A practical solution for functional reconfiguration of real-time service based applications through partial schedulability

Marisol García Valls *and* Pablo Basanta Val
Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid
Leganés, Madrid, Spain
{mvalls, pbasanta}@it.uc3m.es

***Abstract*— Timely reconfiguration in distributed real-time systems is a complex problem with many sides to it ranging from system-wide concerns down to the intrinsic non-robust nature of the specific middleware software and the used programming techniques. In an completely open distributed system, it is not possible to achieve time-deterministic functional reconfiguration; the set of possible target configurations that the system can transition to could be extremely large threatening the temporal predictability of the reconfiguration process. Therefore, a set of bounds and limitations to the structure of systems and to their open nature need to be imposed. In this paper, we present the different sides of the problem of reconfiguration. We provide a solution for timely reconfiguration based on reducing the solution space of solutions of partially closed applications; we have enhanced the logic of a middleware for distributed soft real-time applications with the proposed technique. As a result, applications require a limited number of schedulability tests to search for the valid target configuration. We present some results on the actual reduction of the configuration space achieved by our middleware.**

Keywords-reconfiguration; distributed systems; middleware; real-time

I. INTRODUCTION

Information and communication technologies are rapidly driving the road to an enhanced distributed computing paradigm where public, private, individual, and group's resources can all be put in common to provide enhanced processing power and an ultra connected sensation. In this way, the resources needed and utilized by application users are, in part, hosted remotely and provided from the outside. This is one of the principles of cloud computing that allows to make use of huge amounts of computational and storage resources [1] by means of reduced computing power devices as personal smart phones [2], tablets, etc. Users' applications and data are executed on virtualized resources that are potentially shared with large numbers of other users. At the same time, this brings in interesting green properties as energy savings that are of great importance to private organizations and to the society in general.

The appearance of flexible software paradigms as Service Oriented Architectures (SOA) and efficient communications middleware technologies has been of paramount importance for enabling ultra-connected environments and providing a fertile soil for developing new applications with a flexible structure. Still, support for real-time in such domains has not progressed so fast as the technologies themselves. There are a number of reasons for this. Firstly, real-time research has not advanced at the same speed in the scheduling theory than in the achievement of predictable software-most parts of systems. The latter is still limited specially at the middleware level that is sometimes taken as a black box on top of which performance measurements are obtained. Middleware technologies have not completely eliminated the uncertainty points of such technology as the techniques for serialization, address resolution, transport and Internet level details, etc.

Providing real-time guarantees in distributed computing environments is a hard problem; it becomes extremely hard if these are considered to be open systems with dynamic behavior. However, these are characteristics of the emerging applications; systems undergo changes that may require a modification of their structure to adequately process and react to events from the environment. In this work, we refer to such changes as *reconfigurations*, i.e., it is the process of transitioning from the *current structure* (or *configuration*) of the system to the *target structure*.

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 techniques. A set of bounds and limitations to their structure need to be imposed in order to reduce complexity by means of limiting the size of the space of solutions; that is the number of possible target configurations. In a previous work [3][4], we have identified a set of phases for the complete reconfiguration process which need to be time-bounded. However, we did not further elaborate on the different sides of the reconfiguration problems. In this paper, we provide a high-level view of these, and we show how to reduce the space of solutions by embedding extra logic in the middleware. We provide some validation results that have been obtained on the iLAND middleware reference implementation [3][5].

In this paper, section 2 introduces the problems of software reconfiguration. Section 3 presents the system model. Section 4 describes a middleware-centric approach for reconfiguration by illustrating the coordination scheme and the logic for reduction of the space of configurations, and it validates the presented concepts. Section 5 draws some conclusions.

II. EXPOSING THE PROBLEMS OF *SOFTWARE* RECONFIGURATION

Supporting dynamic behavior in modern real-time applications opens up new possibilities for providing more powerful and added value to final users. Achieving dynamic behavior in real-time, however, challenges temporal predictability at the different architectural layers, from system-wide design down to the operating system and I/O drivers concerns. Still, in some application domains with soft real-time requirements (e.g., intelligent video surveillance) high utility levels can be achieved by imposing a set of bounds and limitations to the actual dynamics.

We have investigated in different sides of software reconfiguration in distributed systems with real-time requirements. In this paper, we provide an approach to this problem based on single-threaded services that enable application dynamics. Following, a general description of the challenges and approaches to reconfiguration is provided.

A. Challenges to reconfiguration

In most current distributed applications, achieving real-time behavior is an evident added-value; some of these applications have different degrees of tolerance to deadline misses resulting only in a degraded but acceptable operation (e.g., real-time video processing in intelligent co-operative nodes). The latter are, then, referred to as soft real-time systems. Specially current distributed soft real-time systems must be designed in order to react to the occurrence of events that may require some adaptation (or reconfiguration) of the system in terms of: (i) *application/functional structure* where some parts of the functionality may be removed, replaced, or newly added, or (ii) *internal processing configuration* where the same functionality may continue to be provided but adaptation of the processing parameters is performed. An example of a reconfiguration event is one generated by an unexpected movement in some high-security perimeter as a sign of a possible intrusion detected by a sensor; it may require that a different camera set be activated instantaneously and higher resolution images are captured and sent at the same time.

Reconfiguration events can be triggered either *internally*, i.e., due to the self monitoring, or *externally*, i.e., requested by an outside entity, e.g., the user. Moreover, reconfiguration triggers may arrive:

- *Synchronously*; at a specified time when the system is able to process it.
- *Asynchronously*; with an unknown arrival pattern and modeled as a periodic task with a release time greater than zero.

To adequately process reconfiguration events, real-time systems require that the transition either to a new functional structure or a new internal processing configuration is time bounded. There are different important threats to preserving temporal predictability in the presence of dynamic behavior such as software reconfiguration:

- **Dynamic versus static entity membership**. There are two types of systems: *static size* system and *dynamic size* systems. The former does not allow any modification of the system services at run-time, whereas the latter does allow the spontaneous appearance of new active software units in the system at any time.
- **Size of the *space of solutions***. Upon a reconfiguration event, the reconfiguration protocol logic decides on the target system configuration to make the transition to. There is a number of possible target configurations, named *space of solutions*, where the final target configuration is selected from. For a system of dynamic size, the space of solutions can be unlimited; so, there is no feasible solution to derive a time-bounded reconfiguration strategy.
- **Network effects and schedulability**. In a distributed system, the distributed coordination of the entities may incur in network delays. There are some solutions that bound this problem as the real-time traffic scheduling and the synchronous communications [6]. Also, there are some approaches as the worm whole [7]**¡Error! No se encuentra el origen de la referencia.** that allot space for both synchronous and asynchronous traffic. The existence of central entities for coordination also limits the unpredictability of the distributed real-time transition.
- **Virtual platforms**. In virtualized execution environments, hardware is buried; some parts of the application code do not have direct access to the hardware. Temporal predictability is threatened, and it depends on the existence of real-time hypervisors that capture and transform the interrupts into software ones with appropriate priorization; this is a basic tool to prioritize the execution among different virtual machines and, in the end, applications. Such environments can be modeled by means of *virtual resources* and *virtual processors*.

B. Approaches to reconfiguration

Researchers tackle the problem of reconfigurations from different points of view and backgrounds. Mainly, we find:

- *Software-based* versus *system-level* schemes. Software-based approaches focus on the specific programming-level details as the software units (e.g. objects, components, etc.) and their connectors and bindings to make a transition to a new state. On the other hand, system-level approaches take a more abstract view dealing with a formalized system model and the algorithms for selecting the target configurations that meet the given system-level properties (e.g., timeliness, quality values, or interface functionality).
- *Non real-time* versus *real-time schemes*. Non real-time schemes work basically on efficient transition models, but timeliness is not a primary issue; sometimes, it is not considered at all. In real-time reconfiguration schemes, the system must be temporally predictable not only during normal operation but also during reconfiguration.
- *Centralized* versus *distributed* schemes. A centralized perspective does not take into consideration the network effects in the coordination of the reconfiguration; whereas the distributed reconfiguration must account for the schedulability model of the network, the effect of

possible communication delays, and the distributed coordination problems.

- *Closed* versus *open* schemes. In closed schemes, all possible system states and transitions are known a priori; therefore the reconfiguration is performed at a given time or synchronization point (i.e., when the system is in a safe state). This is typical of mode changes in hard real-time systems. In open reconfiguration schemes, all possible system states are not known; therefore, extra logic is needed to select the target system configuration or state.
- *Operational or task-level* versus *functional or system-level.* Operational deals with the low-level details of the run-time execution and schedulability. It deals with the compatibility of the execution model of tasks and the required adjustments to their parameters (e.g., change of activation period, priority, execution time refinements, etc.). System level strategies deal with the changes in the functional units from a system wide perspective and how to replace these units by searching for new options that keep the system run-time requirements.

III. SYSTEM MODEL

In our model, we consider that *applications* are extended functionality made by the aggregation of a set of services. *Services* are self contained software functionality pieces that are realized by concrete and specific service implementations. A *service implementation* is an active software entity that provides the functionality of the specific service. Service implementations communicate among themselves only via message exchanges.

An application a_i has different possible realizations, $a_{i,j}$, $\{a_{i,j}: j=1,2,...\}$ or internal structure possibilities. In the same way, a service s_i can have different realizations $\{s_{i,k}: k=1,2,...\}$. Also, a_i has a specific set of services $S_i=\{s_j: j=1,2,...\}$. Therefore, a given application configuration $a_{i,j}$ is made of a set of service implementations $a_{i,j}=\{ s_{k,l}: k=1,2,...; l=1,2,...\}$.

As a result, a reconfiguration is the replacement, removal, or addition of one or more service implementations. *Reconfiguration* refers to a change in the structure of the active software units that are part of an application. In general, a service can contain a number of threads or tasks; in our model, we consider single-task services. A reconfigurable system or application a_i is, therefore, a superset of n service implementations; in the end, $a_i = \{ \tau_1, \tau_2, ..., \tau_n \}$.

A reconfiguration occurs when the system or application configuration $a_{i,j}$ is replaced by another one $a_{i,k}$, such that the target service implementation set is not equal to the orginal $S_{k,l}^{init} <> S_{k,,l}^{target}$. Here, we use the terms *system* and *application* interchangeably.

In a real-time environment, reconfigurations have to be performed in a time bounded fashion. Therefore, the time (t^{re}) taken to complete the transition from $S_{k,l}^{init}$ to $S_{k,,l}^{target}$ has to be bounded.

In the specific case of a real-time distributed system, we must not neglect the effect of the operating system and middleware threads. In our model, we account for such a structure. Therefore, a system state is made of a set of threads that are scheduled by the resource manager of the operating system. The threads of the initial state are:

- $as_i=\{\tau_i: i=1,...,n\}$; where n is the number of service implementations that are running. This set is the "payload" of the system; these tasks are functionality providers for users.
- $os_i=\{\tau_j: i=1,...,m\}$; where m is the number of operating system tasks that are providing the basic execution services and environment. This is the actual run-time services.
- $mw_i=\{\tau_z: z=1,...,p\}$; where p is the number of middleware tasks that are actively providing the communications logic and the extended functionality that is typically contained in it (e.g. thread pools).

A service implementation $s_{i,k}$ is specified by its functionality f_i (all $s_{i,k}$ have a common f_i), it execution time $C_{i,k}$, its release period T_i in our real-time model (see [9] execution priority $P_{i,k}$, the response deadline $D_{i,k}$, the quality value it offers $Q_{i,k}$ (an application-related utility value), and the Δ is the dependency list with respect to other service implementations.

Services are also considered to be activated by periodic messages, and they can be scheduled according to a utilization based model [10] that also accounts for the operating system and middleware tasks. Figure 1 shows the expression for calculation of the utilization assuming that tasks are independent. This is in practice a limitation.

$$\sum_{i \in as} \frac{C_i}{T_i} + \sum_{j \in os} \frac{C_j}{T_j} + \sum_{k \in mw} \frac{C_k}{T_k}$$

Fig. 1. Utilization based model

All tasks in our model are contemplated in the schedulability analysis of the system. Therefore, the whole task set to be considered in the analysis is $\{as_i, os_i, mw_i\}$ with different priority level values that determine the pre-emption and execution mode privileges. Priority assignment also plays a major role; we use a simple priority assignment technique based on priority bands [11].

IV. A MIDDLEWARE-BASED APPROACH

One of the approaches to reconfiguring a distributed real-time system consists of including extra logic in the middleware to undertake the needed operations for adapting to the environment, correcting execution time problems, or adjusting to a controlled and limited level of changing user requirements.

Besides including the basic real-time communication facilities for timely message exchange among services (e.g. provided by DDS or RT-CORBA among others), middleware supporting real-time reconfiguration must include logic for:

- *Detection of reconfiguration events*. The middleware should recognize what is a source of reconfiguration;

false positives should be avoided since it may cause instability.

- *Selection the target system configuration*. Among all possible target configurations, the middleware should decide on a feasible one that meets the application requirements and it is schedulable.
- *Transitioning to the target system configuration*, i.e, the sequence of steps to perform a run-time transition to the target configuration in bounded-time.

iLAND follows a system-level reconfiguration scheme; transitions are not completely closed, and a controlled degree of freedom is left for application transitions.

A. *iLAND approach*

iLAND [3][5] follows the classical principles of a layered middleware (see figure 2) with enhanced logic for supporting reconfiguration of distributed service-based real-time applications. The *Core Functionality Layer (CFL)* contains most of the key added-value functionality (management of services, and time-deterministic composition and reconfiguration). Communications are timely with two different levels of temporal predictability:

- Real-time communications are supported by defining the complete network protocol stack, i.e., time-triggered level 2 media access control networks that enable network traffic scheduling.
- QoS-aware communications are supported through the usage of an underlying DDS communication backbone. In fact, any other asynchronous (publish-subscribe) or synchronous backbone middleware can be used since iLAND defines a *common communication bridge* for adjusting to communication paradigms that is provided inside the *Communication Backbone and Resource Management Layer (CBL)*.

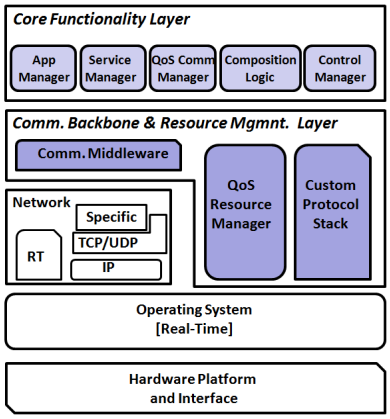

Fig. 2. iLAND middleware architecture

The *Communication Backbone and Resource Management Layer (CBL)* also contain the *QoS Resource Manager* component aimed at QoS-based resource management that arbitrates resource assignment among the different service implementations (i.e., threads) of the system. Scheduling for multi-resource management lies inside this component that follows the HOLA-QoS [12] architecture and budget scheduling [11].

The *Control Manager* component (or *ReCoM - Reconfiguration Control Manager*) contains the logic for coordinating the transition upon a reconfiguration event is detected; it performs high-level monitoring of the application, and it controls the reconfiguration times and sequence to be time-deterministic [4]. Reconfiguration time slots are scheduled as part of the overall timing analysis of the system [9], and real-time dynamic resource management algorithms based on dynamic priorities are embedded in this middleware component [10].

B. *Reducing the complexity*

It is impossible to achieve real-time behavior in the reconfiguration of a completely open distributed real-time. In iLAND, we require that systems are checked a priori to determine the size of the solution space and schedulability of their solutions. This is done by means of a priori system profiling tools [4][5] that allow the pre-visualization of the different possible system states and transitions.

Figure 3 illustrates the high-level view of the reconfiguration logic aiming at reducing the complexity of the solution space.

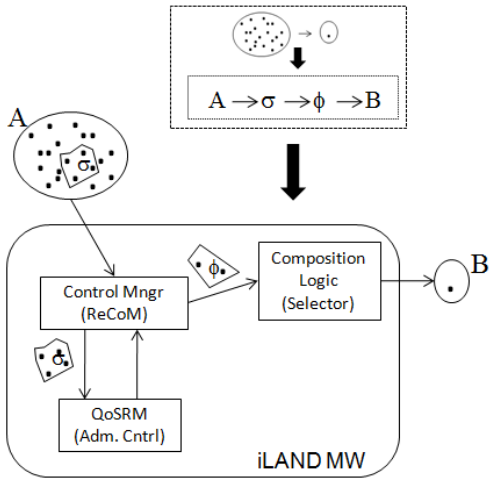

Fig. 3. Main components that participate in the reconfiguration process

We are able to perform a prune of the *A* set, i.e., the complete space of solutions, to obtain at the end the B set, i.e., the selected target configuration. The control manager coordinates the process in the following way shown in algorithm 1.

There are two mandatory operations for obtaining the target configuration to transition to:

- *Construction of the possible target configurations* done with respect to a quality value that depends on the application.
- Temporal check of the possible target configurations applying schedulability analysis.

The search and checking of the solutions of *A* can be done in two ways:

- *Exhaustive schedulability* check by checking of all solutions to determine if they are schedulable.
- *Partial schedulabiltiy* test of a subset of solutions. It is a hybrid model where the solution space is first reduced in size, and then a schedulability test is applied. Therefore, first a subset of A is selected/obtained and then, it passes the schedulability test. This uncouples the reduction of the space of solutions from the schedulability analysis technique used.

Algorithm 1. Reconfiguration logic

Require: q^{sys} is the minimum quality of the target state
1: **for all** $i \in$ A **do**
2: **if** representative(*i,* A) **== true**
3: $i \in A^R$
4: $\sigma = A^R$
5: **for all** $j \in \sigma$ **do**
6: **if** schedulable(*j*) **== true**
7: $j \in \sigma^S$
8: $\phi = \sigma^S$
9: **for all** $k \in \phi$ **do**
10: **if** check_quality(*k*) $\geq q^{sys}$
11: B = { q^{sys} }
12: return **true**
13: **else**
14: B = Ø

For time-bounded reconfiguration, it is not necessary to search for the optimum solution. Instead, we aim at finding one feasible solution that is valid, i.e., it is schedulable and close to the application-desired quality value. Therefore, it is not necessary to check the feasibility of all possible solutions or target configurations. Moreover, undertaking a schedulability analysis of the complete *A* set (the global solution space) can be highly inefficient specially depending on the technique that is used; a utilization-based algorithm will typically yield a better performance than response time analysis, depending on the number of tasks and the values of their real-time parameters.

As a consequence, we extract a representative subset of *A* containing a variety of solutions that are a valid summary of *A*. This is performed by function *representative(i,A)* that decides whether solution *i* (or target state *i*) is a representative solution of the whole space of solutions of *A*. With all selected *i* solutions, we create subset σ.

Results have been collected to compare between different inputs. In table 1, we present experimental results collected to compare the reduction of the initial solution space, *A*. We shown different complexities of *A*.

Table 1*. Experimental results for the reduction of the solution space*

Service Implementations [s_1,s_2,s_3,s_4]	***Sets***			***Red. factor (%)***
	A	**σ**	**ϕ**	
[3,3,3,3]	81	8	4	95,06
[6,5,4,7]	840	18	12	98,57
[3,3,3,3]	81	8	4	95,06
[6,5,4,7]	840	12	8	99,05

For the four different states of the space reduction process, table 1 shows the number of solutions that are contained in the different sets. *A* is the initial space of solutions, σ is achieved after the pruning of the initial *A* and keeping only the solutions that are representative of the complete *A* set. This process is a complex algorithmic development that has been developed in iLAND middleware and that is out of the present work. ϕ is obtained after the QoSRM performs the admission test.

These initial results assume a negligible effect of the operating systems and middleware tasks. Therefore, it is evidenced how the reduction factor achieved by the reduction process is considerably high (above 90%) for all cases, and over 98% for all cases of a graph with four initial services that contains over 840 solutions at the A stage. The reduction factor describes the number of solutions that are eliminated from the original A set. The main reduction of the number of solutions happens in two important algorithmic steps that are based on the placement of service implementations in the different nodes of the distributed systems and the distance to the desired QoS value of the application.

For all cases, the final ϕ set contains over 90% less solutions than those contained in the A set; as a consequence, the number of possible solutions sent to the admission control for schedulability analysis is dramatically decreased. Therefore, it shows the efficiency of the whole idea and, as the ultimate goal, of the reconfiguration of the system.

V. RELATED WORK

In distributed systems, we currently face two types of reconfiguration approaches: those centered at the network level providing traffic scheduling integrated with the task scheduling at the nodes or real-time middleware approaches that offer a higher-level view of the many problems faced by the communication software that tries to provide real-time interaction for distributed applications.

In distributed systems, the local and remote tasks as well as the messages over the network must be properly scheduled and synchronized to meet the deadlines of the application. This is not a simple task since the future arrivals of tasks and requests is typically not known in an complex open system. Some approaches only consider synchronous communication real-time networks as [6] that eases the schedulability analysis cost.

Dynamic scheduling was also addressed by some contributions that aimed at providing guarantees to the distributed processes communicating via synchronous primitives, combining off-line and on-line scheduling [13].

New applications have appeared over the years introducing more dynamic requirements. New solutions have provided resource management techniques for adapting the resource assignments to applications tasks according to their instant computation needs [14][15].

The middleware level has also been improved by the addition of efficient characteristics at very specific levels. As an example, distributed protocols executing in unpredictable environments as the Internet or ambient computing, have shown their need for dynamically adapting to environment changes while preserving the Quality of Service (QoS)

[15][16]. Some solutions have also appeared from the dependable computing community such as [17][18].

One of the drawbacks perceived by the real-time community with respect to middleware is that it decreases the robustness of the system. Often middleware is seen as a black box over which only performance tests are carried out. However, the appearance of middleware technologies such as DDS, ICE, or RT-CORBA, etc., have provided a powerful, flexible, and versatile development push. One of most interesting characteristic is the abstraction of the network details to the business-level logic. However, when providing real-time guarantees, programmers must specify the required application properties in the pertinent middleware hooks that must be traceable across the software layers.

The problem of combining distribution middleware and real-time is even harder if we try to support dynamic reconfiguration. Some work is already available although it typically targets mainstream distributed systems. For instance, in [15], an algorithm for dynamic reconfiguration of applications is provided that keeps the structural integrity and system states consistency, but it is silent about timeliness. Other approaches, mainly focus on the reconfiguration of the software components at the level of binding and re-binding of the component connectors such as [16]. More recently, the iLAND middleware [3][5] has appeared as a framework to support time-bounded dynamic functional reconfiguration of distributed real-time applications. In this paper, we present additional reconfiguration issues and problems to the time-bounded property related to the reduction of the space of solutions.

VI. CONCLUSIONS

We have presented the different sides of the functional reconfiguration problems for distributed real-time systems that are partially open. In this paper, we have described a system model and approach based on the integration of reconfiguration coordination logic in an enhanced communications middleware. We describe a solution to the problem of computational complexity due to large solution spaces; we provide a high-level view on how to reduce the space of solutions by selecting only a representative subset of configurations; as a result, a decrease in the number of schedulability analysis is performed. Finally, we present empirical results on a real implementation of iLAND middleware reference implementation that validate the proposed ideas.

ACKNOWLEDGEMENTS

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

REFERENCES

[1] D. Williams, E. Elnikety, M. Eldehiry, H. Jamjoom, H. Huang, and H. Weatherspoon. "Unshakle the Cloud! "Appears in *Proceedings of the 3rd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, June 2011, Portland, CA.

[2] J. Andrus, C. Dall, A. Van't Hof, O. Laadan, J. Nieh. "Cells: A Virtual Mobile Smartphone Architecture". In Proc. of 23rd ACM Symposium on Operating Systems Principles (SOSP'11). Cascais, Portugal. October 2011.

[3] M. García-Valls, I. Rodríguez-López, L. Fernández-Villar. "iLAND: An Enhanced Middleware for Real-Time Reconfiguration of Service Oriented Distributed Real-Time Systems". IEEE Transactions on Industrial Informatics, DOI 10.1109/TII.2012.2198662 Accepted for publication. 2012.

[4] M. García-Valls, R. Castro-Fernández, I. Estévez-Ayres, P. Basanta-Val, I. Rodríguez-López. "A Time-Bounded Service Composition Algorithm for Service Based Distributed Real-Time Systems". In IEEE Conference on Embedded Software and Systems. Liverpool, UK. June 2012.

[5] iLAND project. "iLAND Reference Implementation Installation & User Guide". http://sourceforge.net/projects/iland-project/ Available on-line. September 2012.

[6] H. Kopetz, G. Bauer. "The Time-Triggered Architecture". Proceedings of the IEEE, vol. 91(1), pp. 112-126. 2002.

[7] P. Verissimo. "Travelling through wormholes: a new look at distributed systems models". SIGACT News, vol.37(1), pp. 66-81. 2006.

[8] A. Nogueira, M. Calha. "Predictability and efficiency in contemporary Hard RTOS for multiprocessor systems". In Proc. of 17th IEEE Int'l Conference on Embedded and Real-Time Computing Systems and Applications. Japan, 2011.

[9] M. García-Valls, P. Basanta-Val, P., I. Estévez-Ayres. "Real-time reconfiguration in Multimedia Embedded Systems". IEEE Transactions on Consumer Electronics, Vol. 57, No. 3, pp. 1280-1287. August 2011.

[10] C. L. Liu and J. W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment". Journal of the Association of Computing Machinery, vol. 20(1), pp. 46-61. January 1973.

[11] M. García-Valls, A. Alonso, and J.A. de la Puente. "A Dual-Band Priority Assignment Algorithm for QoS Resource Management". Future Generation Computer Systems, vol. 28(6), pp. 902-912. June 2012.

[12] M. García Valls, A. Alonso Muñoz, F. Ruíz Martínez, A. Groba. "An Architecture of a QoS Resource Manager for Flexible Multimedia Embedded Systems". In Proc. of Int'l Workshop on Software Engineering and Middleware. LNCS vol. 2596. 2003.

[13] M. Di Natale, J. A. Stankovic. "Dynamic End-to-End Guarantees in Distributed Real-Time Systems". In Proc. of IEEE Real-Time Systems Symposium. Dec. 1994.

[14] L. Palopoli, T. Cucinotta, G. Liperi, G. Buttazzo. "AQuoSA - Adaptive Quality of Service Architecture". Software: Practice and Experience.

[15] A. Rasche and A. Polze. "ReDAC - Dynamic Reconfiguration of distributed component-based applications with cyclic dependencies". In Proc. of the 11th IEEE Int'l Symposium on Object and Component-oriented Real-Time Distributed Computing (ISORC), pp. 322-330. Orlando (FL), USA. May 2008.

[16] V. Vanneschi, L. Veraldi. "Dynamicity in distributed applications: issues, problems, and the ASSIST approach". Parallel Computing, vol. 33(12), pp. 822-845, Elsevier. December 2007.

[17] M. Dixit, A. Casimiro, P. Lollini, A. Bondavalli, and P. Verissimo. "Adaptare: Supporting automatic and dependable adaptation in dynamic environments", ACM Transactions on Autonomous and Adaptive Systems". 2011.

[18] A. Kangarlou, S. Gamage, R. Kompella, and D. Xu. vSnoop: Improving TCP throughput in virtualized environments via acknowledgement offload. In Proc. of ACM/IEEE SC. New Orleans, USA. Nov. 2010.