Towards a Load Balancing Middleware for Automotive Infotainment Systems

Yara Khaluf[1] and Achim Rettberg[2]

[1] University of Paderborn, Warburger Str. 100, 33098 Paderborn, Germany
khaluf@googlemail.com
[2] Carl von Ossietzky University Oldenburg, Escherweg 2, D-26121 Oldenburg, Germany
achim.rettberg@informatik.uni-oldenburg.de

Abstract. In this paper a middleware for distributed automotive systems is developed. The goal of this middleware is to support the load bal- ancing and service optimization in automotive infotainment and entertainment systems. These systems provide navigation, telecommunication, Internet, audio/video and many other services where a kind of dynamic load balancing mechanisms in addition to service quality optimization mechanisms will be applied by the developed middleware in order to improve the system performance and also at the same time improve the quality of services if possible.

Keywords: Load Balancing, Automotive Infotainment Systems, Service Migration, Service Quality, Service Availability.

1 Introduction

The last decade has seen a resurgence of advanced technologies being implemented in automobiles. Functions that were considered highly complex and difficult to be implemented are now being provided with various facilities.

When so many services are being embedded in a car on many devices, there might be mechanism which regulates the way of providing the services (tasks) in an optimal way with a high performance. The rest of this paper is organized as follows: section 2 describes the related work in the field of research where the developed middleware is located. Section 3 explains shortly the motivation of this work depending on some scenarios. The description of the middleware architecture and services provided by it are in section 4. Section 5 gives a look at the C-Simulation of the middleware and the conclusion and future work are then in section 6.

2 Related Work

Several publications regarding load balancing and extensive research has been done on static and dynamic strategies and algorithms [1]. Load balancing is

A. Rettberg et al. (Eds.): IESS 2009, IFIP AICT 310, pp. 238–249, 2009.

used in the domain of parallel and grid computing for optimization. Cybenko addresses the dynamic load balancing for distributed memory multiprocessors [2]. In [3] Hu et. al. regard an optimal dynamic algorithm and Azar discusses on-line load balancing. Diekmann et. al. present the difference between dynamic and static load balancing strategies for distributed memory machines [4]. Heiss and Schmitz introduce the Particle Approach for mapping tasks to processor nodes at runtime in multiprogrammed multicomputer systems solved by considering tasks as particles acted upon by forces. Furthermore the developed load balancing mechanisms are located on a separate middleware layer. Balasubramanian, Schmidt, Dowdy, and Othman consider in [5], and [6] middleware load balancing strategies and adaptive load balancing services. They introduce the Cygnus, an adaptive load balancing/ monitoring service based on CORBA middleware standard. The concept of dynamic reconfigurable automotive systems is also regarded in [7] and [8]

3 Motivation

Performance in automotive infotainment systems is a critical field of research. For many infotainment services like navigating and telecommunication ones, the performance is a question of service robustness and accurate response time of the system. For some other infotainment or entertainment systems the performance is a question of service quality more than other factors like by the audio and video services.

Overloaded ECUs, crashed ECU or new attached ECUs are cases to be faced in the domain of infotainment systems, this will make it important for the automotive industry to have some strategies which help to overcome such problems.

4 Middleware Architecture and Services

In this section, I will give an overview of the proposed architecture of the developed middleware in addition to a summary of the different services provided by it.

The design decision of embedding the developed load balancing and load optimization mechanisms on a separate middleware layer has the advantage to use this middleware later independent of the operating system running on the considered infotainment system 1. There are on the other hand middlewares which provide such load balancing techniques like CORBA [12] and Autosar [13], but they are not suitable to be used in the scenario of automotive infotainment systems we present in this work.

First of all, the developed automotive middleware has some design requirements to fulfill like fault tolerance, special communication model for automotive network, global time base and resource frugality. Such requirements are needed to satisfy the distributed, real-time and service criticality of automotive systems.

On the automotive infotainment systems there is a kind of real time behaviour with soft deadlines system needed. Like in the navigation system, where sometimes it is not more useful to get the direction where to drive after passing the

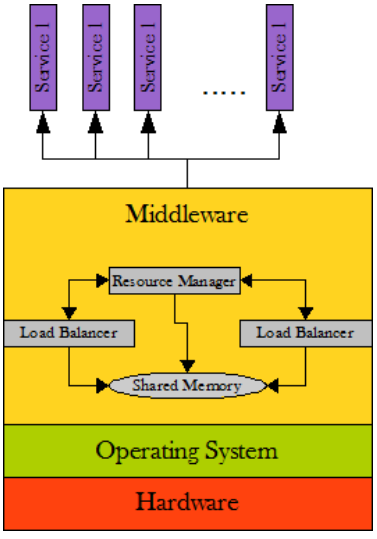

Fig. 1. Developed Middleware Layer

traffic crossing. Real time behaviour with hard deadlines is not needed on the infotainment systems yet, which does not give the possibility to discuss some concepts like automotive safety critical systems. Before explaining the main services provided by the developed middleware, I would like to give a short overview of the main components of the middleware 1.

4.1 Middleware Main Components

There are three main components of the developed middleware listed as followings:

Load Balancer: The load balancer is the component responsible of applying the different load balancing mechanisms supported by the middleware in addition to the performance optimization services.It is responsible of assigning first the tasks to the different vehicle ECUs and then balance and rebalance the different loads on the system ECUs in order to get optimal resource utilization and decrease the computing time of these tasks.

A kind of dynamic load balancing is needed here to be applied by the load balancer. There is always the ability to receive user requests and system tasks to perform on the system without expecting these a prior, in addition to the ECU crash and new attachment cases. for these reasons is the static load balancing not useful to be applied and a dynamic load balancing is the right solution. There maybe more than one load balancer on the system to guarantee the redundancy and reduce the communication costs.

Resource Manager: The resource manager supervises the different resources of the system and the ECUs. It is aware of the complete network resource situation

at any point of time and can provide the load balancer with the information it needs at any time. There maybe more than one resource manager in the system and all the resource managers synchronize with each other.

Shared memory: The shared memory is needed to store some information which is required by the other middleware components like load balancer and resource manager. According to the relative small size of the ECU local memory, we will use the concept of distributed shared memory, distributed on some or all system ECUs.

4.2 Middleware Services

A summary of the different services provided by the developed middleware to improve the overall performance is in following:

- **Tasks Assignment service:** This service is provided to assign the different tasks arrive to the system to the different ECUs to be executed. A specific scheduling policy is applied on the different tasks to determine their priorities. EDF (Earliest Deadline First) is the scheduling policy used in this work.
  The automotive infotainment and entertainment systems are heterogeneous environments, where the different ECUs differ from each other in one or more property. The main difference between the ECUs is that they are not able to execute the same kind of tasks referred to as task categories. Each ECU is able to execute a set of tasks categories consists minimum of one, for example the navigating system can provide navigating, music and internet services. This heterogeneity of the system ECUs make it not possible to assign a specific task to any of the available ECUs. By assigning a specific task according to its priority to some ECU, it must be checked that the ECU can execute the selected task and that the new load of the ECU is hold within a predefined limit.
  This service will be provided by the load balancer periodically. At each time the load balancer applies this mechanism, it sends a request to the resource manager to get the necessary information about the available ECUs in the system and the different task categories they can execute 2. After that the load balancer select the tasks according to their priorities and for each one tries to find the suitable ECU which can execute it with keeping the load within the predefined limits. After that, the assigned task will be stored also on the shared memory in a special queue called Currently Executing Tasks queue which will be used later in the case of ECU crash. This information will also be stored on the resource manager, as the resource manager must have at each point of time the up-to-date information of the system.
- **Load Balancing Service:** This service is the core of the developed middleware. it aims to balance the different loads of the different ECUs in order to improve the performance of the system. Dynamic load balancing mechanisms are used and developed to be suitable on the heterogeneous systems. The two developed policies developed mainly are the sender and receiver policies. The

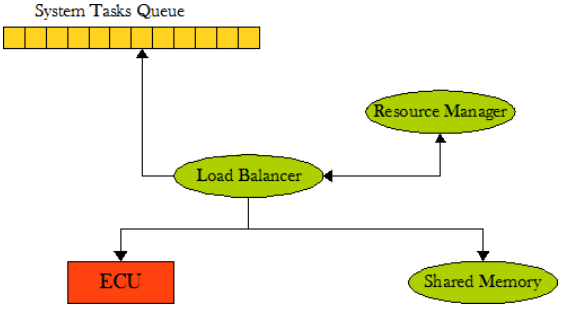

Fig. 2. Tasks Assignment Service

load balancing policies are applied also by the load balancer in aid of the information provided by the resource manager. At the beginning a threshold must be defined referred to as overload threshold. All ECUs with loads higher than the overload threshold are overloaded and all others are not overloaded in other words under-loaded. The resource manager performs checks of the system and ECUs periodically and depending on the overloaded and under-loaded number of ECUs it triggers the sender or the receiver load balancer policy on the load balancer. The development of the tradition sender and receiver policies for our special heterogeneous infotainment systems are in followings:

- **Sender Policy:** Like in the traditional sender policy, the load balancing mechanism will be triggered by the overloaded ECUs. So when the resource manager discovers in one of its periodic checks that there is a specific number of overloaded ECUs on the system, it will trigger the sender policy on some load balancer. The load balancer then will send back a request to the resource manager for getting some information necessary to apply the sender load balancing policy like the lists of overloaded and under-loaded ECUs and the different task categories they can execute 3. After this information is sent back to the load balancer it will then checks the tasks running on the overloaded ECUs according to their priorities and for each task search for the suitable destination ECU to migrate this task to in order to reduce the load of the handled overloaded ECU. Now the destination ECU must in the same time satisfy some condition to be suitable for the migration:
  - The destination ECU must be able of executing the task. In other word it must be able of executing the task category to which the task belongs.
  - The load of the destination ECU after the migration must not exceed the defined overload threshold. In other words the ECU must not become overloaded after the migration of the task.
  - The load of the destination ECU after the migration must be smaller than the load of the source ECU of the task to have benefit from this migration.

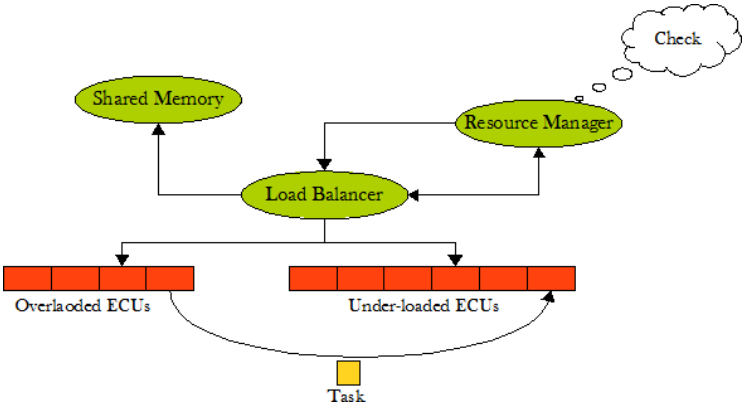

Fig. 3. Sender Load Balancing Policy

In case that only one ECU found which satisfies the previous conditions then the task will be migrated to this ECU. otherwise the task will be assigned to the ECU which execute it at a lower load. If all found ECUs can execute the task at the same load then the task will be assigned to the fastest ECU.

- **Receiver Policy:** This load balancing policy is triggered by the under-loaded ECUs of the system. This is the main difference between the sender and receiver policies. So when the resource manager discovers a specific number of under-loaded ECUs, it triggers the receiver policy on the load balancer. Then the load balancer in its turn sends a request back to the resource manager to get the required information to apply the policy and begin applying it in order to balance the loads overall the system.

If the defined overloaded threshold of the system is high then it is preferable to select the receiver load balancing policy. The reason is that there will be a few number of overloaded ECUs and some times no overloaded ECUs at all. And as the sender policy is triggered by the overloaded ECUs, this means that the sender policy will not be mostly triggered and no load balancing will be applied to handle the case of ECUs with high loads on the systems. On the other hand the receiver load balancing policy is triggered by the under-loaded ECUs and with a high overload threshold there will be always enough number of under-loaded ECUs to trigger this load balancing mechanism and improve the system performance.

In both sender and receiver load balancing policies, after a task is migrated, the information on the resource manager must be updated and the migrated task must be also stored on the shared memory in a queue called migrated tasks queue.

– **Related Loads Service:** This is a performance improvement Service developed with the goal of making the loads of the different ECUs on the infotainemnt system having similar loads which are related to a predefined

threshold referred to as Balance threshold. This service is applied by the load balancer and triggered first by the resource manager. To determine which of the available load balancers will be responsible to apply the mechanism, a token is used. The token is generated by the resource manager which set also the token timer. This token is sent to one of the load balancers and when its timer expires the load balancer begin to apply the related loads service. After finishing, the timer of the token is reset and it is sent to the next load balancer to apply the mechanism again. So the token will travel from one load balancer to the another till finish applying the mechanism. The related loads mechanism is applied on rounds. There is a predefined number of rounds determined as a design decision. Each round consist of two phases:

- **First Phase:** In the first phase of the round the load balancer builds two lists of ECUs 4, source ECUs and destination ECUs lists. The list of the source ECUs contains the ECUs with loads higher than the defined balance threshold, where the list of the destination ECUs contains the other ECUs. After the ECUs in the destination ECUs lists will be checked one by one and for each destination ECU tasks running on the source ECUs will be tested and the ones which are suitable to be migrated to this specific destination ECU will be marked with The checked ECU ID. To determine if some task is suitable to be migrated later to some destination ECU, the following conditions must be satisfied:
  - The destination ECU must be able to execute the task category of the tested task
  - The load of the destination ECU after migration must be smaller than the load of the source ECU of this task to have benefit from this migration

  After finished selecting the task suitable to be migrated to the currently handled destination ECU, comes the next destination ECU to check and

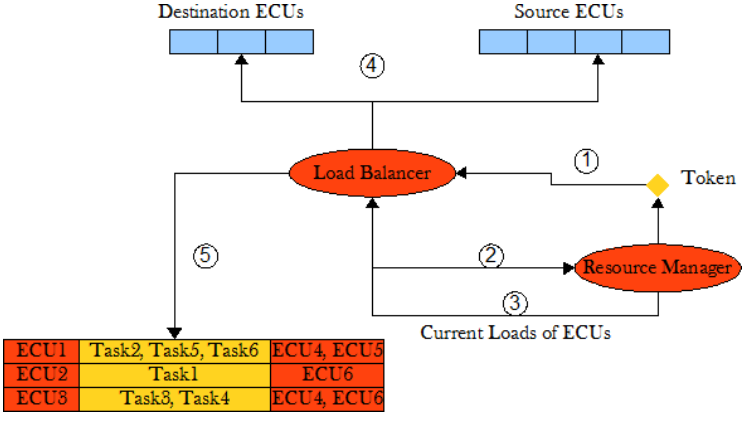

Fig. 4. Related Loads Mechanism-Round First Phase

so on till all destination ECUs are checked. At the end of this phase a table is built by the load balancer contains the destination ECUs and suitable tasks to be migrated to each of them in addition to the source ECUs of these tasks 4.

One rules of this mechanism is that: During the same round only one task is allowed to be migrated to a specific destination ECU. Migrating a task to some ECU represent an increment of the load of this ECU and to prevent a big increment jumps in load within one round, maximum one task is allowed to be migrated to a specific destination ECU.

At the end of the round first phase, we have noticed like in figure 4 that sometimes more than one task is suitable to be migrated to the same destination ECU. In such a case we need to select only one of these tasks and this is what will be done in the second phase of the round.

- **Second Phase:** The goal of this mechanism was to make the loads of the system different ECUs related to the predefined balance threshold as much as possible. In this phase we are going to choose at the maximum one task to be migrated to each of the destination ECUs. The best task to be chosen according to the goal of this mechanism, is the task which its migration makes the loads of its source and destination ECUs related to the defined balance threshold at the maximum. To determine which task, the calculations of the source and destination ECUs loads in case of migrating the tasks found in the first phase must be done.

  Depending of these calculations at most one task will be selected to migrate to each of the destination ECUs. After the migrating the information of the resource manager will be updated and the migrated tasks will be stored on the shared memory again in the migrated tasks queue.

The mechanism is applied recursively in rounds till one or both of the two following conditions is satisfied:

- The predefined number of rounds is reached
- The loads of the system ECUs are related to the balance threshold with an acceptable defined failure limits

– **Handle ECU Crash Cases Service:** electronic control units are like any controlling system can experience a crash. The reasons of this crash is not the subject of this work, but we must know that there is two kinds of crashes.

- Partly crash: where the ECU still able to perform some tasks or requests and can still provide some of the services it had provided.
- Totally crash: in the case of totally crash the ECU become not more able to provide any service or execute any task more.

In both crash cases the philosophy followed by the developed middleware to handle such crash cases is to try migrating and tasks and services the ECU can not execute or provide any more to other available ECU which can do this. In the case of our automotive infotainment system,as we know that the resource manager performs periodic checks on the system. If it discovers in one of these periodic checks that there is a crashed ECU in the system, the resource manager sends the IDs of the tasks and services could not execute and provide any more on the crashed ECU to the load balancer 5. Some of

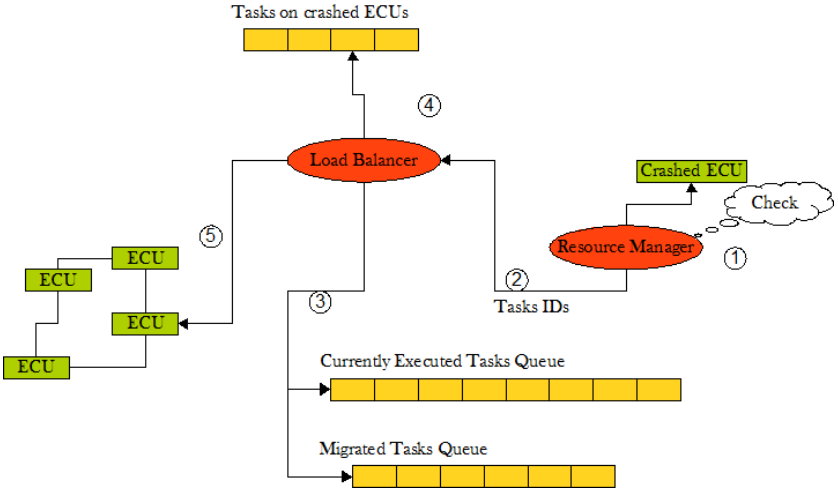

Fig. 5. Handle ECU Crash

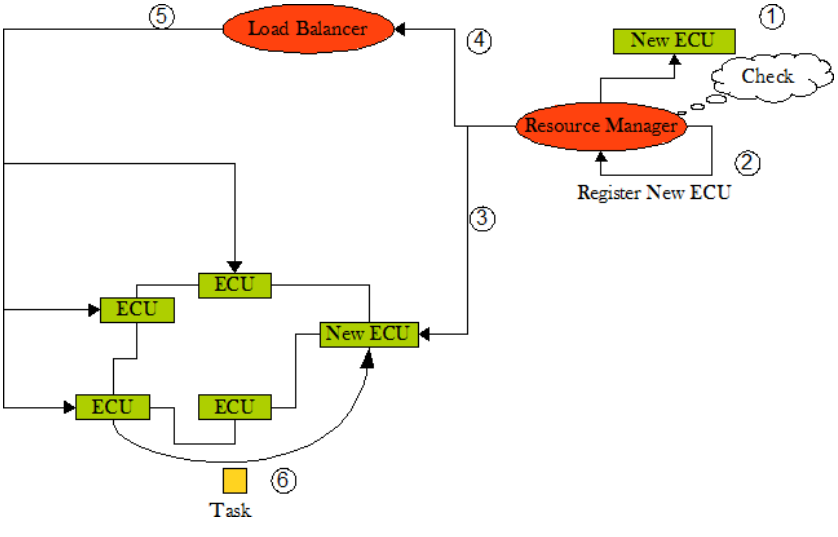

Fig. 6. Handle New ECU Attachment

these tasks were executing on the crashed ECU and other were migrated to it and not executed yet. The load balancer in its turn will access the shared memory the currently executing tasks queue and the migrated tasks queue and get the tasks were executing on or migrated to the crashed ECU. After that the load balancer will try to assign these tasks one by one according to their priorities to the available ECUs which can execute them and execute their categories.

– **Handle New ECU Attachment Service:** In the modern automotive infotainment systems there is the ability to attach new ECU to the system. The developed middleware then must be able to migrate tasks to the new ECU if it can execute them with a better quality. The resource manager discovers first the existence of a new ECU. After that it will register and integrate the new ECU in the system 6. The resource manager will also inform the load balancer that there is a new ECU attached to the system. The laod balancer access all the system ECUs and checks all the tasks and services running on them. If there is any of these tasks or services could be run with a better quality on the new attached ECU and with taking the ECUs loads into account. This task will be then migrated from its source ECU to the new attached ECU.

5 Middleware Simulation

A C-Simulation is also implemented within this work for the developed middleware including some of the main services it provides. Different scenarios were simulated with different parameters. One of the most important result to present here is the loads improvement after applying the sender load balancing policy. Figure 7 shows the loads improvement after applying the sender policy for only one time on five ECUs with twenty tasks. As we can notice in figure 7, ECU1 and ECU3 were the overloaded ECUs, where the other ECUs were not overloaded. After applying the load rebalancing mechanism for one time, the loads of ECU1 and ECU3 were reduced and the loads of ECU2, ECU4 and ECU5 were increased after becoming some tasks from the overloaded ECUs. But The new loads of ECU2, ECU4 and ECU5 are still under the predefined balancing threshold. so the result is an improvement in the overall performance of the system.

Usually many Parameters affect the performance improvement or load rebalancing on the system. Some of the most important ones are the size of the task categories the ECUs can perform and the load balancing threshold which is a predefined parameter. The size of the tasks categories the ECUs can execute

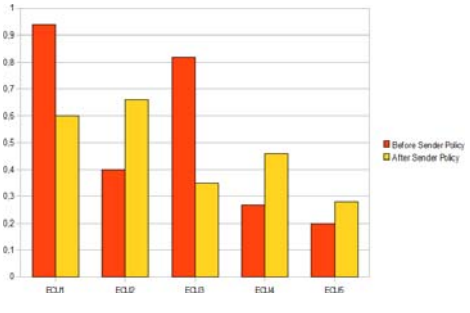

Fig. 7. Loads Improvement after applying Sender Policy one time

determines the level of migration flexibility by having the chance to migrate a service or task to more than one ECU, which lead to a better load rebalancing. on the other hand the predefined load balancing threshold is the parameter we depend on to determine the overload cases. It would improve the performance of the load balancing algorithm later if we can set the load balancing threshold dynamically at each time the algorithm is executed.

6 Summary and Future Work

We have presented in this work a middleware architecture for automotive infotainment and entertainment systems. This middleware enables a dynamic load balancing on a heterogeneous environment. Several services are provided by it like tasks assignment service, load balancing, related loads mechanism and handling of the ECU crash cases and new ECU attachment. The integration of load balancing is a step towards a self-reconfiguration within the vehicle. The main parameter considered in this heterogeneity environment was the difference in task categories could be executed by each ECU. later other parameters could be taken into account like the speed and memory size of the ECUs. Also the selection of the overload threshold used in the load balancing mechanisms could be dynamically done each time the load balancing policy is applied, depending on the current loads of system ECUs.

References

1. Hui, C.-C., Chanson, S.T.: Improved strategies for dynamic load balancing. IEEE Concurrency (1999)
2. Cybenko, G.: Dynamic load balancing for distributed memory multiprocessors. In: Parallel and Distributed Computing (1989)
3. Hu, Y.F., Blake, R.J.: An optimal dynamic load balancing algorithm (1995), `http://citeseer.ist.psu.edu/hu95optimal.htm`
4. Diekmann, R., Monien, B., Preis, R.: Load balancing strategies for distributed memory machines. World Scientific, Singapore (1997)
5. Balasubramanian, J., Schmidt, D.C.: Evaluating the performance of middleware load balancing strategies (2004), `http://citeseer.ist.psu.edu/635250.html`
6. Othman, O., Schmidt, D.: Optimizing distributed system performance via adaptive middleware load balancing. In: ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (2001)
7. Anthony, R., Rettberg, A.: Towards a Dynamically Reconfigurable Automotive Control System Architecture. In: Proceedings of the IESS 2007, Irvine, California, USA. Springer, Heidelberg (2007)
8. Jahnich, I., Rettberg, A.: Towards Dynamic Load Balancing for Distributed Embedded Automotive Systems. In: Proceedings of the IESS 2007, Irvine, California, USA. Springer, Heidelberg (2007)
9. Podolski, I., Rettberg, A., Drüke, I.: Towards Autonomous Sensor Networks by a Self-Configurable Middleware. In: Proceedings of Workshop on Sensor Networks and Applications (WSeNA 2008), Gramado, Brazil (2008)

10. Drüke, I., Podolski, I., Rettberg, A.: Integrating Dynamic Load Balancing Strategies into the Car-Network. In: Proceedings of the IEEE International Workshop on Electronic Desing, Test and Applications (DELTA 2008), Hong Kong (2008)
11. Drüke, I., Podolski, I., Rettberg, A.: Towards a Middleware Approach for a Self-Configurable Automotive Embedded System. In: Brinkschulte, U., Givargis, T., Russo, S. (eds.) SEUS 2008. LNCS, vol. 5287, pp. 55–65. Springer, Heidelberg (2008)
12. Othman, O., Ryan, C.Ó., Schmidt, D.C.: The Design of an Adaptive CORBA Load Balancing Service. IEEE Distributed Systems Online (2001)
13. Othman, O., Ryan, C.Ó., Schmidt, D.C.: The Design and Performance of an Adaptive CORBA Load Balancing Service. IEEE Distributed Systems Online (2001)