Towards a Reconfiguration Service for Distributed Real-Time Java

P. Basanta-Val, and M. García-Valls
Dpto. de Ingeniería de Telemática
Universidad Carlos III de Madrid
Avda. Universidad nº 30, Leganés- Madrid (Spain) 28911
{pbasanta, mvalls}@it.uc3m.es

***Abstract*—Ancient monolithic distributed systems were attached to well-known development practices and offline analysis. Current scenarios are more dynamic, and open, plenty of applications and services which appear and disappear dynamically at runtime. Likewise, these scenarios require taking into account actions that were traditionally addressed offline, this time in an online scenario. This paper contributes a reconfiguration service in the context of distributed real-time Java application as a means to include real-time reconfiguration into next generation real-time Java systems. The paper addresses the integration taking into account changes required in the API and the cost of some reconfiguration strategies.**

Keywords- DRTSJ, real-time Java, real-time middleware, predictable reconfiguration.

I. INTRODUCTION

Many real-time infrastructures consider admission control as an offline feature [1, 2][3]. Using accurate analysis tools, (real-time) systems engineers analyze system requirements executing feasibility tests before deploying their applications. These algorithms check if all tasks of the system get satisfied their requirements and generate also a static configuration that can be deployed on top of real-time operating systems and/or real-time middleware. However, many applications are much more dynamic and may benefit from a on-line scheduling analysis integrated within its core ([4][5][6][7] [8]). New nodes and tasks may appear and disappear dynamically without deadline misses. Also, there is an increased need for adaptation and reconfiguration.

This paper addresses a real-time reconfiguration service for distributed real-time Java. Its goal is to include algorithms able to modify the behavior of a whole distributed application by adding and removing groups of tasks. The reconfiguration service proposed is also of interest for the DRTSJ (i.e. the Distributed Real-Time Specification for Java) community (e.g. [9][6][10][11][12][13]) that may include this new service within its core. Currently, DRSTJ is silent on real-time reconfiguration issues.

The reconfiguration service proposed in this paper is similar to the reconfiguration strategies proposed in [5] [8] for the iLAND middleware. Both approaches aim at producing online admission control for predictable end-to-end response-times. However, their contexts and goals are rather different. While iLAND [14] is described in the context of SOAs (Service Oriented Architectures) with multiple service implementations for C-applications; the proposed service is focused on distributed real-time Java applications.

Thus, the goal of the service in distributed real-time Java is to produce a *common framework* useful for real-time {re}configuration. The rest of the paper describes the approach and evaluates a simple {re}configuration strategy. Section 2 describes the context of the reconfiguration service. Section 3 relates its performance on a networked infrastructure. Section 4 connects the work to the state of the art. Finally, Section 5 introduces conclusions and outlines related work.

II. RECONFIGURATION SERVICE

The service proposed is described for DREQUIEMI a framework for distributed real-time Java, and its particular API ([11]). The extensions and techniques described in this section may be included without major changes into DRTSJ (The Distributed Specification for Real-time Java).

A. Distributed Real-Time Java middleware

DREQUIEMI (Figure 1 and Table I) divides the system distribution infrastructure into a set of layers that contribute to hide distribution issues. The model identifies three main resources that may be managed (namely: memory, CPU, and the network). These services are accessed via an infrastructure middleware (which could be supported via a real-time Java virtual machine) which accesses these resources. On top of the infrastructure middleware, two new layers (distribution and common services) are in charge of providing standard services for distribution. The list includes stub/skeleton in charge of real-time invocations ([15] [16]); DGC in charge of distributed garbage collection, naming for a white page service; and synch events for end-to-end event models ([17][18]). The common services use three main pools (connection, thread, and memory area-pools [19]) to parameterize the behavior of these services. Globally, resource allocation is managed by three managers: distributed memory manager, processor manager and distributed connection manager. On top of common services, there is application subsystem which divides applications into a set of reusable components (currently addressed with OSGi technology [20][21]).

In DREQUIEMI, there are three natural reconfiguration points. The first is placed at the infrastructure level, as a means to control locally memory processor and network. The second extends this control to a distributed scheme and it is placed at the distributed

manager which controls distributed applications performance. The last level of control is at component level performing adaptations taking into account the components of an application.

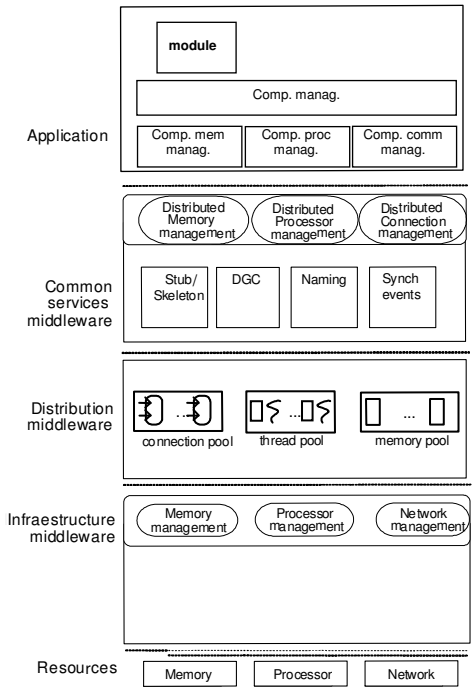


Figure 1. DREQUIEMI's architecture

TABLE I. QOS PARAMETERS FOR DISTRIBUTION MIDDLEWARE IN DREQUIEMI

QoS parameters in distribution layer	**Meaning in DREQUIEMI**
Scheduling Parameters «*Priority*»	Priority used during the up-call at the server
Release Parameters «*Release Time, Period, Deadline, and Cost*»	Invocation pattern at the server according to the real-time scheduling theory
Processing Group Parameters «*Release Time, Period, Deadline and Cost*»	Invocation pattern at the server according to the real-time scheduling theory
Memory Parameters	Parameters used by the garbage collector at the server
Thread Pool	Thread pool used at the sever to manage remote invocations
Connection Pool	Connection pool used at the client to carry out the remote invocation
Memory Area Pool	Memory area pool used at the server to accept remote invocations
port	The IP port on which the remote object is accepting incoming messages
RemoteStub	The stub with the remote reference to the remove object
EventCommonInt.	A remote object which allows subscriptions and may be remotely triggered.

The service analyzed in this section is at distribution level, it manages adaptation at distribution level by controlling resource in all distributed nodes. Internally, it uses the resource manager included in each node. It also offers support to the component manager defined at the application level of DREQUIEMI.

B. Extension for Reconfiguration Service

Figure 2 shows the API changes introduced to accommodate the reconfiguration service in the DREQUIEMI ecosystem, which includes new APIs for centralized and distributed real-time Java.

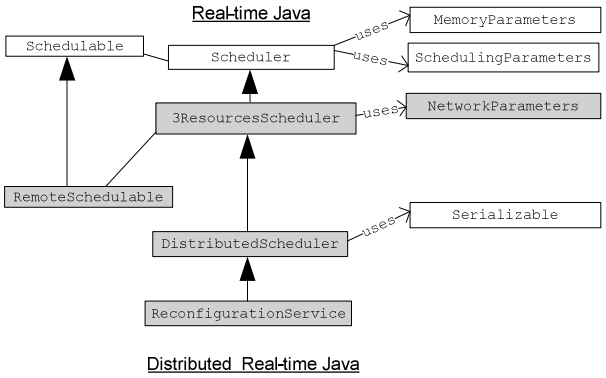


Figure 2. New classes proposed for the reconfiguration service

In centralized Java, the main extension required is network support. Currently RTSJ [22], the main approach in real-time Java, does not take into account the existence of different types of networks. In the support proposed for reconfiguration two new elements are included to introduce the network into the system. The first is a characterization for networks similar to memory parameters and scheduling parameters currently included in RTSJ via a tagging interface (`NetworkParameters`). All network classes include a tagging interface that extends scheduling parameters list included in Table I with network parameters. Network parameters are domain dependant and may consist of a simple non preemptive priority for the router. (e.g., in [23] the authors used a non-preemptive model with network release times, and scheduling parameters to model a prioritized IP-based Switched-Ethernet.) The possibility of including additional networks is added via a new scheduler template named `3ResourcesScheduler`.

Then, the distributed scheduler API extends the API scheduler interface of RTSJ to be accessible as a remote object via DRTSJ. This simple mechanism is enough to allow the allocation of schedulable entities from another remote node. The code for the extension is provided in Listing 1. In all this code all the parameters of the remote object have to be *serializable* in order to be able to be transferred via the network. (Note: a way to transfer this non-functional information among in distributed remote invocations is described addressed in [24] [25]).

From the point of view of the definition of a reconfiguration service (`ReconfigurationService` in Listing 2), the approach analyzed in this article is simple.

It is based on modeling the distributed scheduler (described in Listing 1) as an RTSJ' schedulable object. This simple approach enables to define generic real-time parameters for the reconfiguration service.

```
01: public interface DistributedScheduler
02:    extends java.rmi.Remote {
03:    boolean addIfFeassible(
04:            Schedulable s,
05:            ReleaseParameters rs,
06:            MemoryParameters mp,
07:            ProcessingGroupParameters pgp,
08:            NetworkParameter np )
09:           throws RemoteException;
10:    boolean removeIfFeassible(Schedulable s)
11:           throws RemoteException;
12:}
```

Listing 1. API interface for the distributed scheduler

```
01: public interface ReconfigurationService extends
02   DistributedScheduler, Schedulable{
03:     MemoryParameters   getMemoryParameters()
04:           throws RemoteException;
05:     ProcessingGroupParameters
06:          getProcessingGroupParameters()
07:           throws RemoteException;
08:     ReleaseParameters
09:          getReleaseParameters()
10:           throws RemoteException;
11:     Scheduler getScheduler();
12:           throws RemoteException;
13:     SchedulingParameters getSchedulingParameters();
14:           throws RemoteException;
15:     NetworkParameters getNetworkParameters();
16:           throws RemoteException;
17:     void  setMemoryParameters(MemoryParameters mp)
18:           throws RemoteException;
19:     void  setProcessingGroupParameters(
20:                 ProcessingGroupParameters pgp)
21:           throws RemoteException;
22:     void  setReleaseParameters(
23:                 ReleaseParameters rp)
24:           throws RemoteException;
25:     void  setScheduler(
26:                 Scheduler sch)
27:           throws RemoteException;
28:     void  setSchedulingParameters(
29:                 SchedulingParameters sp)
30:           throws RemoteException;
31:  }
```

Listing 2. API interface for the reconfiguration service.

By using this approach, the API of the reconfiguration could be bounded time. It offers mechanisms to limit declare the amount of CPU, memory, bandwidth in the network required for the reconfiguration process. Furthermore, the reconfiguration process could be globally modeled as a *periodic*, or *sporadic* process which may scheduled with the remaining tasks in each remote node.

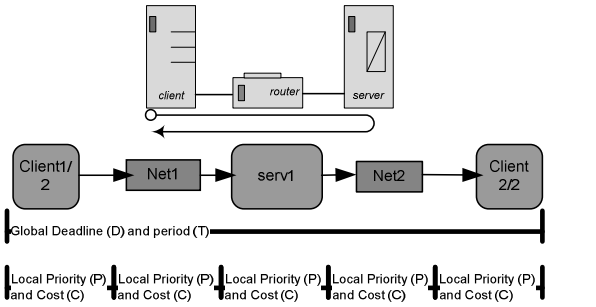
Lastly, notice that by default there is not a dominant reconfiguration algorithm or technique. This is a distinctive feature of the scheduler of RTSJ too. In both cases the idea is that different algorithms may extend the basic tasking template to include system requirements. A developer may also extend the basic reconfiguration policy to take into account different application domains.

C. *End-2-End Flow-Shop Reconfiguration Use-Case*

The use case, which is empirically explored in the evaluation of Section III, is the simple end-to-end flow-shop model used by Sun [26], Tindell [27], and Palencia [28]. In this model, an end-to-end restriction is modeled (see Figure 3) as a sequence of subtasks that execute in order (i.e. with precedence constraints). Each end-to-end transaction ($\Gamma^j \mid j \in 1..M$) is defined as follows:

- Global deadline (D^j)
- A global period (T^j)
- A set of ${}^j{}_n$ schedulable segments $\{S^j{}_1 \rightarrow \ldots \rightarrow S^j{}_n\}$
    - A local execution priority ($P^j{}_n$) for each end-to-end transaction segment ($S^j{}_n$).
    - A local worst-case execution time ($C^j{}_n$) for each segment ($S^j{}_n$).

In addition, each node runs a periodic enforcer [29] that allows analyzing each flux in a node as it was local.




```
01: interface FlowShopSchedulables extends Schedulable{
02:  Vector getAllShedulable(); //It returns all sch.
03: }
```

```
01: interface  PriorityBasedReconfigurationService
02: extends ReconfigurationService{
03:}
```

Figure 3. Reconfigurable Priority-Based End-To-End transactions classes for the reconfiguration service. It consists of five schedulable segments.

III. EMPIRICAL EVIDENCES

An empirical evaluation of the mechanism described in Section II was carried out. The goal pursued in this

evaluation was to obtain a reference performance for the system. This performance has been evaluated on a networked infrastructure. The evaluated application consists of N Java nodes (Figure 4) that host distributed applications. There is a single local area network (LAN) supported with an IP-router that connects nodes one each other. The router enforces a priority driven policy via end-to-end communications.

In the example of Figure 4, the application consists of two end-to-end transactions (`A1` and `A2`). `A1` is made of five consecutive schedulable segments (one local in node 1, another two for the network and another at node 3). `A2` is allocated in Node1. All updates are done via the manager node which adds locally the two applications and transfers this information to the required nodes.

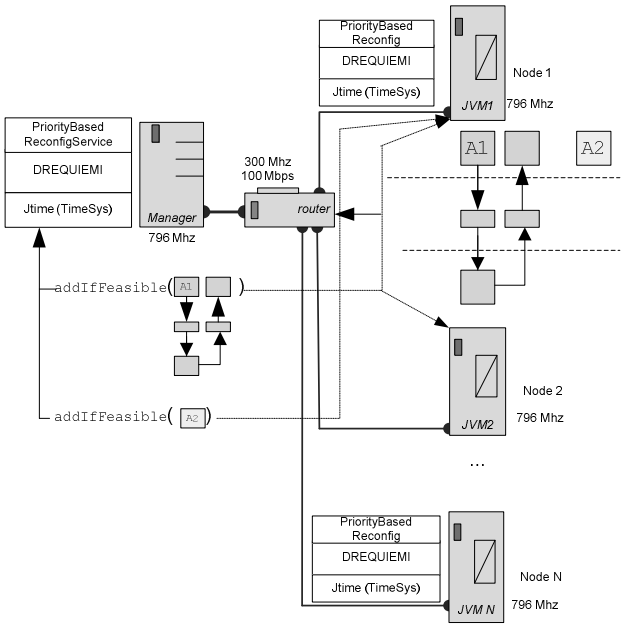


Figure 4. Evaluated scenario (All 796Mhz DREQUIEMI- 100 Mbps Ethernet)

A. *Cost Tributaries*

The results (Figure 5) show a strong dependence among the number of schedulable entities (tasks) and the time required to carry out their deployments. In the experiments 4096 schedulable tasks keep the cost over 1 second. The minimum cost, which happens with one task and one processor, keeps it in below 100 µs. There is also a linear dependence among the number of processors and the time required to run the feasibility algorithm. This dependence is more moderate than the existing one in the number of JVMs.

The previous results (Figure5) do not take into consideration the cost of allocating tasks in a remote node. Figure 6 shows the cost of allocating a number of schedulable objects in a JVM node. It considers three different schedulable segments: threads (`Rthread`), event handlers (`AsyncEventHandler`), and remote objects (`RtRO`).

These results are complemented considering the cost of sending setup information from the manager node to the subscribed nodes. Figure 7 shows the total cost of creating a remote entity in a local node considering a different number of schedulable entities.

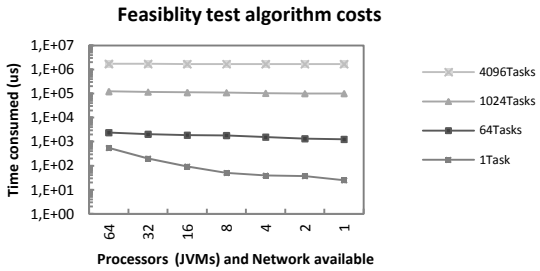


Figure 5. Minimum Cost of partitioning the system dependence using DREQUIEMI (796Mhz-100Mbps at manager node)

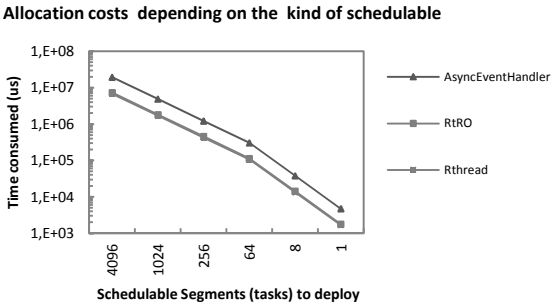


Figure 6. Task instantiation (796Mhz-100Mbps) cost, locally in a JVM node

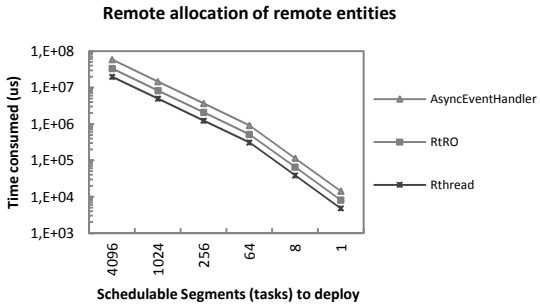


Figure 7. Remote allocation costs (796Mhz-100Mbps) using DREQUIEMI

B. *Total Configuration Time*

The total {re}configuration time is calculated as the time consumed deciding in which node is allocated each schedulable part of an application and the cost of deploying this task. Figure 8 shows this time for the given configuration (i.e. considering [4-32] JVMs (Java Virtual Machines) and [1-4096] AyncEventHandler schedulable segments). The results showed that the service is not able to perform 1024 assignments and deployments per second, even when there is a single JVM that has to host all tasks.

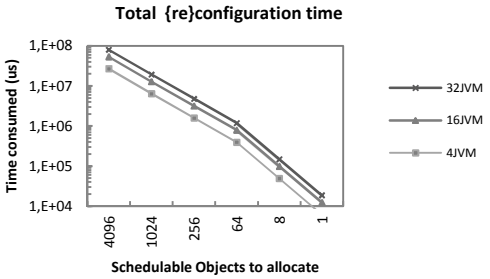


Figure 8. Total reconfiguration cost dending on the number of schedulables and the number of virtual machines (796Mhz-100Mbps) using DREQUIEMI. All schedulables are async event handlers (worst cost).

IV. Related Work

This section focuses on middleware aspects. It goes through the real-time middleware reviewing the way different architectures have dealt with similar problems.

DRTSJ [9, 30], the leading effort towards a distributed real-time Java, technology is focused on a scheduling framework to accommodate the distributed real-time thread. The proposed real-time reconfiguration service may be adapted to the distributable threads used by DRTSJ to configure a global system dynamically. York's approach [31] may also use the real-time reconfiguration service to carry out on-line configurations. From the point of view of the three profiles defined by UPM [32], the described reconfiguration service is useful for its flexible profile.

Recently, researchers working in RT-CORBA have described a distributed real-time manager [33, 34]. This distributed real-time manager connects different ORBs with a task manager in charge of allocating components to ORBs and performing load balancing and admission control. The described real-time reconfiguration service is closer to their model, i.e. both carry out admission control dynamically.

In RT-SOA (Service Oriented Architecture), it stands out the work carried out in real-time service composition ([4,35][5][8][14]), which deals with the problem of dynamic allocation of services with multiple available implementations. The authors addressed the problem of finding an optimum system configuration with composition algorithms, which have heuristics and figures of merit. The use of these techniques allows reducing the overhead of executing these algorithms several orders in magnitude. The proposed real-time reconfiguration service could be extended with these parameters.

An eventual piece of work is the distributed transaction manager (DTM) [36]. This service extends the contract model of FRESCOR [37] from a local scenario to a distributed system. The main difference between DTM and the service designed in this paper is in terms of entities negotiated and negotiation algorithms. Nevertheless, the algorithms described for the configuration service could be also included in DTM and vice versa.

V. Conclusions and Future work

Next generation real-time systems will benefit from having an enhanced infrastructure able to cope with predictable reconfiguration. The proposed reconfiguration service provides a template for a reconfiguration service in distributed real-time Java. The template may be extended to include different strategies and different kinds of applications. The empirical evaluation of a simple configuration strategy showed total reconfiguration costs that may be in hundreds milliseconds for a small number of nodes and schedulable segments.

Our ongoing work is focused on mode change protocols ([38, 39]). Implicitly, the addressed configuration service assumes that applications may be stopped and started at runtime without requiring configuration from the user. However, this is not realistic because reconfiguration is application-dependent in general. One solution to this problem is mode change protocols, which carry out this activity taking into consideration application nature. Additionally, the authors are exploring reconfiguration for real-time OSGi platforms (like [20, 40][41][42]).

Acknowledgements

This research was partially supported by the European Commission (ARTIST2 NoE, IST-2004-004527; iLAND ARTEMIS-JU Call 1) and by the Spanish national project REM4VSS (TIN-2011-28339).

References

[1] R. Rajkumar, Insup Lee, Lui Sha and J. Stankovic. Cyber-physical systems: The next computing revolution. Presented at Design Automation Conference (DAC), 2010 47th ACM/IEEE. 2010, .

[2] E. A. Lee. Cyber physical systems: Design challenges. Presented at International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC). 2008,

[3] J. White, B. Dougherty, R. E. Schantz, D. C. Schmidt, A. A. Porter and A. Corsaro. R&D challenges and solutions for highly complex distributed systems: A middleware perspective. *J.Internet Services and Applications 3(1),* pp. 5-13. 2012.

[4] M. Garcia-Valls, I. Rodriguez-Lopez, L. Fernandez-Villar, I. Estevez-Ayres and P. Basanta-Val, "Towards a middleware architecture for deterministic reconfiguration of service-based networked applications," in *15th IEEE Conference on Emerging Technologies and Factory Communication,* Bilbao, 2010, pp. 1-4.

[5] M. García-Valls, I. Rodríguez-López and L. Fernández-Villar. iLAND: An enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems. *Industrial Informatics, IEEE Transactions on* pp. -. 2012.

[6] P. Basanta-Val and J. S. Anderson, "Using real-time java in distributed systems: Problems and solutions," in *Distributed and Embedded Real-Time Java Systems*, T. H. Toledano and A. J. Wellings, Eds. Springer, 2012, pp. 23-45.

[7] P. Basanta-Val, M. Garcia-Valls and I. Estevez-Ayres. Towards a cyber-physical architecture for industrial systems via real-time java technology. *Computer and Information Technology, International Conference on 0*pp. 2341-2346. 2010.

[8] M. García-Valls, P. Basanta-Val and I. Estévez-Ayres. Real-time reconfiguration in multimedia embedded systems. *Consumer Electronics, IEEE Transactions on* pp. 1287. 2011. Available: 10.1109/TCE.2011.6018885.

[9] J. S. Anderson and E. D. Jensen. Distributed real-time specification for java: A status report (digest). Presented at JTRES '06: Proceedings of the 4th International Workshop on Java Technologies for Real-Time and Embedded Systems. 2006, .

[10] D. Tejera, A. Alonso and M. A. de Miguel. RMI-HRT: Remote method invocation - hard real time. Presented at Proceedings of the 5th International Workshop on Java Technologies for Real-Time and Embedded Systems. 2007

[11] P. Basanta-Val, M. Garcia-Valls and I. Estevez-Ayres, "An architecture for distributed real-time java based on RTSJ and RMI," in *15th IEEE Conference on Emerging Technologies and Factory Communication,* 2010, pp. 1-8.

[12] J. Rodriguez, D. Decouchant, S. Mendoza and C. M. Escobar. Java-based framework for implementing soft real-time distributed applications. Presented at Computer Science, 2008. ENC '08. Mexican International Conference on. 2008, .

[13] M. Malohlava, A. Plsek, F. Loiret, P. Merle and L. Seinturier, "Introducing distribution into a RTSJ-based component framework," in *2nd Junior Researcher Workshop on Real-Time Computing (JRWRTC'08),* Rennes, France, 2008, .

[14] García-Valls M., R. Castro, I. Estévez-Ayres, P. Basanta-Val and I. Rodríguez-López, "A bounded-time ServiceComposition algorithm for distributed real-time systems," in *Proc. 9th International Conference on Embedded Software and Systems,* Liverpool, UK, 2012, pp. 25-27.

[15] P. Basanta-Val, M. Garcia-Valls and I. Estevez-Ayres. Simple asynchronous remote invocations for distributed real-time java. *Industrial Informatics, IEEE Transactions on 5(3),* pp. 289-298. 2009. Available: 10.1109/TII.2009.2026271.

[16] P. Basanta-Val, M. Garcia-Valls and I. Estevez-Ayres. A dual programming model for distributed real-time java. *Industrial Informatics, IEEE Transactions on 7(4),* pp. 750-758. 2011.

[17] P. Basanta-Val, I. Estevez-Ayres, M. Garcia-Valls and L. Almeida. A synchronous scheduling service for distributed real-time java. *Parallel and Distributed Systems, IEEE Transactions 21(4),* pp. 506. 2010.

[18] P. Basanta-Val, L. Almeida, M. Garcia-Valls and I. Estevez-Ayres. Towards a synchronous scheduling service on top of a unicast distributed real-time java. Presented at Real Time and Embedded Technology and Applications Symposium, 2007.RTAS '07.13th IEEE. 2007, .

[19] P. Basanta-Val, M. Garcia-Valls and I. Estevez-Ayres. No-heap remote objects for distributed real-time java. *ACM Trans.Embed.Comput.Syst. 10(1),* pp. 1-25. 2010.

[20] P. Basanta-Val, M. Garcia-Valls and I. Estevez-Ayres. Enhancing OSGi with real-time java support. *Software: Practice and Experience -(-),* pp. ---. 2012.

[21] P. Basanta-Val, M. Garcia-Valls and I. Estevez-Avres, "Real-time distribution support for residential gateways based on OSGi," in *11Th IEEE Conference on Consumer Electronics,* Las Vegas, 2011, pp. 747-748.

[22] Bollella G. et al. The real-time specification for java. 2001. Available: http://www.rtsj.org/.

[23] P. Basanta-Val, M. Garcia-Valls and I. Estevez-Ayres, "Using switched-ethernet and linux TC for distributed real-time java infrastructures," in *Work-in-Progress Proceedings IEEE RTAS 2010,* 2010, .

[24] P. Basanta-Val, M. Garcia-Valls and I. Estevez-Ayres. Non-functional information transmission patterns for distributed real-time java. *Software: Practice and Experience 41(12),* pp. 1409-1435. 2011.

[25] P. Basanta-Val, M. Garcia-Valls and I. Estevez-Ayres. Towards propagation of non-functional information in distributed real-time java. Presented at Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2010 13th IEEE International Symposium on. 2010, .

[26] J. Sun. Fixed-priority end-to-end scheduling in distributed real-time systems. 1997.

[27] K. Tindell, A. Burns and A. J. Wellings. Analysis of hard real-time communications. *Real-Time Syst. 9(2),* pp. 147-171. 1995.

[28] J. C. P. Gutierrez and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. Presented at IEEE Real-Time Systems Symposium. 1998, .

[29] K. Lakshmanan and R. Rajkumar. Scheduling self-suspending real-time tasks with rate-monotonic priorities. Presented at Proceedings of the 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium. 2010,

[30] J. S. Anderson, B. Ravindran and E. D. Jensen. Consensus-driven distributable thread scheduling in networked embedded systems. Presented at EUC'07: Proceedings of the 2007 International Conference on Embedded and Ubiquitous Computing. 2007, .

[31] A. Borg and A. J. Wellings. A real-time RMI framework for the RTSJ. Presented at Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on. 2003, .

[32] D. Tejera, R. Tolosa, M. A. d. Miguel and A. Alonso. Two alternative RMI models for real-time distributed applications. Presented at ISORC '05: Proceedings of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05). 2005,

[33] K. Bryan, L. C. DiPippo, V. F. Wolfe, M. Murphy, J. Zhang, D. Niehaus, D. Fleeman, D. W. Juedes, C. Liu, L. R. Welch and C. D. Gill. Integrated CORBA scheduling and resource management for distributed real-time embedded systems. Presented at IEEE Real-Time and Embedded Technology and Applications Symposium. 2005, .

[34] Y. Zhang, C. D. Gill and C. Lu. Configurable middleware for distributed real-time systems with aperiodic and periodic tasks. *IEEE Trans. Parallel Distrib. Syst. 21*pp. 393-404. 2010.

[35] I. Estevez-Avres, L. Almeida, M. Garcia-Valls and P. Basanta-Val. An architecture to support dynamic service composition in distributed real-time systems. 2007.

[36] D. Sangorrin, M. G. Harbour, H. Perez and J. J. Gutierrez. Managing transactions in flexible distributed real-time systems. Presented at Ada-Europe. 2010, .

[37] M. Aldea, G. Bernat, I. Broster, A. Burns, R. Dobrin, J. M. Drake, G. Fohler, P. Gai, M. Gonzalez Harbour, G. Guidi, J. J. Gutierrez, T. Lennvall, G. Lipari, J. M. Martinez, J. L. Medina, J. C. Palencia and M. Trimarchi. FSF: A real-time scheduling architecture framework. Presented at RTAS '06: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium. 2006, .

[38] M. Garcia-Valls, A. Alonso and J. A. d. l. Puente. Mode change protocols for predictable contract-based resource management in embedded multimedia systems. Presented at ICESS '09: Proceedings of the 2009 International Conference on Embedded Software and Systems. 2009, .

[39] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Syst. 26(2),* pp. 161-197. 2004.

[40] T. Richardson and A. J. Wellings, "RT-OSGi: Integrating the OSGi framework with the real-time specification for java," in *Distributed and Embedded Real-Time Java Systems*, M.T. Higuera-Toledano and A. J. Wellings, Eds. Springer, 2011, pp. 293-322.

[41] J. C. Américo, W. Rudametkin and D. Donsez. Managing the dynamism of the OSGi service platform in real-time java applications. Presented at Proceedings of the 27th Annual ACM Symposium on Applied Computing. 2012,

[42] M. Garcia-Valls and P. Basanta-Val. A practical solution for functional reconfiguration of real-time service-based applications through partial reconfiguration schedulability. On 2012 International Workshop on Real-Time and Distributed Computing in Emerging Applications. San Juan (Puerto Rico) 4 December 2012.