbounded reconfiguration time. In this paper, we present an approach to reduce the complexity of the system graphs so that a summarizing one that contains valid solutions is analyzed and not the complete system graph. We have implemented this mechanism inside the iLAND service reconfiguration and composition components to validate the proposed concepts and ideas; the reduction of the space of solutions with the presented approach is very high, which dramatically decreases the computation time of the reconfiguration process.

Keywords: reconfiguration, real-time, SOA, distributed systems, complexity reduction.

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 increase 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. These transformations are called reconfigurations, i.e., the process by which a system that is in a current state evolves to a target state.

Software-level reconfiguration has been traditionally approached from very different domains. The software engineering point of view has mainly presented new component technology for easy (though not timely) replacement of functionality; real-time theory has developed mode changing protocols and algorithms to ensure time-bounded transitions; more recently, service-oriented paradigms have focused at service composition although being silent about timing issues since their main target is still general purpose distributed systems.

We target distributed real-time systems based on distributed services, 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. For this, it is essential to eliminate or, at least, alleviate all sources of complexity in the reconfiguration process.

The presented work is embedded in the context of the iLAND project [1] where we have developed an approach for time-bounded reconfiguration of distributed service-based applications. In iLAND, we define applications as graph based structures where nodes are the services and the links represent the message 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.

The reconfiguration process implies a selection of the new system configuration to make the transition to. This selection is performed according to both the schedulability or
timeliness of the solution and the proximity to the desired application-level quality of service. We consider the end-to-end processing time of a distributed application as the Quality of Service value. Prior to the selection of the new state or solution, all tentative solutions have to be identified; after, they undergo a schedulability test. The new state will be selected from the schedulable solutions according to their proximity to the application-level QoS value [20].

For the sake of efficiency and timeliness, the number of tentative solutions has to be as small as possible. 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 to obtain the new valid state to switch to 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 this paper, we provide an approach to reduce the space of possible solutions that a reconfiguration process will consider to decide on the new state for making the transition. Firstly, we present the reconfiguration process and its logical break down into several time-bounded phases. After, the most complicated phase that is the reduction of the space of solutions of a graph is dealt with. Our approach is validated showing how we improve substantially the reconfiguration time.

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 reduce the graph complexity with a mechanism named RTPrune. 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][4], 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 [5] where web services were applied to manufacturing and SIRENA and OASIS [6] where DPWS (Device Profile Web Services) middleware enables the interaction with the embedded nodes present at the factory floor. Also, other contexts for educational remote monitoring and control apply the service oriented paradigm [22]. 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 [7] merged web services with CAMEX for industrial electronics devices applied to 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 [8], 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 [9] proposed a similar approach based on resource reservations and estimations of the response time of services; a centralized algorithm for service composition is used that is based on the calculation of the shortest path.

Other approaches as [10] 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 as it uses a light real-time characterization which is not realistic in the domain.

Other real-time SOA efforts as [11] 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 [12][13][14] (in which a set of time-bounded steps were defined); or in the distributed systems domain, they were silent about timeliness as in [15][16]. 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 for overcoming the restriction of lack of timely reconfiguration by focusing on one aspect of the reconfiguration process that is essential to provide efficient reconfiguration in complex service-based systems; we present an approach for dramatically reducing the number of tentative solutions that must be checked for schedulability in the system; we provide a solution that is efficient in the transition and in the calculation of the target state based on the reduction of the complete space of possibilities. We identify the different reconfiguration phases, and we focus on the real-time pruning phase where the pruning of the space of solutions of the system takes place. Our previously presented reconfiguration contributions concentrate on in the iLAND middleware framework [1][21][20] providing low-level mechanisms for real-time mode changes as [13], providing resource management techniques based on priority management [14], time bounded server composition algorithms as [2], and a real-time schedulability framework based on budget scheduling as [17]. Also, other contributions aims specifically at the resource management policies for specific domains, such as embedded and resource constrained devices [19]. However, all of them leave aside the real-time
pruning phase which is essential to achieve real-time reconfiguration, and that is presented in this paper.

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 \( \{S_i\} \) 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 all tasks are approximated by periodic \([17]\), \( D \) 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 can be mainly of two 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, and they have been implemented inside the iLAND middleware.

![Figure 1](image)

**Figure 1.** Components involved in reconfiguration

Figure 1 presents a summarized overview of the iLAND architecture [1]. It has two main layers (CFL that is the Core Functionality Layer and CBL that is the Communication Backbone and Resource Manager Layer).

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 CBL 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 dependant.

- **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 stops all of the 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 pruned 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 target EG). This implies replacing old service implementation by new ones. The CM coordinates the mode change; there are different possibilities in the literature for this depending on the nature of the applications [13]. 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 protocol

In brief, figure 3 summarizes all interactions the involved components.

![Figure 2. Reconfiguration sequence through graph evolution](image)

![Figure 3. Overview of component interaction during the reconfiguration](image)
pruning time \( t_p \), service composition time \( t_s \), and mode change time \( t_m \). We define the reconfiguration time budget slot \( t^b \) according to the terminology defined in [17] as the application-dependent maximum allowed reconfiguration time. In this slot, the phases shown in figure 4 must take place in bounded time.

\[
\begin{array}{c|c|c|c|c|c}
\hline
b^o & t^o & t^p & t^m & t^{mc} \\
\hline
\end{array}
\]

Figure 4. Reconfiguration steps

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** \( t^i \). The detection of the reconfiguration event is performed in the first slot, and it is initially processed to determine if a functional or an internal transition must take place.

* **Graph working time** \( t^w \). 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** \( t^a \). 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 \( t^a \) is the graph transformation time or real-time pruning \( t^p \). 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** \( t^s \). This process results in the selection of the new application to run (the new EG, execution graph). The selection is made by the composition logic (CL component) that decides among the different paths the one which is closer to the quality of service parameter \( Q \).

* **Mode change time** \( t^m \). 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., [13]) 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 \( t^p = t^m - t^p - t^m - t^m \).

The remaining time, \( t^o \), 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 \( t^o \) 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. ASSURING TIMELINESS 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.

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 \( t^i \) 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, 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). These two initial steps (summarized in figure 5) are efficient in time since only a few time-bounded operations are done within the slot defined by \( t^p \).

Figure 5. Time-bounded reconfiguration steps 1 and 2
3) RTPrune logic ($t^0$).

In this step, the XG is transformed into a smaller graph that summarizes it; this new graph, SXG, contains only schedulable combinations. Each combination of the pruned SXG is sent to the QoSRM that checks its schedulability.

This process has three main points to be covered: (1) the pruning of the XG, (2) decomposition of the pruned XG to send the paths to be check by the QoSRM, and the (3) rebuilding of the graph with only the valid paths transforming it into the SXG. In our solution, we perform these three activities in parallel with different threads that exchange the paths through synchronized buffers. They are Path Sender Algorithm (PSEND), Path Selection Algorithm (PSEL), and Graph Builder Algorithm (GBUILD).

4) Service composition process.

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 choosing 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. REAL-TIME PRUNNING PROCESS

This section presents the process for the effective reduction of the space of solutions contained in the XG that later undergo the schedulability analysis of the admission control. In practice, this process is the transformation of the expanded graph, XG, into a smaller but representative graph, SXG, that contains only schedulable paths. Figure 6 shows the process for the reduction of the XG that results in the output of the scheduled expanded graph, SXG.

There are two ways to perform the reduction of the solution space of the XG:

- Exhaustive that checks the schedulability of all paths of the XG. It extracts each schedulable path from it; then, it sends it to the admission control to check its schedulability; once all the answers are collected, the new scheduled expanded graph, SXG, is built containing only the schedulable paths.

The exhaustive approach is, however, not suitable for a bounded-time reconfiguration process since the number of solutions depends on the size of the application graph and of the number of service implementations. Also, checking the schedulability of each single path can be unaffordable depending on the task model and the schedulability test that is selected. In the general case, if the number of service implementations in an XG is high, the number of solutions that will raise in an exponential way, making it impossible to achieve a deterministic behavior.

- RTPrune algorithm that transforms the XG into a SXG by means of reducing the complexity of the space of solutions of the XG. As a result, the number of schedulability tests is reduced since only a smaller set of possible solutions will be sent to the admission control. The RTPrune process is shown in figure 6.

![Figure 6. RTPrune internal characteristics](image)

We explain PSEL since it performs the reduction of the solution space of the XG.

A. Path Selection Algorithm (PSEL)

PSEL reduces the XG by selecting only a subset of paths that which be checked for schedulability. PSEL algorithm shrinks the XG by pruning those service implementations that are similar among themselves. The main idea behind it is to extract the most representative service implementations which have the highest probability of being schedulable, thus aiming at increasing the affirmative answers that will be received from the admission control. Nevertheless, due to the fact that some service implementations would be deleted, some possible schedulable solutions could be lost, but PSEL will provide at least a feasible solution.

PSEL is a two-phase algorithm that uses as selecting or reduction criteria: (1) the physical nodes and utilization factor and (2) the application QoS.

Node reduction phase. PSEL deletes all service implementations that belong to the same service and that reside in the same physical node that have higher utilization factor. This step prevents from iterative testing of some solutions that are very similar among themselves, and it provides the solutions with smallest utilization factor since these combinations are likely to have a higher schedulability probability.

Firstly, it sorts the set of service implementations for each service according to the physical node they are located in. After, the algorithm checks if there are any pair of service implementations in the same node, and then pruning
those with the higher utilization factor in the affirmative cases.

**QoS reduction phase.** It reduces the graph obtained at the node reduction phase by removing the service implementations that are more similar among themselves according to the QoS value.

Firstly, for each service, it sorts the set of service implementations according to the QoS value. Then, it calculates the distance value, \( d \), for all the service implementations of the graph as follows:

\[
d = q_{\text{max}} - q_{\text{min}} \tag{1}
\]

where \( q_{\text{max}} \) is the maximum QoS available in the graph, and \( q_{\text{min}} \) the minimum one.

Exploring pairs the service implementations for each service, the algorithm calculates the parameter likeness, \( l \), that measures the similarities among service implementations as follows:

\[
l = q_{i,j} - q_{i,j+1} \tag{2}
\]

where \( q_{i,j} \) is the QoS value of the current service implementation and \( q_{i,j+1} \) is the QoS value of the next service implementation of the same service.

To determine whether a service implementation is removed, a relation between the likeness and distance parameters is established. Those service implementations with the smallest QoS value can be removed according to this. This relation between likeness and distance is calculated according to the RTPrune parameter margin \((m_{\text{RTPrune}})\), which is the highest allowed threshold for the condition:

\[
d \leq m_{\text{RTPrune}} \tag{3}
\]

It is possible to fine tune this value to influence the RTPrune behavior. How this parameter affects the behavior of the RTPrune is shown in the validation section.

**B. Path Sender Algorithm (PSEND)**

PSEND algorithm establishes the criteria for sending the paths selected by PSEL to the admission control logic to be checked for schedulability. Also, PSEND implements the actual transmission and sending of the paths to the admission control.

The pruned graph obtained by PSEL is prepared by PSEND to send to the admission control all the possible paths that the pruned graph contains in order to check their schedulability. The algorithm extracts one by one the set of service implementations of the pruned graph that are computed by means of a formula presented in this section.

Prior to the extraction of the service implementations, some parameter need to be computed; they are:

- \( \text{Maxsolutions} \) which is the maximum number of number of paths contained in the pruned graph,
- \( \text{Max}(i) \) which is the maximum number of service implementations in the service \( i \), and
- \( \text{MaxSub}(i) \) that is the maximum number of solutions of the subset of the service implementations composed of the graph \( i+1 \).

To determine the service implementations, \( j \), to be sent to the admission control, the following formula is applied for all the services, \( i \), of the considered application:

\[
f = \left\lfloor \frac{a}{\text{MaxSub}(i)} \right\rfloor \% \text{Max}(i) \tag{4}
\]

where:

- \( i \) is the number of service for sending \([i \text{ from } 0 \text{ to (Maxserv } - 1)]\)
- \( j \) is the number of service implementation for sending within the service \( i \) \([j \text{ from } 0 \text{ to (MaxS} _i \text{ ) } - 1]\)
- \( a \) is the number of solution that the algorithm is sending \([a \text{ from } 0 \text{ to (Maxsolutions } - 1)]\).

**C. Graph Builder Algorithm (GBUILD)**

GBUILD receives the answer of the schedulability checks done by the admission control to the analyzed paths of the summarized graph. If the admission control determines that a given path containing a set of service implementations is schedulable, then it sends it to GBUILD that will store it to be part of the final scheduled expanded graph (SXG). As a consequence, the SXG contains only the schedulable solutions of the summarized graph. Therefore, the algorithm followed by GBUILD is simple:

If a given path does not pass the schedulability analysis, i.e., the solution is not valid, a service implementation from this solution must be deleted. The reason for this is that by removing one service implementation of this path, it will not be in the final SXG.

If there is any service implementation \( s_{i,j} \) that simultaneously belongs to more than one non-schedulable solution (negative answer), then only \( s_{i,j} \) is removed; this way, we do not remove one service implementation for each path that it belongs to. The reason for this is preserving a higher number of service implementations to obtain a richer graph.

GBUILD gives preference to removing service implementations from different services over removing them from the same service whenever possible; this favors the right balance of the graph whereas it avoid overloading a specific service implementation. Figure 7 presents an overview of GBUILD.
First of all, the algorithm should save the non-schedulable solutions taking into account the number of negative answers that each service implementation could have. Once all the answers have been received from the admission control, if there is any non-schedulable combination, the algorithm will look for and delete the service implementation that belongs to a higher number of non valid paths; then, it will update the values of the number of non-schedulable paths for each of the service remaining implementations. GBUILD algorithm repeats the same process until at least one service implementation can be removed from each of the non-schedulable solutions.

VI. COMPLEXITY REDUCTION

This section describes how the RT Prune algorithm works in the gradual reduction of the XG. Next, it performs the analysis of the computational complexity for realizing the importance of the previous reduction of solutions in each step of the RTPrune.

A. Graph reduction

The XG received by the RTPrune would be subject to several reductions until it becomes a final SXG. The total number of services cannot be reduced since it is an application-level requirement. The reduction is therefore done at the number of service implementations.

We define the set of possible states \( x \) of a graph that undergoes the reduction process:

\[
\begin{align*}
&x = \{\alpha, \beta, \gamma, \pi\} \\
&\alpha: \text{original expanded graph} \\
&\beta: \text{node reduction done} \\
&\gamma: \text{QoS reduction done} \\
&\pi: \text{non-schedulable solutions}
\end{align*}
\]

The \( \pi \) state will be further described in the subsection on computational complexity discussion. This state contains service implementation sets that the admission control logic identifies within any non-schedulable solution or path.

The set of service implementations (\( G^x_i \)) of the service \( i \) in state \( x \) is modeled as follows:

\[
G^x_i : S_{ij}(n,q) \quad \forall j \in J^x_i, \quad x = \alpha, \beta, \gamma, \pi
\]

Let \( S_{ij}(n,q) \) be the service implementation \( j \) of the service \( i \) with the physical node \( n \) and QoS \( q \) as parameters. Let \( J^x_i \) be the maximum number of service implementations that the service \( i \) has in the state \( x \). Hence the relationship between the different sets \( G^x_i \) is given by:

\[
G^\pi_i \subseteq G^\gamma_i \subseteq G^\beta_i \subseteq G^\alpha_i
\]

The set of service implementations after the physical node reduction \( G^\beta_i \) is computed as follows:

\[
\forall s_{ij} \in G^\alpha_i \quad \text{if} \quad n_{ij} = n_{ij+1}, \forall j \in J^\alpha_i \Rightarrow G^\beta_i
\]

\[
G^\beta_i = G^\alpha_i - s_{ij}
\]

The set of service implementations after the quality reduction \( G^\gamma_i \) is computed as follows:

\[
\forall s_{ij} \in G^\beta_i \quad \text{if} \quad q_{ij} - q_{ij+1} < \frac{q_{\text{max}} - q_{\text{min}}}{m_{\text{RTPrune}}}, \forall j \in J^\beta_i \Rightarrow G^\gamma_i
\]

\[
G^\gamma_i = G^\beta_i - s_{ij}
\]

Let \( f_{AC}(s_{ij}) \) and \( p_{AC} \) be the internal function and probability of the admission control logic (AC) in order to perform the schedulability analysis. Let \( f_{AC}(s_{ij}) > p_{AC} \) be the answer of the admission control in case that the solution that contains the service implementation \( s_{ij} \) is non-schedulable. The set of non-schedulable service implementations \( G^\pi_i \) is defined as:

\[
\forall s_{ij} \in G^\gamma_i \quad \text{if} \quad f_{AC}(s_{ij}) > p_{AC} \quad \forall j \in J^\pi_i \Rightarrow G^\pi_i
\]

\[
G^\pi_i = G^\gamma_i - s_{ij}
\]

The SXG and, therefore, the \( G^{\pi\text{XG}} \) depended on the set \( G^\pi_i \), the set \( G^\gamma_i \), and the maximum number of non-schedulable solutions \( q_{\text{max}} \) received from the admission control. Let \( G^{\pi\text{XG}}_i \) be the set of service implementations \( G^\pi_i \) with the exception of the subset \( e_{\text{del}} \) of \( G^\pi_i \), being \( e_{\text{del}} \) the number of service implementation eliminated by the Graph Builder Algorithm GBUILD. \( (e_{\text{del}} \text{ from } 0 \text{ to } q_{\text{max}}) \). GBUILD contributes in this form to the effort of reducing the graph by eliminating the service implementations with more non-schedulable paths in common.

B. Computational complexity

To analyze the computational complexity of RTPrune, we have adopted a pessimistic approach that considers worst-case measures.

Firstly, let us consider \( I \) as the maximum number of services of the graph.

The complexity in this first step is related to the node and QoS reductions. We establish that number of service implementations of service \( i \) initially is \( J^a_i \) whereas this value is \( J^\beta_i \) after the first reduction. The computational complexity of PSEL algorithm is given by:
\[ o(PSEI) = I \cdot (J_i^o)^2 + I \cdot (J_i^o - 1) \cdot (J_i^p - 1) + I \cdot (J_i^p)^2 + I \cdot J_i^p + I \cdot (J_i^o - 1) \cdot (J_i^p - 1) \]
\[ \max \Rightarrow I \cdot (J_i^p)^2 + I \cdot (J_i^p - 1) \Rightarrow I \cdot (J_i^o)^2 \]

There are two main groups that influence complexity in PSEI as follows:
- \( I \cdot (J_i^o)^2 + I \cdot (J_i^o - 1) \cdot (J_i^p - 1) \) relate to the node reduction, and
- \( I \cdot (J_i^p)^2 + I \cdot J_i^p + I \cdot (J_i^o - 1) \cdot (J_i^p - 1) \) relate to the QoS.
Both of them involve sorting all the service implementation of each service \( I \cdot (J_i^o)^2 \) and \( I \cdot (J_i^p)^2 \) and deleting all the service implementations required that are \( I \cdot (J_i^o - 1) \cdot (J_i^p - 1) \) and \( I \cdot (J_i^p - 1) \cdot (J_i^o - 1) \). Also, \( I \cdot J_i^p \) relates to the calculation of the previous QoS parameters.

**Path Sender Algorithm (PSEND - PSA):** Let \( I \) be the term corresponding to the calculation of the previous parameters necessary to perform the sending. The computational complexity of the PSA is given by:

\[ o(PSA) = I + \prod_{i=1}^{I} \left[ J_i^o \cdot (1 + I \cdot J_i^o) \right] \max \Rightarrow I \cdot \prod_{i=1}^{I} J_i^o \]

The algorithm tests all the possible solutions \( \prod_{i=1}^{I} J_i^o \), it means, the maximum number of combinations. For each sending it take one service implementation of each service \((I)\) and the data management in case the combinations are non-schedulable \((I \cdot J_i^p)\).

**Graph Builder Algorithm (GBUILD - GBA):** The last step of the RTPrune is highly depending on the negative answers received from the Admission Control \( \log \prod_{i=1}^{I} J_i^o \). The computational complexity of the GBA is given by:

\[ o(GBA) = \log \prod_{i=1}^{I} J_i^o \cdot \left[ J_i^o + (J_i^o - 1) + I \cdot J_i^p \right] \max \Rightarrow \prod_{i=1}^{I} J_i^o + I \cdot J_i^p \]

Let \( J_i^p \) be the searching of the service implementation that is going to be deleted \((J_i^p - 1)\). Once it has been removed the information of the rest service implementation non-schedulable needed to be updated \((I \cdot J_i^p)\).

**VII. VALIDATION**
We have carried out a set of experiments on a real platform with a full implementation of the iLAND middleware in a distributed environment. We present the validation tests that evaluate the RTPrune phase of the reconfiguration process. The tests focus on the study of the reduction of the complexity of the Expanded Graph in order to achieve a timely pruning process with the ultimate goal of suiting the needs of the real-time reconfiguration process of iLAND. The experiments have been conducted on an Intel Celeron E3400 @ 2.60GHz with 2GB RAM with an Ubuntu 10.04 LTS 32-bit and java 1.6.

On the one hand, the reduction of the complexity of the RTPrune process has been studied. The decrease of the number of service implementations at each step of construction of the summarized graph is shown by physical node, by QoS value, and by the number of schedulable paths. The number of different physical nodes of the space of solutions is shown in the experiments. This is indicated in table 1 that summarizes a set of 18 representative experiments containing different system configurations where the reduction of the number of nodes of the graph is shown. Also, we have studied the execution time of the process over our profiling tool Real Time Prune Simulator that implements the RTPrune process that indicates the response time of the algorithm.

At the input of the software tool implemented for the validation of the RTPrune mechanism, for each XG we allow to fine tune the number of services, service implementations, and the attributes of physical nodes and QoS applied to service implementations and application.

The RTPrune profiling tool is a software tool that can be prior to the run-time phase of the system in execution, and it runs the RTPrune algorithm in a standalone mode; this way, it offers an environment to study and fine tune the temporal behavior and complexity of the RTPrune process. The tool presents a graphical interface that enables the loading of several Expanded Graphs and entering the maximum allowed reconfiguration time of the system. We measure the time taken to obtain its Scheduled Expanded Graph inside the RTPrune slot. The tool can also perform a quality test consisting on executing the algorithm 100 times. The RTPrune profiling tool GUI is presented below in figure 8.

**Figure 8. RTPrune profiling tool**
The RTPrune profiling tool outputs the following information for every experiment or system configuration entered:

- **Max Possible Paths**: number of paths in the XG.
- **Schedulable Paths**: number of paths in the final SXG.
- **Max Reconf. Time**: maximum time required by the application level for completing a reconfiguration.
- **Actual Reconf. Time**: it is the measured time taken by the full execution of the RTPrune algorithm in order to build the SXG. It should not exceed the RTPrune Slot to guarantee the time-deterministic reconfiguration process.
- **Precision**: Ratio between the actual reconfiguration time and the maximum reconfiguration time.

In the GUI, there is only the final information, but by console also there is the possibility to display the same data for each component of the RTPrune mechanism.

The input graphs have some real-time simplifications in order to be compliant with the iLAND model:

- Period and Deadline are matched to the same value
- Priority management are not supported
- The schedulability analysis is based on the Utility factor in real-time distributed systems

For the sake of precision each experiment is executed 100 times and results are generated through the mean value of the obtained results.

Results have been collected to compare different input system configurations. The results are presented together with the Expanded Graph which has generated the output.

Table 1 presents the experimental results of the reduction of the solution space for various system configurations that present different complexity.

<table>
<thead>
<tr>
<th>SI no.</th>
<th>(s_{1 \cdot 3 \cdot 5 \cdot 4 \cdot 6})</th>
<th>(m_{RTPrune})</th>
<th>Path no.</th>
<th>(\text{Red. factor} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>XG</td>
<td>Node Red.</td>
<td>QoS Red.</td>
</tr>
<tr>
<td>[3,3,3]</td>
<td>30</td>
<td>81</td>
<td>54</td>
<td>24</td>
</tr>
<tr>
<td>[5,5,5]</td>
<td>30</td>
<td>625</td>
<td>192</td>
<td>81</td>
</tr>
<tr>
<td>[6,5,4,7]</td>
<td>30</td>
<td>840</td>
<td>192</td>
<td>51</td>
</tr>
<tr>
<td>[3,3,3]</td>
<td>10</td>
<td>81</td>
<td>54</td>
<td>8</td>
</tr>
<tr>
<td>[5,5,5]</td>
<td>10</td>
<td>625</td>
<td>192</td>
<td>27</td>
</tr>
<tr>
<td>[6,5,4,7]</td>
<td>10</td>
<td>840</td>
<td>192</td>
<td>18</td>
</tr>
<tr>
<td>[3,3,3]</td>
<td>8</td>
<td>81</td>
<td>54</td>
<td>8</td>
</tr>
<tr>
<td>[5,5,5]</td>
<td>8</td>
<td>625</td>
<td>192</td>
<td>12</td>
</tr>
<tr>
<td>[6,5,4,7]</td>
<td>8</td>
<td>840</td>
<td>192</td>
<td>12</td>
</tr>
<tr>
<td>[3,3,3]</td>
<td>8</td>
<td>729</td>
<td>324</td>
<td>144</td>
</tr>
<tr>
<td>[5,5,5]</td>
<td>8</td>
<td>15625</td>
<td>3072</td>
<td>972</td>
</tr>
<tr>
<td>[6,5,4,7,6,3]</td>
<td>8</td>
<td>15120</td>
<td>2304</td>
<td>324</td>
</tr>
<tr>
<td>[6,5,4,7,6,3]</td>
<td>10</td>
<td>792</td>
<td>324</td>
<td>32</td>
</tr>
<tr>
<td>[6,5,4,7,6,3]</td>
<td>10</td>
<td>15625</td>
<td>3072</td>
<td>162</td>
</tr>
<tr>
<td>[6,5,4,7,6,3]</td>
<td>10</td>
<td>15120</td>
<td>2304</td>
<td>72</td>
</tr>
<tr>
<td>[6,5,4,7,6,3]</td>
<td>8</td>
<td>792</td>
<td>324</td>
<td>8</td>
</tr>
<tr>
<td>[6,5,4,7,6,3]</td>
<td>8</td>
<td>15625</td>
<td>3072</td>
<td>48</td>
</tr>
<tr>
<td>[6,5,4,7,6,3]</td>
<td>8</td>
<td>15120</td>
<td>2304</td>
<td>16</td>
</tr>
</tbody>
</table>

The prototype applications considered consist of a graph structure with 4 services residing in 4 nodes and 6 services residing in 6 different nodes. The latter exhibits a complex structure with expanded graphs that gather up to over 15,000 possible solutions to be explored. Different experiments presenting varying number of service implementations for each of these 4 and 6 service graphs are presented.

The first column presents the number of service implementations for each one of the four services that is set to 3, 5, and a varying number. The number of service implementations has been set in order to check and validate the reduction of the amount of paths in different scenarios. The RTPrune Margin \(m_{RTPrune}\) is set to 30%, 10% and 8%. This parameter controls the aggressiveness of the prune process in terms of QoS; a smaller \(m_{RTPrune}\) implies that only service implementations with very different QoS among themselves will survive for the SXG.

For the four different states of the graph reduction process, table 1 shows the number of paths that are contained in the graphs. The four stages are the XG (expanded graph), node reduction stage, QoS reduction stage, and final SXG.

It is evidenced how the reduction factor achieved by the RTPrune process is considerably high (above 90%) for all cases, and over 98% for all cases of a graph with six initial services that contains over 15000 paths at the XG stage. The reduction factor describes the number of paths that are eliminated from the original XG. The main prune in the number of paths happens at the stages of node reduction and QoS reduction. This is also shown for one of the scenarios with 4 nodes in figure 9; figure 10 shows how the same trend is observed for 6 nodes.

![Figure 9](image)

**Figure 9.** Reduction factor of the number of paths for the different stages with 4 service graphs

![Figure 10](image)

**Figure 10.** Reduction factor of the number of paths for the
different stages for 6 service graphs

For all cases, the final SXG contains over 90% less paths than those contained in the XG; as a consequence, the number of paths sent to the admission control for schedulability analysis is dramatically decreased. Therefore, it shows the efficiency of the whole RTPrune process and, as the ultimate goal, of the reconfiguration of the system.

VIII. CONCLUSIONS

Handling dynamic behavior in 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 reconfiguration of them will be required in order to build responsive systems. In this paper, we have described an approach for reducing the complexity of system graphs in order to increase achieve time-bound reconfiguration. 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. However, the difficulty is passed to the graph reduction process as presented in this paper. The outcome of the approach is a graph that summarizes the system graph, so that the schedulability process is applied to a simple graph with a limited number of paths in it. We have shown the results obtained from its actual implementation on an existing middleware that validate the presented approach, showing its suitability for the soft real-time domain.

ACKNOWLEDGEMENT

This work has been partly supported by the iLAND project (ARTEMIS-ITU 100026) funded by the ARTEMIS ITU 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 REM4VS (TIN 2011-28339).

REFERENCES