A Model for System Resources in Flexible Time-Triggered Middleware Architectures

Adrian Noguero[1], Isidro Calvo[2], Luis Almeida[3], and Unai Gangoiti[2]

[1] Tecnalia, Software Systems Engineering Unit,
Parque Tecnológico de Zamudio #202, 48170, Zamudio, Spain
adrian.noguero@tecnalia.com
[2] University College of Engineering of Vitoria-Gasteiz
Dept. of Systems Engineering and Automatic Control (UPV/EHU), Spain
{isidro.calvo,unai.gangoiti}@ehu.es
[3] IEETA/DEEC-FE University of Porto, Portugal
lda@fe.up.pt

Abstract. Middleware has become a key element in the development of distributed Cyber Physical Systems (CPS). Such systems often have strict non-functional requirements, and designers need a means to predict and manage non-functional properties. In this work, the authors present a mathematical model for the most relevant resources managed by FTT middleware architectures; namely, (1) processor, (2) memory, (3) energy and (4) network. This model can be used both off-line for simulation and designing purposes of a Cyber Physical System (CPS), or in run-time within an admission test or inside the algorithm of a specific scheduling policy executed by the middleware. In such case, the admission test is aimed at predicting whether a system fulfils the non-functional requirements or not before carrying out any modification in its execution plan at run-time.

Keywords: Middleware architectures, Cyber-Physical Systems (CPS), Distributed Embedded Real-Time Systems, Resource Modeling, Admission Test.

1 Introduction

Cyber-Physical Systems (CPS) are integrations of computation and physical processes [20]. This kind of systems is being increasingly used in different domains such as healthcare, transportation, process control, manufacturing or electric power grids. CPS interact with the physical world and, typically, must operate dependably, safely, securely, efficiently and in real-time. Consequently, they require new computing and networking technologies capable of supporting them adequately in environments qualitatively different from those found in general purpose computing.

CPS require intensive use of the communication networks and, as described in [21] IP technologies are increasingly used. However, as the complexity of the distributed systems increases the use of middleware distribution technologies such as CORBA

R. Szabó and A. Vidács (Eds.): EUNICE 2012, LNCS 7479, pp. 215–226, 2012.
© IFIP International Federation for Information Processing 2012

[1], DDS [2], ICE [3] or Java RMI [4] becomes convenient, since they help developers in the construction of the new applications. Still, most of these solutions lack of mechanisms to deal with non-functional requirements. Several extensions of these middleware architectures have been developed to cope with one or more of these non-functional requirements in distributed applications [5, 6, 7, 8]. The way in which these extensions manage non-functional properties varies significantly among them. Moreover, although some of these extensions provide mechanisms to control non-functional properties, frequently they do not model the resources used by the distributed system, which is a key issue in Cyber-Physical Systems.

According to [20], new computing and networking abstractions are needed to deal with the entities used in CPS, since they must be able to represent the passage of time and concurrency which are intrinsic to the physical world. In addition, as CPS do not operate in controlled environments, they must be designed in a robust way so they are capable of adapting to subsystem failures or changes in the operational environment.

In this scenario, some middleware architectures based on the Flexible Time-Triggered (FTT) paradigm [22] like [9] and [11] provide some mechanisms to deal with the passage of time and concurrency in a flexible way in distributed real-time systems. Even though there are some differences between these middleware architectures, they are essentially based on a central component, the so-called Orchestrator, which is aimed at coordinating the execution of applications distributed over a LAN. This coordination involves both the synchronized activation of the tasks of a distributed system as well as the communication among them. However, differently from other time-triggered approaches, the Orchestrator is capable of supporting the reconfiguration of the execution plan in run-time. In particular, in the case of FTT-CORBA [9, 10] tasks are wrapped as CORBA methods that are activated by the Orchestrator.

In this work, the authors present a mathematical model for the most relevant resources managed by FTT middleware architectures; namely, (1) processor, (2) memory, (3) energy and (4) network. This model can be used both off-line for simulation and designing purposes of the CPS, or in run-time within an admission test related to a specific scheduling policy executed at the Orchestrator. This admission test is aimed at predicting whether a system fulfils the non-functional requirements or not before carrying out any modification in the execution plan.

The rest of the paper is structured as follows: section 2 provides an overview of the FTT middleware architecture; section 3 describes the proposed model; and finally, section 4 draws some conclusions and presents some future work.

2 The FTT Middleware Architecture

The goal of the model presented in this paper is to provide a means to predict the status of the distributed system resources in design time and design an admission test to be applied in runtime. This test will consider certain non-functional properties of the distributed system, such as memory consumption, available computing power, etc. Therefore, modeling the resources of the distributed system is a prerequisite to any prediction algorithm or admission test. It has already been discussed that, as the number of non-functional properties managed by each middleware solution and the

way in which they are managed vary greatly from each other, a specific resources model is required per middleware architecture. In this paper the authors have focused on a test for FTT middleware architectures.

FTT middleware architectures use time windows, so called Elementary Cycles (EC), to plan the execution of and communications among the nodes of a distributed system. The EC is a configurable parameter that is loaded during start up.

A distributed system executes distributed applications each of which is comprised of a set of tasks. Each task represents an indivisible functional requirement, i.e. an encapsulated functionality available in the system. Indeed, one application may use the same task several times, and it is also possible that several distributed applications use the same functionality. As shown in figure 1, the order in which the tasks of a distributed application are executed is described by a directed graph.

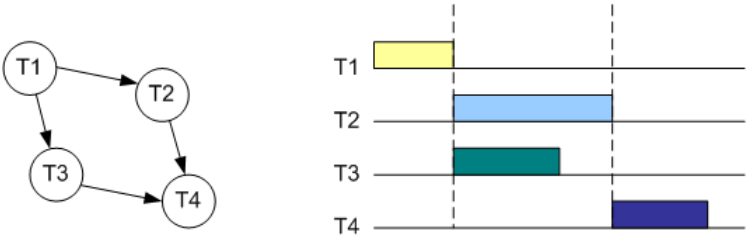

Fig. 1. Applications described as a directed graph and the associated task activation diagram

Each task is implemented by one functional object which is deployed in the distributed system. Objects are used to wrap one or more task implementations. It is possible that the same task is implemented several times, for example, to deploy each implementation in a different distributed node.

Communications among tasks are delivered by one mechanism based on the publisher/subscriber paradigm. Thus, each task declares which data tokens it will produce upon completion and which data tokens it will consume. Data tokens are related to a specific data topic, which identifies the type of the data token.

Following the FTT paradigm, the middleware architecture is governed by a central node which is in charge of managing the time in the distributed system, the so-called Orchestrator. At the beginning of a new EC this component notifies the distributed nodes about which tasks should be launched and which messages should be sent within that EC. The selection of the tasks that will be triggered and that will be allowed to communicate in each EC is carried out by means of specific scheduling policies that take into account not only the requirements of the distributed applications, but also the status of the distributed nodes.

2.1 Task Synchronization

Distributed task implementations are activated according to the functional requirements provided by the distributed applications loaded in the system and the scheduling policy deployed in the Orchestrator. Following the scheduling algorithm, the Orchestrator generates an activation plan that defines, for each EC, which task implementations should be activated.

At the beginning of each EC the Orchestrator sends an activation message to all the nodes of the system to activate the tasks (see figure 2). A local agent that executes at every node of the distributed system, the so-called Clerk, listens to these activation messages. Clerks can understand these activation messages and decide, based on the information sent by the Orchestrator, whether any task implementation deployed in the node must be activated, and the priority that should be applied to the activation. Activation messages are sent using UDP multicast.

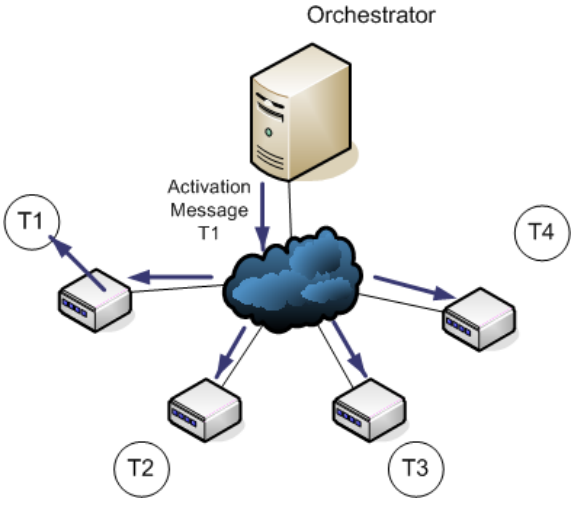

Fig. 2. Task activations in FTT middleware using multicast messages

2.2 System Resources Monitoring

In addition to the regular task activation messages, the FTT middleware architecture uses a reserved task identifier for monitoring the local resources of every node. More specifically, when Clerks receive an activation message that includes the reserved monitoring task identifier, they send to the Orchestrator the current status of the resources of the distributed node, namely, CPU utilization, available memory and remaining battery. Resources state messages are implemented using TCP communications, instead of multicast messages.

The status of each of the nodes is stored in the Orchestrator and it is made available to scheduling algorithms. This way, distributed system designers may implement new scheduling policies that fit the non-functional requirements of their applications.

2.3 Data Distribution

A model in which distributed tasks do not communicate among them is unrealistic; however, an uncontrolled data exchange paradigm could affect the timeliness of the task activations, thus affecting the overall timeliness of the distributed system. To cope with this issue the FTT middleware architecture includes a time-triggered data distribution service that allows the tasks to communicate without affecting the overall synchronization of the distributed system.

As shown in figure 3, the Orchestrator uses a polling strategy to gather the information of all the tasks that need to send a data token. After the polling process is completed the Orchestrator orders the messages in the queue according to their priority, and then, the data distribution process starts.

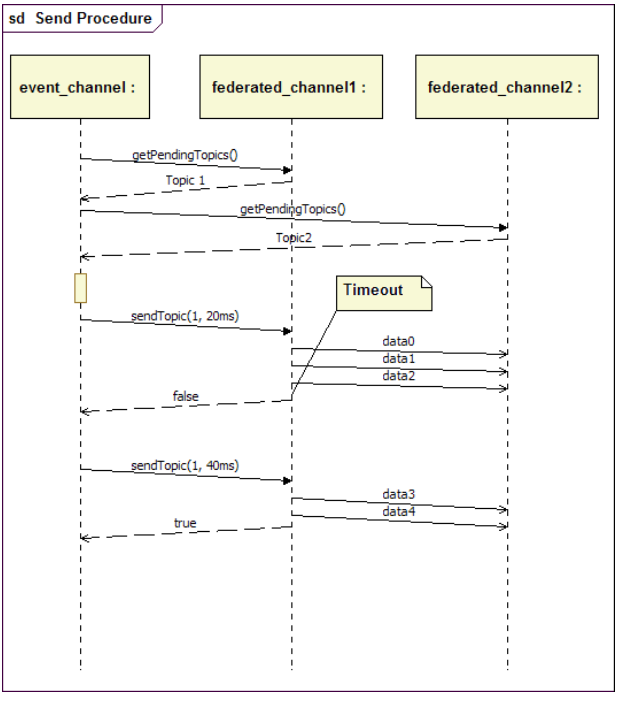

Fig. 3. Sequence diagram of the FTT middleware polling process

To trigger data transactions, the Orchestrator sends to the task that holds the first data token in the queue a petition to start sending the message and a timeout value, that is, the remaining time in the window (i.e. the time lapse to the next EC). Then, the selected task begins to send data messages using a multicast communication strategy to enable one to many communications. If one task sends a data token before its timeout, the FTT data distribution service notifies the Orchestrator, which will notify the next task in the data queue. Otherwise, the current sending process is stopped until the next EC.

It is important to note that the polling period, that is, the number of EC between two subsequent polling processes, is an important configuration parameter, since, as it will be discussed later, it will affect the available network bandwidth and the latency of the messages.

3 System Resources Modeling

Non-functional requirements of distributed applications are becoming more and more important. Consequently, it is necessary to define mathematical models that allow

system designers to predict the non-functional performance of distributed applications. However, as the intrinsic characteristics of middleware architectures have a crucial influence on these non-functional properties, ad-hoc mathematical models should be created in every case. The model presented in this paper is aimed at FTT middleware architectures, described above. More specifically, the presented model will focus on the non-functional properties managed by this architecture, namely, computing resources, memory and remaining battery. Additionally, the proposed solution will also provide a model for FTT data communications, analyzing the bandwidth and latency of the messages in terms of the configuration parameters of the middleware architecture.

3.1 CPU and Memory Utilization Modeling

The FTT middleware architecture provides means to manage the CPU utilization, available memory and remaining battery of the distributed system nodes. This information is very valuable for scheduling policies, since they enable fine grain tuning of the system behavior in terms of its non-functional properties.

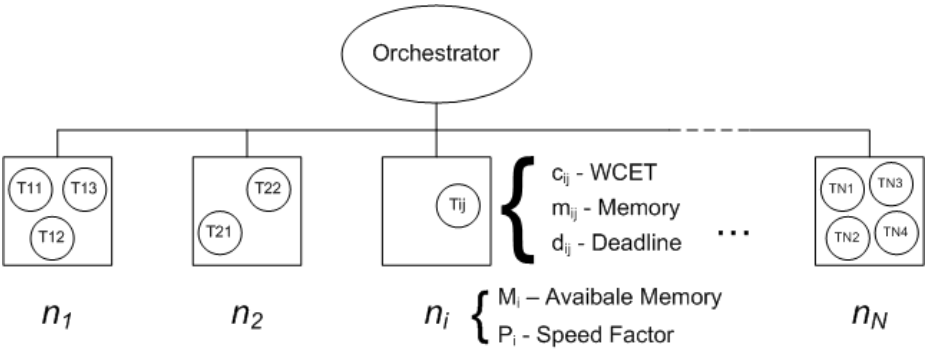

Fig. 4. Distributed system model in the FTT middleware paradigm

As depicted in figure 4, provided a theoretical distributed system constructed on top of a FTT middleware layer with N nodes, each node may be referred to as n_i, and its available memory bytes as M_i, where $i \in [1, N]$. Similarly, task implementations deployed in the distributed nodes can be referred to as $t_{i,j}$, where $j \in [1, T_i]$ and T_i is the number of task implementations deployed in node i. Task implementations are characterized by their worst case execution time ($c_{i,j}$), their relative deadline ($d_{i,j}$) and their memory requirement ($m_{i,j}$).

Since the FTT middleware architecture can be deployed on top of different hardware platforms, it is important to model the relative computing capabilities of the different distributed nodes in some way to cope with this heterogeneity. To keep the model simple, the distributed nodes have been provided of an extra parameter: the speed factor (P_i) [12], which measures the relative computing power of node i with regard to a theoretical reference processor whose computing power is taken as unit. Similarly, the worst case execution time ($c_{i,j}$) of each task implementation is referred to this theoretical reference processor.

In the FTT middleware architecture, the Orchestrator generates an activation plan defining which task implementations and with which priority level must be activated during each EC. In the latter context, the conditions that ensure that all task activations can be executed and that no deadlines will be missed can be expressed as:

$$\sum_{j=1}^{T_i} \frac{c_{i,j}}{d_{i,j} \cdot P_i} \leq U_{MAX} = T_i(2^{1/T_i} - 1) \qquad \forall i \in [1, N] \tag{1}$$

$$\sum_{j=1}^{T_i} m_{i,j} \leq M_i \qquad \forall i \in [1, N] \tag{2}$$

In equation 1, it has been assumed that the task deadlines coincide with its periods and that Deadline Monotonic priority assignment has been used in the Orchestrator [13]. The latter conditions stand for static distributed system configurations where deployed applications are periodic and do not change over time; however, many distributed applications need to deal with and react to environmental changes, especially in CPS. This is the case of the FTT middleware architecture. To cope with the dynamism of changing applications it is possible to refine these conditions by calculating the computational and memory loads of each node in each EC.

Let $U_i(p)$ and $M^U_i(p)$ be the computational load and the consumed memory of node i in the p-th EC respectively. Moreover, provided that the computation required by each task implementation is performed evenly over the lapse of time before the deadline is reached, it is possible to extract from the activation plan, for each EC, which tasks become active on the p-th EC, and which ones become inactive. Let τ_p be the set of task implementations activated on the p-th EC, and let τ'_p be the set of task implementations that become inactive on the p-th EC. In this case, the relative increment of the computational load in node i during the p-th EC may be expressed as follows:

$$\Delta U_i(p) = \sum_{\tau_p} \frac{c_{i,j}}{d_{i,j} \cdot P_i} - \sum_{\tau'_p} \frac{c_{i,j}}{d_{i,j} \cdot P_i} \quad \forall i \in [1, N],\ \forall p \in N \tag{3}$$

The latter expression can be further simplified to:

$$\Delta U_i(p) = \frac{1}{P_i}\left(\sum_{\tau_p} u_{i,j} - \sum_{\tau'_p} u_{i,j} \right) \quad \forall i \in [1, N],\ \forall p \in N \tag{4}$$

Where $u_{i,j}$ is the mean computational load introduced by $t_{i,j}$ per unit of time. Supposing that initially the load in the distributed system is 0 (i.e. $\Delta U_i(0) = 0$, $\forall i \in [1,N]$) it is possible to calculate the load of node i in the p-th EC as:

$$U_i(p) = \sum_{k=0}^{p} \Delta U_i(k) \quad \forall i \in [1, N],\ \forall p \in N \tag{5}$$

And, thus, the condition that prevents deadline misses can be expressed:

$$U_i(p) \leq U_{MAX} \quad \forall i \in [1, N], \ \forall p \in N \tag{6}$$

Similar equations can be derived for the available memory in terms of the relative increment of memory consumption:

$$M_i^U(p) = \sum_{k=0}^{p} \Delta M_i^U(k) \leq M_i \quad \forall i \in [1, N], \ \forall p \in N \tag{7}$$

3.2 Energy Consumption Modeling

The modeling of energy consumption in embedded systems has already been broadly covered in the literature [15, 16, 17]. These models provide energy consumption predictions with small error rates. However, these models are very complex and require numerous configuration parameters, many of which are difficult to measure. Thus, a simpler energy consumption model has been developed.

The consumed energy can be calculated in terms of power and time as E=P·t. Moreover, modern CPUs have different power figures in the IDLE state and during computation. Taking W, the length of an EC (see figure 5), as time unit, the energy consumed in the p-th EC can be expressed as:

$$\Delta E_i(p) = \left[P_i^{ACT} \cdot U_i(p) + P_i^{IDLE} \cdot (1 - U_i(p))\right] W \quad \forall p \in N \tag{8}$$

Where P_i^{ACT} and P_i^{IDLE} represent the power consumption of the node during active computation and in the idle state respectively. These two parameters can be easily found in the datasheets of the CPUs of the distributed nodes in the system. Indeed, in most modern CPUs it can be verified that [18]:

$$P_i^{ACT} = K_i \cdot P_i^{IDLE} \mid K_i > 2 \tag{9}$$

The fact that $K_i > 2$ (e.g. K_i=2.31 for Intel i7-975 and K_i=2.86 for AMD FX-8150) clearly shows the importance of the computational load, L_i, in the energy consumption figures of the distributed system. According to K_i, expression 8 can be rewritten as:

$$\Delta E_i(p) = \left[1 + (K_i - 1) L_i(p)\right] P_i^{IDLE} \cdot W \quad \forall p \in N \tag{10}$$

The authors are aware of the fact that the energy consumed by a distributed node is not only derived from computation tasks; in fact, other tasks, such as hard disk or flash accesses, consume much more energy that pure computation [19]. However, as the FTT middleware may only modify the performance of the distributed system through scheduling, the latter model enables the designers to implement energy aware scheduling policies (e.g. minimizing $\Sigma \Delta E_i(p)$ or minimizing the energy consumed by battery dependent nodes). Thus, expression 10 provides the FTT middleware

architecture of a powerful energy consumption model which is easy to configure, since it is only based on parameters available on the data-sheets of the computing resources of the distributed nodes.

3.3 Network Resources Modeling

The FTT middleware uses a centralized node to manage and split the network bandwidth. To do so, the central node of the architecture, this is, the Orchestrator, polls all the data producing task implementations that produce data for data tokens to be sent. Then, the data tokens are put in a queue and are ordered according to their priority. There is, therefore, part of the available network bandwidth that is not used for data messages, but for brokering the communication media.

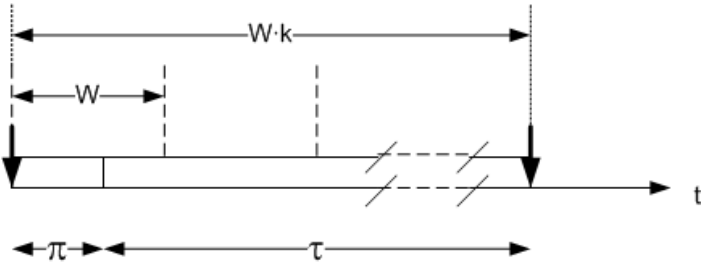

Fig. 5. A complete communication cycle under the FTT middleware architecture

Figure 5 depicts the latter scenario. It is important to note that it has been assumed that the communication medium is shared among the nodes. This assumption, even if it is rather pessimistic, increases the applicability of the proposed model with different communication technologies. The Orchestrator performs the polling process periodically, with a period that is a multiple of the EC. Hence, provided that W is the duration of an EC, the polling period of the system can be expressed as W·k where $k \in \mathcal{N}$. Note that the polling period represents a complete communication cycle, including a network brokering window or polling window (π) and a data transmission window (τ). Let n_p be the number of data producing nodes in the distributed system. Taking into account that the Orchestrator polls each of the data producing nodes individually for available data tokens, it is possible to conclude that the polling window depends on n_p. As a first approximation, it is possible to say that:

$$\pi(n_p) = P_0 \cdot n_p \tag{11}$$

Where P_0 is the mean polling time among all data producing nodes. Hence, the system will only be capable of transmitting data if some time is reserved for the data transmission window, τ. This can be expressed mathematically as follows:

$$\tau(k, n_p) = W \cdot k - P_0 \cdot n_p \geq 0 \quad \forall k, n_p \in N \tag{12}$$

From equation (12) distributed system designers may extract the limit values for k and n_p in the case any of these two are fixed by application requirements. More specifically:

$$n_p = N_0 \Rightarrow k > \frac{P_0 \cdot N_0}{W} \Rightarrow k_{\min} = \left\lceil \frac{P_0 \cdot N_0}{W} \right\rceil \tag{13}$$

$$k = K_0 \Rightarrow n_p < \frac{K_0 \cdot W}{P_0} \Rightarrow n_{p\max} = \left\lfloor \frac{K_0 \cdot W}{P_0} \right\rfloor \tag{14}$$

The equivalent network capacity is also an important parameter for the distributed system designer, as it must be ensured that this capacity limit is not exceeded. The relative capacity of a network managed by the FTT middleware can be modeled as the actual data transmission window (τ), divided by the nominal transmission window of the network, this is, the whole communication cycle:

$$C(k, n_p) = \frac{\tau(k, n_p)}{W \cdot k} = \frac{W \cdot k - P_0 \cdot n_p}{W \cdot k} = 1 - \frac{P_0 \cdot n_p}{W \cdot k} \tag{15}$$

The utilization of the latter models can be twofold: on the one hand, these mathematical models can be exploited by distributed system designers in design time as support for their design decisions and, moreover, they can also be used to create simulation tools; and, on the other hand, they can be implemented on the FTT Orchestrator to enable non-functional property management at runtime, exploiting this information for example, in the implementation of scheduling algorithms.

4 Conclusions and Future Work

In this paper the authors have presented a mathematical model of the system resources for a distributed system built on top of an FTT middleware architecture, such as FTT-CORBA. More specifically, the work presents models for CPU and memory utilization, power consumption and network resources.

The provided models can be applied both in design time and in runtime; and, thus, they can be implemented in several ways (e.g. design support tools, simulation tools or scheduling policies).

As future work, the authors will integrate the presented models in the FTT-Modeler tool [14], which is the user level tool of FTT-CORBA. Moreover, the authors will explore new scheduling policies and algorithms that fully exploit the presented models in runtime, enabling runtime optimization of the non-functional properties of these systems.

Finally, the proposed models are relatively pessimistic, in order to ensure that they are valid for a broad number of network technologies. With regards to the network management it has been assumed that a shared transmission medium is used. This

may be pessimistic when Switched Ethernet is used. Also, it has been assumed that local schedulers at the distributed nodes execute the rate monotonic algorithm where task deadlines are the periods. Executing Earliest Deadline First scheduling would be more efficient. However, these more precise models could be integrated in FTT middleware architectures in the same way as the ones describes above.

Acknowledgements. This work has been partly supported by the ARTEMIS JU through the CRAFTERS project (grant no. 295371) and the iLand project (grant no. 10026), the MICINN through the DPI2009-0812 project and the Basque Government through the S-PE11UN061 SAIOTEK project.

References

1. OMG, Object Management Group, Common Object Request Broker Architecture: Core Specification, Version, 3.0.3 (March 2004)
2. OMG, Object Management Group, Data Distribution Service for real-time systems, version 1.2 (2007)
3. Henning, M.: A new approach to object oriented middleware. IEEE Internet Computing 8(1), 66–75 (2004)
4. Maassen, J., van Nieuwpoort, R., Veldema, R., Bal, H., Kielmann, T., Jacobs, C., Hofman, R.: Efficient Java RMI for Parallel Programming. ACM Transactions on Programming Languages and Systems 23(6), 747–775 (2001)
5. Sadjadi, S.M., McKinley, P.K.: Transparent autonomization in CORBA Comput. Netw. 53(10), 1570–1586 (2009)
6. Zhang, Y., Lu, C., Gill, C., Lardieri, P., Thaker, G.: Middleware Support for Aperiodic Tasks in Distributed Real-Time Systems. In: Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium, pp. 113–122. IEEE Computer Society (2007)
7. Sadasivam, S.G., Ravindranathan, G.R., Gopalinis, R., Suresh, S.: A Novel Real Time Scheduling Framework for CORBA-Based Applications Journal of Object Technology 5(2), 171–188 (2006)
8. Hoffert, J., Schmidt, D.C., Gokhale, A.S.: Adapting Distributed Real-Time and Embedded Pub/Sub Middleware for Cloud Computing Environments Middleware, 21–41 (2010)
9. Calvo, I., Almeida, L., Perez, F., Noguero, A., Marcos, M.: Supporting a reconfigurable real-time service-oriented middleware with FTT-CORBA. In: Proceedings of Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference, pp. 1–8. IEEE Computer Society (2010)
10. FTT-CORBA project webpage, `http://sourceforge.net/projects/fttcorba/`
11. Basanta-Val, P., Estévez-Ayres, I., García-Valls, M., Almeida, L.: A Synchronous Scheduling Service for Distributed Real-Time Java. IEEE Transactions on Parallel and Distributed Systems 21(4), 506–519 (2010)
12. OMG, A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Version 1.0, OMG Adopted Specification: ptc/2009-11-02 (2009)
13. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM 20(1), 40–61 (1973)

14. Noguero, A., Calvo, I.: FTT-Modeler: A support tool for FTT-CORBA. In: 7th Iberian Conference on Information Systems and Technologies CISTI 2012 (2012)
15. Rakhmatov, D., Vrudhula, S.: Energy Management for Battery-Powered Embedded Systems. ACM Transactions on Embedded Computing Systems 2(3), 277–324 (2003)
16. Rakhmatov, D., Vrudhula, S., Chakrabarti, C.: Battery-conscious task sequencing for portable devices including voltage/clock scaling. In: Proceedings of the 39th Annual Design Automation Conference, pp. 189–194. ACM (2002)
17. Panigrahi, D., Dey, S., Rao, R., Lahiri, K., Chiasserini, C., Raghunathan, A.: Battery Life Estimation of Mobile Embedded Systems. In: Proceedings of the The 14th International Conference on VLSI Design (VLSID 2001). IEEE Computer Society (2001)
Schmid, P., Roos, A.: AMD FX: Energy Efficiency Compared To Eight Other CPU (2011), http://www.tomshardware.com/reviews/fx-power-consumption-efficiency,3060-11.html
18. Tiwari, V., Malik, S., Wolfe, A.: Power analysis of embedded software: a first step towards software power minimization. In: Proceedings of the 1994 IEEE/ACM International Conference on Computer-Aided Design, pp. 384–390. IEEE Computer Society Press (1994)
19. Lee, E.A.: Cyber Physical Systems: Design Challenges. In: 11th IEEE Symp. Object Oriented Real-Time Distributed Computing (ISORC 2008), pp. 363–369 (2008)
20. Koubâa, A., Andersson, B.: A Vision of Cyber-Physical Internet. In: Proc. of the Workshop of Real-Time Networks (RTN 2009), Satellite Workshop to, ECRTS 2009 (July 2009)
21. Almeida, L., Pedreiras, P., Fonseca, J.A.G.: The FTT-CAN Protocol: Why and How. IEEE Trans. on Industrial Electronics (TIE) 49(6), 1189–1201 (2002)