iLAND: An Enhanced Middleware for Real-Time Reconfiguration of Service Oriented Distributed Real-Time Systems

Marisol García Valls, *Senior Member, IEEE*, Iago Rodríguez López, and Laura Fernández Villar

***Abstract*—Future networked embedded systems will be complex deployments deeply integrated in the environment that they monitor. They will have to react to both user and environmental events, and this may require modifying their structure to handle the changing situations. In a number of domains including industrial environments, this modification of the system structure will need to take place in real time. This is a hard problem that will require novel and paradigmatic solutions involving cross-domain knowledge to build a middleware for enabling real-time interaction between nodes allowing time-bounded reconfiguration. For supporting real-time, this middleware must be vertically architected in a modular way from the network and operating system levels up to the application software and the real-time policies for achieving time-bounded behavior. This paper presents a middleware that addresses these characteristics supporting timely reconfiguration in distributed real-time systems based on services. Experimental results are shown to validate the middleware in an actual small-scale video prototype.**

***Index Terms*—Distributed systems, middleware, real-time systems, reconfiguration, service composition, SOA.**

I. Introduction

CURRENT computing environments are becoming complex infrastructures containing numerous collections of nodes of heterogeneous nature forming a cloud of processing points. These nodes sense, process, actuate, and generate information for human operators and/or for other nodes and subsystems. Information captured in different parts of the complex system can be processed and delivered in real time to other remote spots to further process it and generate the required results on time. This is, for instance, the case of a complex video surveillance system with content analysis for monitoring of industrial processes in hostile environments where no human operators are present. In the near future, these infrastructures are expected to perform cross-processing of huge bulks of information in real time.

To enable the existence of complex computing environments, a flexible approach for their design, development, and interoperability is needed. A natural way for this is to allow its subparts to interact in a decoupled manner. Envisioning other limiting possibilities (as monolithic development) is really hard at the present history of technology and in a context where further application domain possibilities and user demands appear in the market so fast.

Introducing decoupled interaction solutions will solve part of the problem, but it will bring in other challenges related to security, distributed information processing, synchronization, and timeliness, among others. Precisely, real-time operation of such complex infrastructures will be one of the key challenges that must be handled from a multidomain perspective.

Currently, a clear trend is to support system reconfiguration as part of the dynamics of these complex environments. A reconfiguration implies the transition between two different system states. A number of events may be generated through the system that require a timely reaction to handle either unexpected or even expected (application defined) situations. The infrastructure (i.e., the nodes and their connections) will have to adapt the execution and even the node links to handle new events, offering a correct answer to the new system state.

With this challenging problem in mind, we describe the vision towards real-time reconfiguration from a system level view presenting the middleware architecture developed in the iLAND[1] project; it is a platform-independent solution for supporting time-bounded operation and reconfiguration in service-oriented distributed soft real-time systems. The architecture defines the set of components and the associated logic to provide time-deterministic guarantees. Related state-of-the-art solutions do not integrate built-in algorithms and mechanisms for supporting time-bounded reconfiguration. They either: 1) concentrate on application-level service *composability* techniques being silent about real-time characteristics or 2) when targeting real-time systems, they deal mainly with low-level processor scheduling with no realistic considerations of distribution issues.

The outline of this paper begins by describing the problem in Section II. Section III presents the related work. Section IV describes the concept of real-time reconfiguration and the application model; the iLAND architecture is presented in Section V. Section VI validates it in a small-scale industrial video surveillance application. Section VII concludes the paper.

Manuscript received August 30, 2011; revised December 22, 2011; accepted February 28, 2012. Date of publication May 10, 2012; date of current version December 19, 2012. This work was supported in part by the iLAND Project (ARTEMIS-JU 100026), by the ARTISTDesign NoE under Contract EU FP7 IST-2007-214373, and by the Spanish national project REM4VSS under Contract TIN-2011-28339. Paper no. TII-11-461.

The authors are with the Telematics Engineering Department, Universidad Carlos III de Madrid, 28911 Madrid, Spain (e-mail: mvalls@it.uc3m.es).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2012.2198662

[1][Online]. Available: http://www.iland-artemis.org.

1551-3203/$31.00 © 2012 British Crown Copyright

II. Problem Presentation and Approach

A. Requirements

In the first phase of the iLAND project, the above presented issues with respect to complex networked embedded systems (NES) have been extensively studied, and a set of requirements have been identified for the architecture of the software and the role of the different hardware nodes. Related solutions have also been analyzed in detail. Knowledge of the industrial experts on different small-, medium-, and large-scale deployments has been incorporated, and crosschecks with them for fast creation of prototypes has been key to successfully obtain the problem description and the requirements. The project target systems are, in the end, closely related to the concept of cyber physical systems (CPSs) [1] that include high-confidence medical devices and systems, assisted living, traffic control and safety, advanced automotive systems, process control, and energy conservation, among others. These studies have concluded by collecting a set of specific requirements that are the following:

- *Heterogeneity*: solutions must be adequate for all platforms, so they must be easily customizable.
- *Interoperability*: systems of different nature will interact though simple and well defined interfaces. Besides, most existing NES will be networked; suitable protocols are data-efficient and lightweight and specifically designed to facilitate interoperability.
- *Time-bounded operation*: real-time properties are present in most of current environments with the only difference of their temporal criticality level. For some, timeliness is just a clear added value whereas for others it is a mandatory requirement. In the context of this project, we deal with soft real-time requirements.
- *Dynamic behavior*: it brings in a complex run-time problem since it requires to deal with functionality replacement, load-balancing, and system self-healing, among others.
- *Composability*: upgradability, component reuse, flexible and incremental development, maintainability, and scalability are required in the majority of current developments.
- *Integration*: system integration can be carried out at software/hardware levels. Modular design is key for achieving integration, based on either components, services, or standard communication protocols.

B. Vertical View

To meet the above mentioned requirements, a vertical architectural approach is needed that considers not only the execution platforms but also the software stack that is capable of dealing with both the application logic and the extra intelligence necessary for supporting dynamic reconfiguration. In this case, the software stack will contain the required layers that embed extra intelligence at different levels:

- OS: enhanced primitives will be needed to manage execution of individual threads, thread groups, and to provide the basic synchronization and timing facilities.
- Resource Managers: as enhancement to the OS, resource managers will execute the algorithms for resource scheduling; resources will be assigned to the active entities (i.e., applications, threads, or thread groups). These algorithms need to be effective and feasible online.
- Middleware: it is the minimum communication abstraction layer providing an efficient distributed software paradigm for flexible application programming, supporting the specification of QoS-based execution properties and temporal characteristics.

C. Timing Requirements and QoS

There are different levels of temporal guarantees. Some components require specifying a deadline for performing their operations, so every individual job must fulfill the deadline; otherwise, the system may fail to work properly. However, other components may also have temporal constraints, but the failure of some operation to meet some deadline (maybe even influenced by the network effects) does not fully threaten their correct operation. Examples of such component operations may be the detection and reaction to some alarm and the display of background information, respectively.

Current practice for meeting temporal guarantees in critical parts uses dedicated HW and SW units with an extensive schedulability analysis. However, the trend is more and more to execute all software in general-purpose HW/SW platforms and networks based on TCP/IP protocols and Ethernet. Mixing software components with different criticality levels in the same platform threatens the timely execution of the most critical ones. This is especially evident if also the network effects and Ethernet collision are considered. Therefore, to guarantee timeliness the appropriate solutions have to be put in place ensuring temporal isolation; for this, the operating system specifics, multiresource scheduling, and efficient middleware architectures that aid timely interaction in a networked environment must be considered.

D. Contribution

In current solutions, the integration of new software platforms is based on the usage of direct communication protocols or general purpose middleware. However, support for dynamic behavior, QoS-based execution, and online application composability is not specifically addressed in a combined way in any solution. Although some online application composition techniques exist, they have been applied outside the context of real-time systems. In this work, the time-deterministic dimension applies to both the execution time of services (including their communication) and to the duration of the reconfiguration process. We describe a middleware that addresses the above-mentioned requirements providing support for dynamic behavior with QoS guarantees in the context of a flexible software paradigm for composability of soft real-time distributed applications. Therefore, time-deterministic reconfiguration is managed from an integrated perspective by providing the following.

- A vertical architectural design for a middleware that enables distributed SOA-based applications and that supports: 1) real-time execution and 2) real-time (or QoS-based) message exchange among services and nodes.

- The software architecture shows the specific hooks to support real-time execution. It is detailed by presenting the component, structural, and behavioral views.
- A reconfiguration protocol and a service-based composition algorithm that obtains new system configurations in bounded-time and with bounded-error according to the user/system specified criteria. The location of these mechanisms inside the middleware is described in the components, views, and diagrams of the architecture.

III. Related Work

Most popular application domains of distributed systems are not strictly sensitive to real-time, and therefore, most research work either does not consider the temporal aspects in their dynamics or does not handle it adequately in the different required dimensions (i.e., network, operating system, communication middleware, and application-level programming paradigm). It is also true that working strictly at network level can bring in more efficiency and temporal control in networked real-time systems implementation; however, middleware-based solutions have progressively gained popularity by trading-off some efficiency and predictability for flexibility and programmability. Therefore, quite some research effort has moved from the low level network details to the design and development of middleware in different domains as multimedia [2] or CPS [3].

At the network level, real-time traffic reconfiguration for service-based systems has been partially handled as shown in [4], where the service composition is done in a central master entity inside a time-triggered network. Here, new configurations are sent every elementary cycle by means of a trigger message supporting a basic level of reconfiguration.

On the side of communication middleware, there are a number of solutions with distinct acceptance levels as, for instance, the former popular CORBA [5] (and RT-CORBA) that offered distributed communication based on well defined interfaces; they are currently abandoned mainly due to their complex architecture and the associated execution overheads, especially if compared with other lighter weight technologies that are gaining market position as Internet communications engine (ICE) [6], data distribution service (DDS) for real-time systems [7], or language-dependent solutions as [8] for the real-time Java language.

Current trends on engineering of dynamic distributed systems mostly rely on the service-oriented architectures (SOA) paradigm and on enhanced middleware solutions that use common exchange languages as XML. The majority of SOA systems and related middleware target web environments as [10], [11]; also, research on dynamic SOA-based systems focuses on web services composition based on ontology crosschecks and dependency resolution [12]. The temporal aspect is typically not a concern in web service environments as further evidenced in [13], and they are silent about finding a solution in bounded-time. QoS crosschecking applied in web-based service composition techniques are mainly restricted to application-related levels and data semantics [14]. Over the last few years, SOA paradigms have been applied to resource-limited embedded systems [15], and, more recently, the real-time embedded community has also stepped into scene-providing architectures for gateways to differentiate real-time from non real-time parts of the service-based distributed system [16]. Therefore, some contributions are appearing at different levels as QoS properties for service execution and inter-service interaction [17], middleware support for service-based applications, and composition algorithms theory [18].

Dynamic execution has also been addressed by means of virtual machines that provide, among other facilities, transparent service execution and portable code. These environments are typically thick software layers that abstract the real platform. However, real-time services require a deterministic execution platform integrating a suitable scheduling scheme as [19], [20]. Advanced features as service migration and code downloading is still a threat for predictability, and other existing architectures such as OSGi [21] (and some attempts to integrate timeliness as [22]) are still far from being real-time.

Some contributions from the software engineering side have built reconfigurable middleware studying the reflective aspects to support interoperability, discovery, and mobility through pluggable overlay networks [23]. Although our middleware can be deployed in various versions, its configuration must be done off-line; therefore, our focus is different since the middleware does not reconfigure itself at run-time but it supports the timely reconfiguration of applications.

Summarizing, the difference between the existing solutions and iLAND middleware is twofold: 1) we explicitly consider the temporal behavior of distributed systems, and 2) we target at achieving time-bounded reconfiguration to support functional variations for changing the structure of the distributed system at run-time in bounded-time. Currently, there is no solution that integrates support for real-time communication and reconfiguration through service-based composition to achieve time-bounded reconfiguration. This paper presents a contribution in this precise direction by architecting a QoS-aware component-based middleware that supports the execution and communication of service-based applications and their reconfiguration in real-time. The first stage of the design and architecture of the middleware was introduced in [24] and the application model in [25] and [26].

IV. System Model

A. Reconfiguration Model and Real-Time Properties

The presented middleware targets distributed soft real-time systems, where failing to meet some deadline need not be terrible and acceptable performance can still be achieved; it may happen that only a degraded output is delivered, but this result may still be correct and of a certain utility. As a consequence, soft real-time systems use the concept of QoS, which relates to the ability of trading off the resources used by tasks for the delivered quality. In our model, QoS attributes have two dimensions [25] depending on their relation to: 1) the physical resources and 2) the application-level functionality.

Dynamics introduce execution uncertainty mainly related to either functional or resource variation. Functional variation refers to the replacement of some self-contained activity (or functional unit[2]) by a different one possibly delivering distinct results. Resource variation refers to the different possible assignment of resources to the different existing functional units. Both variation types are inter-related. For instance, a replacement of a video functional unit by an audio one will result in a variation in the resource assignment since audio processing is, on average, less resource consuming than video processing. However, a variation in the assignment of resources is not always the result of a functional change; it can be an internal monitoring decision in response to, for example, a recalculation of the load balance.[3]

In this context, a system reconfiguration occurs whenever there is a functional or a resource variation, which implies to modify either functional units or their resource assignment. In any case, it implies a transition from one system state ($\mathrm{ss}_{\mathrm{init}}$) to a target state ($\mathrm{ss}_{\mathrm{target}}$). A reconfiguration is, therefore, part of the dynamic behavior of a system. Consequently, timely reconfiguration is achieved whenever the transition from $\mathrm{ss}_{\mathrm{init}}$ to $\mathrm{ss}_{\mathrm{target}}$ is performed in less than the specified time t_{re} as

$$f(\mathrm{ss}_{\mathrm{init}}, \mathrm{ss}_{\mathrm{target}}) < t_{\mathrm{re}}.$$

The goal of real-time reconfiguration techniques is to develop the suitable mechanisms and architectures to develop a framework where the function f can be time-bounded. From a software-architect point of view, f depends on a number of levels that must be carefully considered.

- Hardware: effects of instruction processing, cache, and the processor architecture directly influence the execution timeliness and predictability.
- Operating system: a real-time operating system has time-bounded primitives, so efficient resource managers are implemented mostly inside the kernel to offer enhanced resource scheduling mechanisms.
- Middleware: communication abstractions that hide node heterogeneity and networking effects have also implications on the predictability of the system.

B. Software Model

A service-oriented model has been chosen as the software paradigm, since it increases design and execution flexibility. *Services* are self-contained functionality pieces that enable the building of distributed applications in a decoupled way, i.e., services reside in remote nodes in the network and communicate via messages or events. In this context, a service manages a data channel as shown in Fig. 1; it receives data through some input interface, processes the data, and generates a result that can be delivered to other services through its output interface.

It is possible to build extended functionality (applications) by connecting services. *Service-based applications* are a set of connected services in the form of a graph, where the *nodes* are the services and the *connecting lines* or arrows are the messages exchanged between them. Service connections inside an application can have dependencies that can be both functional (the type of data processed by two connected services must be compatible) and nonfunctional (e.g., execution timeliness and resource requirements). In this model, a service is characterized by the functionality it offers. A service (which is a conceptual functionality-driven entity) is realized by one or more *service implementations* that are the actual running entities. A service implementation is a particular version of a specific service [see Fig. 2(a) and (b)].

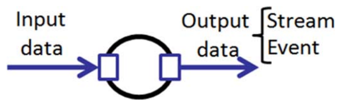


Fig. 1. Service data channel.

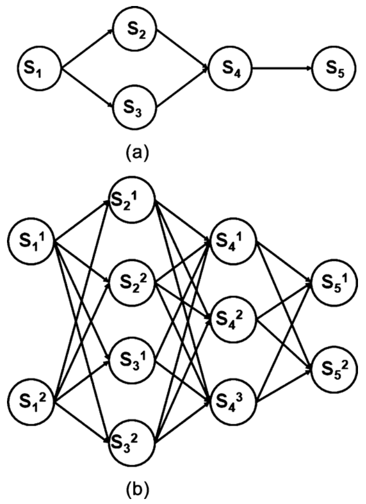


Fig. 2. (a) Application as a graph structure (AG or *application graph*). (b) Expanded graph (XG) containing the different service implementations of each service of the AG.

Therefore, a running application is a collection of *service implementations*, and, as a consequence, an *execution graph* contains only service implementations in execution. We define three types of graphs:

Application graph (AG) is one that contains only services. It is the static view of a desired application. $\mathrm{AG}^i = \{S, R, Q\}$, where S is the set of services of application i; R is the set of relations (arrows) between nodes and contains elements of type $S_j \rightarrow S_k$, where service S_j is connected to S_k in the graph; Q is the set of quality of service parameters related to the specific application data processing and to the needs for resources.

Expanded graph (XG) contains all possible service implementations. $\mathrm{XG}^a = \{\mathrm{SI}^a, R', Q\}$, where SI^a is the set of service implementations of application a; SI_i is the set of service implementations of service S_i ($\mathrm{SI}_i = \{s_i^1, \ldots, s_i^n\}$); R' is the set of relations (arrows) between nodes and contains elements of type $S_j^a \rightarrow S_k^d$ where service implementation S_j^a is connected to S_k^d in the graph; Q is the same set of quality of service parameters of its application graph.

Execution graph (EG) is the application in execution; it contains running service implementations. $\mathrm{EG}^a = \{\mathrm{SI}^a, R'', Q\}$

[2]A functional unit is an execution unit which may either be a task, operating system thread, active component, or service, depending on the jargon of the run-time system or the considered execution platform.

[3]HOLA-QoS [28], [27], [20] deals with this correspondence by off-line profiling of applications through the assignment of the different resource requirements to tasks and relating it to the different output qualities. iLAND follows this principle, and we apply it to the concept of *service* instead of task or thread.

where the set of service implementations contained in SI^a are only the ones that have been selected by the composition logic (later explained in Section V); R'' is a subset of R', and Q has the same structure as for the XG^a.

The service-based approach suits most naturally the dynamic behaviour of systems (i.e., where a *reconfiguration* must take place by modification of the execution graph). A reconfiguration is, therefore, the transition from one execution graph to another. Triggers for reconfiguration can be diverse: 1) *user action* as a request for a change of a functionality piece in the application or an explicit request for a change in the output quality delivered by some application; 2) *internal system decision* usually detected through system monitoring; this can be the case when the load of the system must be balanced and the required mechanisms are launched for this purpose; and 3) *Application decision* due to some application-specific condition or programmed event. These triggers lead to two reconfiguration types: 1) functional, where a service has to be started, stopped, or replaced, and 2) internal, where a service implementation is replaced.

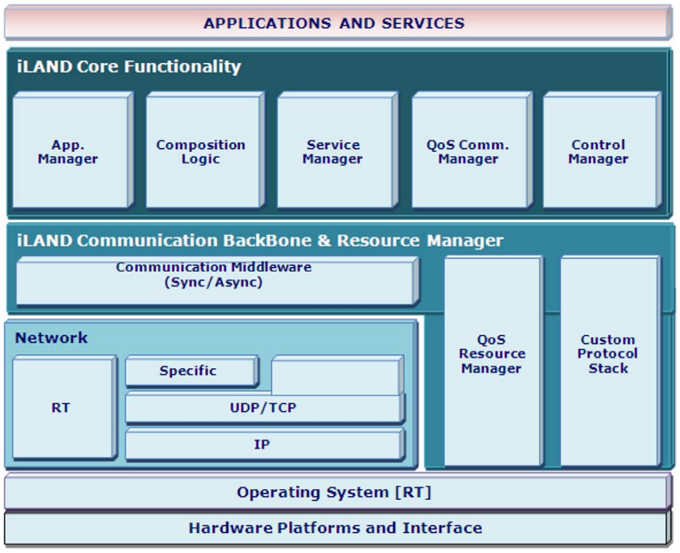


Fig. 3. iLAND middleware architecture.

V. Middleware for Real-Time Reconfiguration

To achieve real-time reconfiguration in iLAND, an *a priori* study of the system must be undertaken where the temporal analysis of the system is performed. *a priori* study of the system allows also to discard nondeterministic executions and complex graph structures that will lead to a combinatorial explosion of the system state set. In real-time systems, it is not possible to achieve time-bounded reconfiguration in the complete absence of knowledge about the system structure, its temporal requirements, and the characteristics of its constituent parts. For this reason, iLAND distinguishes different phases, given here.

- *Initialization phase* consisting of the specification of the system by establishing the application structure, its service set, the service implementations of each service, etc. In this phase, the off-line study of the system and its timing properties (graphs and temporal cost of the transitions) is carried out. This phase brings in knowledge about the future run-time behavior and possibilities of the system.
- *Operation phase*; after passing an initialization phase, the system is ready to execute. In this phase, the composition of the services and their execution takes place.

In the initialization phase, a default configuration is stored by the middleware; this *default configuration* is a system state that is of the minimum acceptable quality and, therefore, it has been off-line proved to be always schedulable. The purpose of this *default configuration* is to have a system state that could be executed in case that the reconfiguration time slot is not fulfilled by the middleware at some point of the execution.

A. Architecture

iLAND middleware architecture follows the classical principles of a layered middleware, adding a number of extensions for specific cases:

- support of service oriented applications;
- integration of time-deterministic reconfiguration techniques and service-composition algorithms;
- real-time communications by defining the complete network protocol stack (i.e., time-triggered level-2 media access control to enable schedulability analysis);
- portability to different communication off-the-shelf middleware backbones such as DDS or ICE, and also integration with DSA [9]; it is achieved by the definition of a *common communication bridge* for synchronous and asynchronous interaction models.

The above-mentioned characteristics are reflected in Fig. 3 that presents the overview of the middleware architecture.

The *Core Functionality Layer (CFL)* contains most of the key added-value functionality related to application management and reconfiguration control. Its components are given here.

- *Service Manager (SM)* contains the primitives for declaration, deletion, and modification of the properties (including their resource requirements) of individual services and their particular implementations.
- *Application Manager* (AM) component includes the logic to define the structure of an application (service graph), i.e., its services and their connections. Application-level QoS parameters (i.e., end-to-end properties for the whole service graph) are specified using this component.
- *Composition Logic* (CL) contains the logic for service-based composition consisting of finding a set of services to construct a specified application compliant with the specified QoS parameters. Examples of composition criteria are minimizing the memory consumption of all services or minimizing/meeting the end-to-end deadline. Since each service can have different versions that realize it, the complete graph to search for a valid solution can be of high complexity; this is eliminated in iLAND by defining different phases in the design, development, and execution of the system. Algorithms should be efficient and time-bounded since they will be executed on-line.
- *Control* Manager (*CM*) component contains the logic for coordinating the reconfiguration of the system; it captures the reconfiguration triggers and initiates the process that

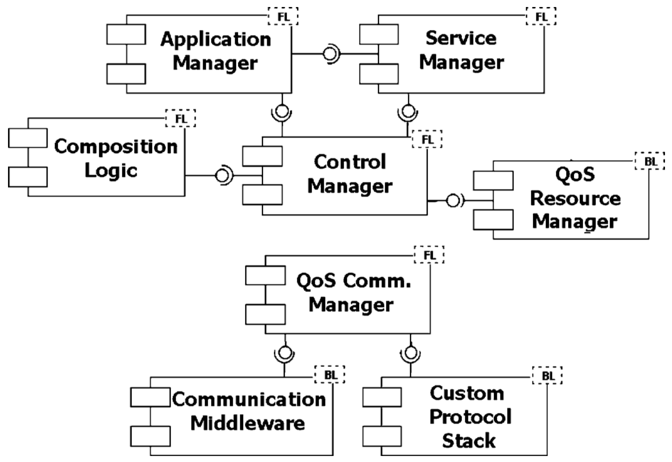


Fig. 4. Component diagram.

has different individual steps. The duration of each step is also controlled to perform a bounded-time transition.

- *QoS Communication Manager* (QS) component configures the global QoS parameters for the communication.

The *Communication Backbone and Resource Management Layer (CBL)* includes the following components.

- *Backbone Core Communication Middleware (CC)* or core communication model. It allows to use and benefit from any of the basic middleware interaction: P/S (pub/sub), MOM (message oriented middleware), DOM (distributed object middleware), or RPC (remote procedure call).
- The *QoS Resource Manager (QoSRM)* aims at QoS-based resource management. Scheduling for multi-resource management lies inside this component. It uses the primitives of an RTOS (i.e., priority-based preemptive scheduling and access to higher resolution timers) to implement budget scheduling techniques providing the basic resource usage accountability, temporal isolation, and mode change protocols.
- The *Custom Protocol Stack (CPPS)* provides the network traffic management. It allows to offer backward compatibility with legacy systems as well as supporting applications with hard real-time requirements by including real-time network scheduling protocols.

The *Network Layer (NL)* contains the basic functionality for real-time transmission on general networks that offer TCP/UDP over IP. An application-specific communication protocol can be used in the *specific* module, such as streaming communication with specialized protocols as RTP enhanced with RTCP-based communication at transport level.

B. Component and Structural Views

The component view of iLAND is presented in Fig. 4, showing the most relevant modules of the middleware as the constituent components.

This view is complemented by the structural view (in UML) outlined in Fig. 5 that is a class diagram showing the functionality of the components of iLAND and their interfaces. It can be seen that the SM and AM components allow the specification of

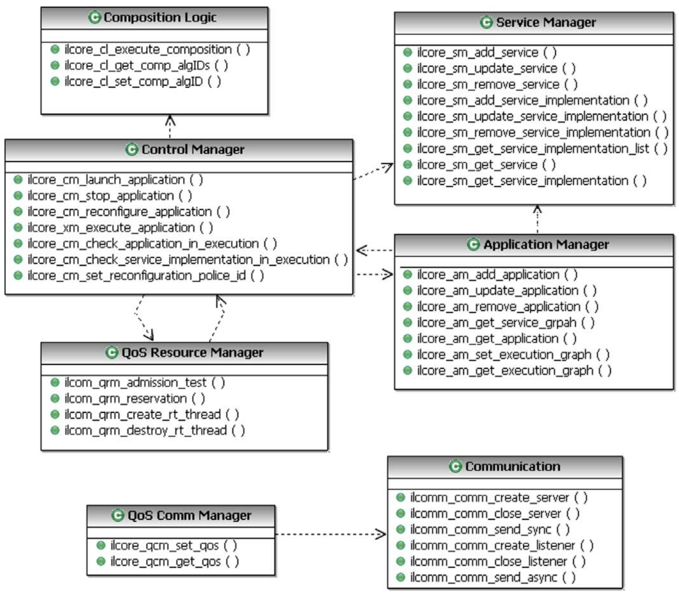

Composition Logic
ilcore_cl_execute_composition ( )
ilcore_cl_get_comp_algIDs ( )
ilcore_cl_set_comp_algID ( )
Service Manager
ilcore_sm_add_service ( )
ilcore_sm_update_service ( )
ilcore_sm_remove_service ( )
ilcore_sm_add_service_implementation ( )
ilcore_sm_update_service_implementation ( )
ilcore_sm_remove_service_implementation ( )
ilcore_sm_get_service_implementation_list ( )
ilcore_sm_get_service ( )
ilcore_sm_get_service_implementation ( )
Control Manager
ilcore_cm_launch_application ( )
ilcore_cm_stop_application ( )
ilcore_cm_reconfigure_application ( )
ilcore_xm_execute_application ( )
ilcore_cm_check_application_in_execution ( )
ilcore_cm_check_service_implementation_in_execution ( )
ilcore_cm_set_reconfiguration_police_id ( )
Application Manager
ilcore_am_add_application ( )
ilcore_am_update_application ( )
ilcore_am_remove_application ( )
ilcore_am_get_service_grpah ( )
ilcore_am_get_application ( )
ilcore_am_set_execution_graph ( )
ilcore_am_get_execution_graph ( )
QoS Resource Manager
ilcom_qrm_admission_test ( )
ilcom_qrm_reservation ( )
ilcom_qrm_create_rt_thread ( )
ilcom_qrm_destroy_rt_thread ( )
Communication
ilcomm_comm_create_server ( )
ilcomm_comm_close_server ( )
ilcomm_comm_send_sync ( )
ilcomm_comm_create_listener ( )
ilcomm_comm_close_listener ( )
ilcomm_comm_send_async ( )
QoS Comm Manager
ilcore_qcm_set_qos ( )
ilcore_qcm_get_qos ( )

Fig. 5. Simplified class diagram.

services, service implementations, and applications, including their QoS characterization.

One of the key components of the middleware is the CM, which coordinates the reconfiguration. The CM component interacts with SM and AM components to obtain information about services, service implementations, and the application itself. It interoperates with the CL component to execute the composition algorithm and with the QoSRM to run the admission control test to determine the resource availability and temporal schedulability of the application. In the case that there is not a feasible solution (the admission test is not passed), the middleware may either renegotiate a lower quality for the application or reduce the quality of other running applications[4] installing the default configuration.

Once the application has been constructed fulfilling the real-time application requirements, the CM component should start the execution of the application by launching the service implementations, thus the application execution flow starts.[5]

The execution is monitored by the QoSRM component; it also enforces the resource budgets assigned to the service implementations according to the resource scheduling policy[6] and guarantees temporal isolation. The QoSRM has a centralized and a distributed part. Local monitoring of resource consumption is performed at each node. One of the nodes acts as the master node that has the overall view of the system state.

During the execution of the application, service implementations communicate with each other using either: 1) the CC component functions since it allows to use QoS communication

[4]Negotiation is based on HOLA-QoS principles with an *a priori* mapping of delivered output quality to resource requirements [28], [20] for service implementations.

[5]Different mode change policies exist (as [27]) that can include some application semantics, i.e., they consider the sequence to stop and launch the required threads (i.e., *service implementations* in our case) depending on the nature and the relative importance of applications.

[6]Budget scheduling is used for assigning resources to service implementations and guaranteeing temporal isolation [19], [20].

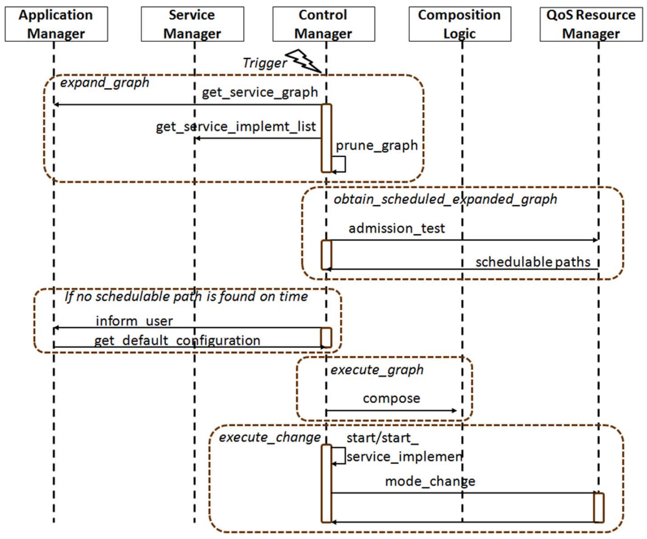


Fig. 6. Component interactions during the reconfiguration.

characteristics of the underlying backbone middleware as, for instance, those provided by DDS, or 2) the CPPS component that enables the usage of a custom real-time network protocol. The communication properties are set with the functions provided by the QS component.

Depending on the computation power of the nodes and the role they play in the system, they will contain a different version of the architecture: *star*, *planet*, or *satellite*. A star node contains the composition logic (the master node), the planet contains all iLAND code except for the composition and reconfiguration control parts. Satellite nodes only contain the basic functionality as the interacting parts in the iLAND network. Satellite nodes run the lightest-weight version of the middleware since some component are not present.

C. Time-Bounded Reconfiguration

Here, we describe the specific functionality of the middleware components and their interactions to achieve time-bounded reconfiguration controlled by the CM (control manager of the CBL layer). The CM detects the reconfiguration triggers that are a user request, an application logic event, or a run-time condition of the system state. The latter is detected by the QoSRM that monitors the execution of the service implementations. Situations requiring reconfiguration can be the failure of a node or of a service implementation, overruns of the resource budgets assigned to some service implementation, or frequent overrun attempts of a given service implementation. The reconfiguration protocol is orchestrated by the CM (see Fig. 6), and it includes internal readjustment of resource assignments to the running entities. If the detected situation cannot be handled by the QoSRM (i.e., there is no schedulable execution graph), this is caused by an erroneous *a priori* study of the system in the initialization phase or by an exceptional execution condition.

Once the reconfiguration is triggered, the CM component expands the application graph (AG). To do this, the CM checks the restrictions introduced by the QoS parameters[7]; consequently, not all existing service implementations are part of the final XG, so the XG is pruned to contain only those service implementations that have passed the compatibility crosschecks. Then, the CM initiates the admission protocol for all possible paths of the XG. Only those paths that are schedulable are extracted from the XG.

The QoSRM contains the algorithms that determine the schedulability of a set of service implementations using real-time theory (utilization based or response time analysis). The XG will be further pruned to be left only with the schedulable paths and leading to a *schedulable expanded graph* or *schedulable graph*. It is worth noting that for the complete reconfiguration process to be time bounded, the duration of the interaction between CM and QoSRM for obtaining the schedulable graphs must be bounded. The initialization phase guarantees that the XG contains schedulable paths that will allow, in the worst case and for short reconfiguration slots, that at least one path can be selected in the first round.

The scheduled graph is then fed to the composition algorithm in the CL component. As a result, one of the scheduled paths is selected that constitutes the *execution graph* (EG), i.e., the application to be executed. The generation of the execution graph follows a QoS criteria (*Q parameter*) that is the *end-to-end response time*. This parameter is the center element for all the validation application domains of iLAND. However, such criteria may be different according to the application needs; in real-time applications, a widely accepted approach is to use *time* values (i.e., end-to-end deadline).

Eventually, the QoSRM will coordinate the launching of the EG through a mode change. If there is no schedulable solution found inside the reconfiguration slot, the system sets the *default configuration* established in the initialization phase.

VI. Experimental Validation

The validation of the architecture has been carried out on a small scale video surveillance application for remote monitoring of industrial processes, implemented and deployed in a distributed environment. Results have been obtained after developing the iLAND reference implementation. The validating scenario contains all of the needed elements and possible deployments of the middleware to validate it. The different nodes of the prototype are given here.

1) Terminal machine that contains a *satellite* version of the middleware acting as the access point for an external operator to interact with the system which, in turn, triggers the reconfiguration process.
2) Controller machine containing a *star* version of the middleware and, therefore, embeds the logic for coordinating the reconfiguration process. It receives reconfiguration triggers from the terminal machine and interacts with all the nodes containing services to reconfigure the system.
3) Service machine with a *planet* version of the middleware. It has the application logic in the form of three different service implementations for three different services.

[7]Functional and non-functional dependency checks across services and service implementations are performed as described in [25].

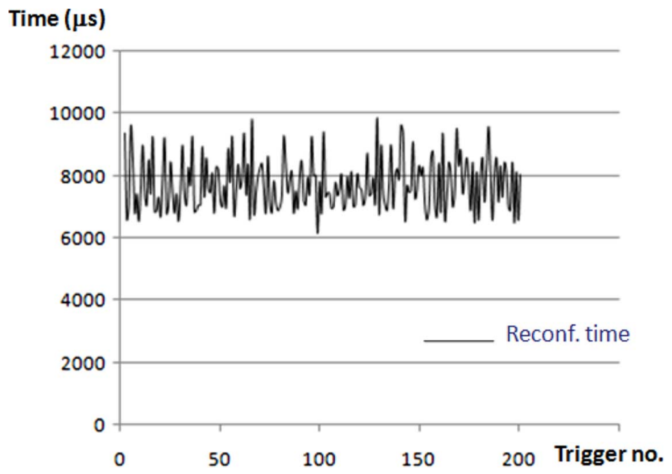


Fig. 7. Execution of an actual video surveillance application.

TABLE I
RECONFIGURATION TIMES

Time	*Value*
Max.	9578 µs
Min.	6530 µs
Average	7753 µs

The network has a 10/100-Mbps Linksys switch. All nodes run a Linux Ubuntu 10.4 distribution with a *real-time patch* that enables the usage of priory-based preemptive scheduling essential for real-time scheduling. The CBL level uses the real-time capabilities of DDS as core communication backbone running the Prismtech implementation v4.5.1.

The demonstrator consists of a full-HD video application delivering a frame rate of 24 fps (frames per second) and having three services: (*i*) *video capture* that uses a camera as input of frames, (*ii*) *video compression*, and (*iii*) *video displays* on screen. The system generates alarms simulating that the monitoring parameters of the physical process has to be changed (e.g., different cameras have to be activated, the compression format has to be changed to increase image resolution, or the system itself has detected an overload and requires nodes to be rebalanced). Video compression is a scalable service therefore it is delivered as different service implementations. The reconfiguration slot is set to 41 ms which is a safe time limit in quality video surveillance. Fig. 7 shows the results obtained for the reconfiguration on a real execution of the system nodes in the presence of interference.

Results show the execution over a representative interval size of 200 reconfiguration triggers showing the efficient behavior of the system. Reconfiguration times are shown in the presence of interference from services and from the control activities of the particular DDS implementation used. In these circumstances, the middleware proves to have a good predictability level and a time-bounded reconfiguration process (as shown in Table I) with an average value of 7753 μs that does not exceed the value set at the initialization phase. The standard deviation is 957 μs that shows that the achieved reconfiguration times are highly stable and suitable for soft real-time environments. Relation between the average and standard deviation times is of 12%. Also, it is remarkable that the internal execution of the middleware proves to be efficient and stable and, therefore, the main deviation in the measurements is derived from the usage of a TCP/IP network with no real-time traffic scheduling that would lead to an increased predictability.

VII. CONCLUSION

This paper has presented the architecture of a middleware that supports time-deterministic reconfiguration in distributed soft real-time environments with a software model based on services. The software architecture has been detailed and justified according to the requirements of the target systems and their temporal properties. With respect to existing middleware solutions that target application composability and dynamics, the presented middleware contributes by including time-bounded reconfiguration and service-based composition algorithms that are built on top of real-time resource management; this supports predictable execution as opposed to other existing solutions. The validation of the middleware architecture has been performed over a real distributed environment that is a reduced scale video surveillance system; this setting can actually be used in hostile environments. Experimental measurements show the validity of the vertical design and development and the stability of the time bounds obtained for the reconfiguration process suiting the needs of distributed soft real-time systems.

REFERENCES

[1] E. Lee, "Cyber physical systems: Design challenges," in *Proc. Int. Symp. Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, Orlando, FL, May 2008, Invited Paper.

[2] B. Bouras and A. Gkamas, "Multimedia transmission with adaptive QoS based on real-time protocols," *J. Commun. Syst.*, no. 16, pp. 225–248, 2003.

[3] A. Dabholkar and A. Gokhale, "An approach to middleware specialization for cyber physical systems," in *Proc. Distrib. Computing Syst. Workshops*, Montreal, QC, Canada, Jun. 2009, pp. 73–79.

[4] I. Estévez Ayres, L. Almeida, M. García-Valls, and P. Basanta-Val, "An architecture to support dynamic service composition in distributed real-time systems," in *Proc 10th IEEE Int. Symp. Object/Component/Service-Oriented Real-Time Distributed Computing, (ISORC)*, Santorini Island, Greece, May 2007, pp. 249–256.

[5] *"Common Object Request Broker Architecture (CORBA) Specification,"* ver. 3.1. Interfaces, OMG, 2008.

[6] *"Distributed Programming With ICE,"* ver. 3.3.1., ZeroC Company, Mar. 2009.

[7] "Data distribution service for real-time systems," OMG, 1.2 formal/07-01-01 edition., Jan. 2007.

[8] P. Basanta Val, M. García Valls, and I. Estévez Ayres, "A dual programming model for distributed real-time Java," *IEEE Trans. Ind. Informat.*, vol. 7, no. 5, pp. 750–758, Nov. 2011.

[9] M. García Valls and F. Ibáñez-Vázquez, "Integrating middleware for timely reconfiguration of distributed soft real-time systems with Ada DSA," in *Reliable Software Technologies—Ada Europe 2012*, Stockholm, Sweden, Jun. 2012, vol. 7308, pp. 35–48, LNCS.

[10] L. Zeng, B. Benatallah, A. Ngu, and M. Dumas, "QoS-aware middleware for web service composition.," *IEEE Trans. Software Eng.*, vol. 30, no. 5, pp. 311–327, May 2004.

[11] F. Esfahani, M. Murad, N. Sulaiman, and N. Udzir, "Adaptable decentralized service oriented architecture," *J. Syst. Software*, vol. 84, pp. 1591–1617, Mar. 2011, Elsevier.

[12] J. Al-Jaroodi, N. Mohamed, H. Jiang, and D. Swanson, "Middleware infrastructure for parallel and distributed programming models in heterogeneous systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 11, pp. 1100–1111, Nov. 2003.

[13] G. Canfora, M. Di Penta, R. Esposito, and M. Villani, "A framework for QoS-aware binding and re-binding of composite web services," *J. Software Syst.*, vol. 81, no. 10, pp. 1754–1769, Oct. 2008.

[14] N. Ben Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas, and V. Issarny, "QoS-aware service composition in dynamic service oriented environments," in *Proc. Int. Middleware Conf.*, Illinois, Dec. 2009, vol. 5896, pp. 123–142.

[15] N. Priyantha, A. Kansal, M. Goraczko, and F. Zhao, "Tiny web services: Design and implementation of interoperable and evolvable sensor networks," in *Proc. 6th ACM Conf. Embedded Sensor Network Syst.*, Washington, DC, 2009, pp. 253–266.

[16] T. Cucinotta, A. Mancina, G. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusina, "A real-time service-oriented architecture for industrial automation," *IEEE Trans. Ind. Informat.*, vol. 5, no. 3, pp. 267–277, Aug. 2009.

[17] I. Estévez-Ayres, M. García-Valls, P. Basanta-Val, and J. Díez-Sánchez, "A hybrid approach for selecting service-based real-time composition algorithms in heterogeneous environments," in *Concurrency and Computation: Practice and Experience*. Amsterdam, The Netherlands: Elsevier, 2011.

[18] I. Estévez-Ayres, P. Basanta-Val, M. García-Valls, J. A. Fisteus, and L. Almeida, "QoS-aware real-time composition algorithms for service-based applications," *IEEE Trans. Ind. Informat.*, vol. 5, no. 3, pp. 278–288, Aug. 2009.

[19] M. García-Valls, P. Basanta-Val, and I. Estévez-Ayres, "Real-time reconfiguration in multimedia systems," *IEEE Trans. Consumer Electron.*, vol. 57, no. 3, pp. 1280–1287, Aug. 2011.

[20] M. García-Valls, A. Alonso, and J. A. de la Puente, "A dual-band priority assignment algorithm for QoS resource management," in *Future Generation Computer Systems*. Amsterdam, The Netherlands: Elsevier, Nov. 2011.

[21] *"OSGi Service Platform Release 4,"* ver. 4.2, OSGi Alliance, May 2011 [Online]. Available: http://www.osgi.org/download/osgi-early-draft-2011-05.pdf

[22] P. Basanta Val, M. García-Valls, and I. Estévez-Ayres, "Enhancing OSGi with real-time Java support," *Software Practice Exper.*, 2012.

[23] G. Coulson, P. Grace, G. Blair, L. Mathy, D. Duce, C. Cooper, W. Yeung, and W. Cai, "A component-based middleware framework for configurable and reconfigurable grid computing," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 8, pp. 865–874, Jul. 2006.

[24] M. García-Valls, I. Rodríguez-López, L. Fernández-Villar, I. Estévez-Ayres, and P. Basanta-Val, "Towards a middleware architecture for deterministic reconfiguration of service-based applications," in *Proc. 15th IEEE Conf. Emerging Technol. Factory Automation*, Bilbao, Spain, Sep. 2010, pp. 1–4.

[25] M. García-Valls, I. Estévez-Ayres, and P. Basanta-Val, "Supporting service composition and real-time execution through characterization of QoS properties," in *Proc. IEEE/ACM Int. Symp. Software Eng. Adaptive and Self-Managing Syst.*, Honolulu, Hawaii, May 2011, pp. 110–117.

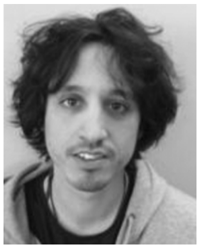
[26] M. García-Valls, P. Basanta Val, and I. Estévez-Ayres, "A component model for homogeneous implementation of reconfigurable service-based distributed real-time applications," in *Proc. IEEE Distrib. Architecture Modeling for Novel Component Based Embedded Systems (DANCE 2010)*, Tozeur, Tunisia, May 29–30, 2010, pp. 267–272.

[27] M. García Valls, I. Estévez Ayres, and P. Basanta Val, "Dynamic priority assignment scheme for contract-based resource management," in *Proc. IEEE Comput. Inf. Technol.*, Bradford, U.K., Jun. 29, 2010, pp. 1987–1994.

[28] M. García Valls, A. Alonso Muñoz, F. Ruíz Martínez, and A. Groba, "An architecture of a QoS resource manager for flexible multimedia embedded systems," in *Proc. Int. Workshop Software Eng. Middleware. LNCS*, 2003, vol. 2596, pp. 36–55.

Marisol García Valls (SM'12) was born in Castellón de la Plana, Spain. She received the Computer Science Engineering degree from Universitat Jaume I de Castellón, Castellón de la Plana, Spain, in 1996, and the Ph.D. degree from the Technical University of Madrid, Madrid, Spain, in 2001.

Currently, she is a Professor with the Telematics Engineering Department, Universidad Carlos III de Madrid, Madrid, Spain. Since 2002, she has been 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 European Project iLAND (ARTEMIS-1-100026) and the Principal Investigator and Coordinator of the Spanish national project REM4VSS (TIN-2011-28339).



Iago Rodríguez López was born in Vigo, Spain. He received the B.S. and M.Sc. degrees in telematics engineering from Universidad Carlos III de Madrid, Madrid, Spain, in 2009 and 2011, respectively, where he is currently working toward the Ph.D. degree.

He is currently involved with in the national project REM4VSS and the European project iLAND and is a Researcher with the Distributed Real Time Systems Lab, Universidad Carlos III de Madrid, Madrid, Spain. His main research interests are real-time systems, middleware technologies, and distributed systems and architectures.

Laura Fernández Villar was born in Madrid, Spain. She received the B.S. degree in audiovisual systems engineering and M.Sc. degree in computer science from Universidad Carlos III de Madrid, Madrid, Spain, in 2009 and 2011, respectively.

She was a member of the Distributed Real-Time Systems Lab, Universidad Carlos III de Madrid, Madrid, Spain, until 2011, where she was a Researcher. Currently, she is working in the private sector, involved with computer science technologies.