TOWARDS DYNAMIC LOAD BALANCING FOR DISTRIBUTED EMBEDDED AUTOMOTIVE SYSTEMS

Isabell Jahnich and Achim Rettberg
University of Paderborn/C-LAB, Germany
{isabell.jahnich, achim.rettberg}@c-lab.de

Abstract: This paper describes a middleware architecture for distributed automotive systems that supports dynamic load balancing of tasks. Load balancing could be applied if an external device is added to the vehicle. The middleware architecture has to deal with the fusion of such non-built-in devices. An important factor to set up such a system is the identification of the requirements that should be handled by the middleware architecture. Enriching the middleware with traditional load balancing strategies allows an easy exchange of tasks.

Keywords: Dynamic Load Balancing, Automotive Systems, Middleware Architecture

1. INTRODUCTION

Future application scenarios for vehicle electronic systems include the access to mobile devices that build ad-hoc networks with the built-in devices of the vehicle. Computer power and services of connected external devices to the built-in vehicle system allow the realization of new resource-intensive and innovative applications.

Nowadays powerful mobile consumer electronic devices like mobile phones and PDAs are very popular. Their connection to the vehicle system allows the migration of resource-intensive applications or tasks to increase the driver's comfort by faster processing, and to offer additional services.

To realize a seamless connection of mobile consumer electronic devices to the vehicle infrastructure, a middleware architecture is needed that supports the fusion of the additional device and the vehicle. Thus the attachment and the detachment of non-built-in devices have to be discovered and detailed device information and resources have to be registered. Furthermore the architecture is also responsible for the possible load balancing by migrating tasks from the vehicle system to the additional device.

Jahnich, I., Rettberg, A, 2007, in IFIP International Federation for Information Processing, Volume 231, Embedded System Design: Topics, Techniques and Trends, eds. A. Rettberg, Zanella, M., Dömer, R., Gerstlauer, A., Rammig, F., (Boston: Springer), pp. 97–106.

In the following a vehicle middleware architecture is presented that supports the attachment of additional devices. Furthermore it offers services to realize a load balancing based on different strategies. The rest of the paper is organized as follows: Section 2 will describe the related work in the field of research where our architectural approach is located. As a motivation for this paper Section 3 describes a use case scenario followed by the requirements of the architecture that is addressed in Section 4. Section 5 explains our architecture which is finally mapped to the use case scenario of Section 3. Some possible strategies of load balancing are regarded in Section 6 and finally Section 7 will give a conclusion and future work motivations.

2. RELATED WORK

There are several publications regarding load balancing and extensive research has been done on static and dynamic strategies and algorithms [6].

On the one hand, load balancing is a topic in the area of parallel and grid computing, where dynamic and static algorithms are used for optimization of the simultaneous task execution on multiple processors. Cybenko addresses the dynamic load balancing for distributed memory multiprocessors [3]. In [5] Hu et. al. regard an optimal dynamic algorithm and Azar discusses on-line load blancing [5]. Moreover Diekmann et. al. differentiate between dynamic and static strategies for distributed memory machines [4]. Heiss and Schmitz introduce the *Particle Approach* that deals with the problem of mapping tasks to processor nodes at run-time in multiprogrammed multicomputer systems solved by considering tasks as particles acted upon by forces.

All these approaches have the goal of optimizing the load balancing in the area of parallel and grid computing by migrating tasks between different processors, while our approach focuses the direct migration of selected tasks to a newly added resource. Furthermore we regard load balancing that is located on the middleware-layer.

Moreover there are static approaches, like [11], that address a finite set of jobs, operations and machines, while our approach deals with a dynamic set of tasks and processors within the vehicle system.

Balasubramanian, Schmidt, Dowdy, and Othman consider in [7], [9], and [8] middleware load balancing strategies and adaptive load balancing services. They introduce the *Cygnus*, an adaptive load balancing/monitoring service based on CORBA middleware standard. Their concept is primarily described on the basis of a single centralized server, while decentralized servers that collectively form a single logical load balancer is not explained in detail.

Moreover the topic of dynamic reconfigurable automotive systems is regarded in [2] and [1].

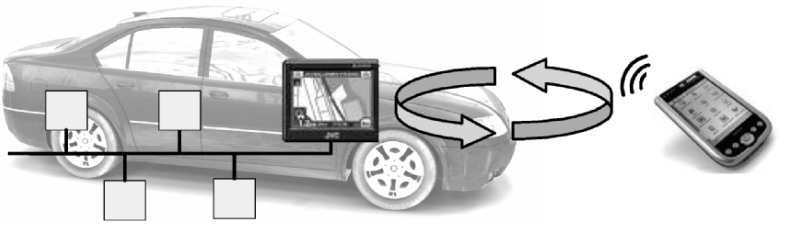

Figure 1. Use case scenario.

3. MOTIVATION

To run applications or tasks for example of the vehicle infotainment system more efficiently, a migration to the additional mobile device makes sense to use its unused resources. Thus it is possible to migrate for example tasks of the navigation system to the connected PDA for faster and more detailed map rendering and more optimal calculation of routing information.

After the connection of the PDA to the vehicle infotainment network with the aid of standardized interfaces like Bluetooth or WLAN, the device is discovered and the appropriate device information, locally running processes, and device and network resources are registered by a dedicated service.

In consideration of all running processes and the resources situation within the vehicle infotainment system appropriate services decide on a possible load balancing according to different strategies and initiate the task migration where required. Thus the appropriate navigation system tasks migrate from the navigation system to the PDA. And after the calculation the results of the tasks are sent back to the navigation system, where they are used.

4. REQUIREMENTS OF THE ARCHITECTURE

To realize the use case scenario described above a middleware architecture is required that fulfills several requirements.

- Event management
  - Additional devices have to be discovered by the vehicle infrastructure.
  - Device removals have to be discovered by the vehicle infrastructure.
- Device registration
  - Detailed information and capabilities of the newly added devices have to be registered.

- Resource management
  - Status information and resource load of each device of the vehicle have to be known.
- Load balancing
  - Potential task migrations have to be initiated based on different strategies.

5. ARCHITECTURE

In this section, we will give an overview of the proposed architecture. The architecture fulfills the requirements described in the previous section.

The operating system builds the interface between the hardware and the middleware (see Figure 2). Additionally, device drivers are necessary for specific hardware parts. The tasks run on top of the middleware. Middleware is a software layer that connects and manages application components running on distributed hosts. It exists between network operating systems and application components. The middleware hides and abstracts many of the complex details of distributed programming from application developers. Specifically, it deals with network communication, coordination, reliability, scalability, and heterogeneity. By virtue of middleware, application developers can be freed from these complexities and can focus on the application's own functional requirements.

Before explaining the design of our automotive middleware and the specific services, we enumerate the five requirements of automotive middleware. These requirements are resource management, fault-tolerance, specialized communication model for automotive networks, global time base, and resource frugality. These requirements are derived from the distributed, real-time, and mission-critical nature of automotive systems and differentiate automotive middleware from conventional enterprise middleware products.

A vehicle has a real-time nature. It is a system in which its correctness depends not only on the correctness of the logical result, but also on the result delivery time. Since a vehicle is subject to various timing constraints, every component in a vehicle should be designed in a way that its timing constraints are guaranteed a priori. At the same time, the timing constraints of a vehicle should be guaranteed in an end-to-end manner since an automobile is a distributed system and its timing constraints are usually specified across several nodes. For example, let us consider a typical timing constraint of an automobile. If pressing a brake pedal is detected at the sensor node, then the brake actuator node must respond to it within 1 ms. To meet this constraint, there must be a global resource manager that calculates the required amount of resources on each node and actually makes resource reservations to network

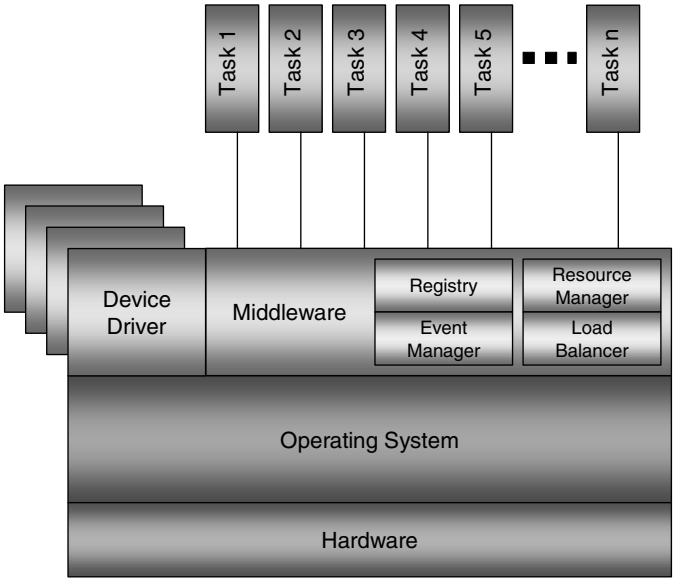

Figure 2. Self-configurable Architecture.

interface controllers and operating systems on distributed nodes. Automotive middleware is responsible for such resource management.

The middleware in our approach includes four components that offer specific services: *Registry*, *Event Manager*, *Resource Manager* and *Load Balancer* (see Figure 2).

The Event Manager is responsible for the device discovery. If a new device is added to the automotive system via technologies like Bluetooth or WLAN for example, it is recognized by the Event Manager component. Vice versa the Event Manager also notices the detaching of the device. In both cases it will inform the Registry of the middleware about the availability or the detaching of the additional device.

New devices are registered and detached devices are unsubscribed within the Registry service. During the registration the specific characteristics of the device (like memory, CPU, etc.) are stored within the Registry. Due to the distributed system the Registries of each vehicle ECU (Electronic Control Unit) communicate with each other to guarantee that each Registry of an ECU knows the actual status of all devices within the network inclusive of the newly added devices.

The Load Balancer spread tasks between the vehicle ECUs in order to get optimal resource utilization and decrease computing time. It evaluates possible migration of tasks based on different load balancing strategies. To guarantee a suitable migration the Load Balancer considers the current resource situation

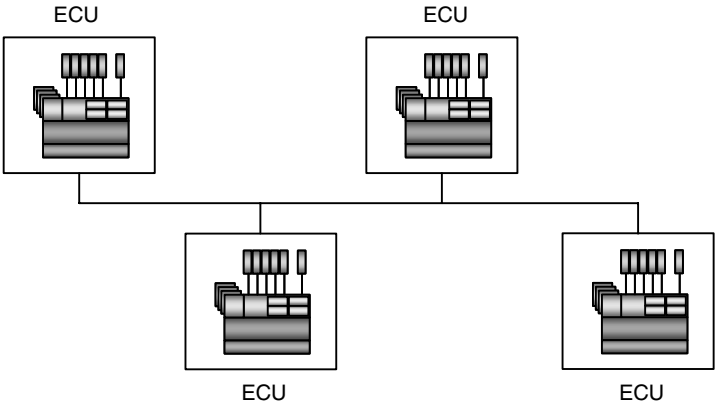

Figure 3. Distributed architecture.

on the ECUs with aid of the Resource Manager. Once a load balancing on an additional device is started, and this device is detached while the migrated tasks are executed, they will be re-started on the original ECU again. In this case the Event Manager is responsible to inform the Load Balancer to initiate this re-start.

The Resource Manager supervises the resources of the local ECU. To be aware of the complete network resource situation all Resource Managers synchronize with each other. Thus the Load Balancer gets the current resource situation of the complete vehicle infrastructure with aid of its local Resource Manager.

In our approach, the middleware is located on each ECU in the vehicle (see Figure 3). Every ECU has a unique ID. The ECU with the lowest ID is the master. Thus it is responsible for the control of the entire vehicle network, and newly connected and the detaching of additional devices are discovered by its Event Manager, device information is registered by its Registry, and its Load Balancer is responsible for the evaluation of the possible migration with the aid of the local Resource Manager. If the master ECU fails a new master will be chosen with the aid of the Bully-Algorithm [10].

5.1 SERVICE COMMUNICATION

According to the use case scenario description of Section 3 the connection of a mobile device like a mobile phone or PDA to the vehicle infrastructure is followed by several actions and communications (see Figure 4). In the following these activities will be regarded:

After the PDA has been attached to the vehicle infrastructure, the Event Manager of the master ECU recognizes the attachment of the additional device

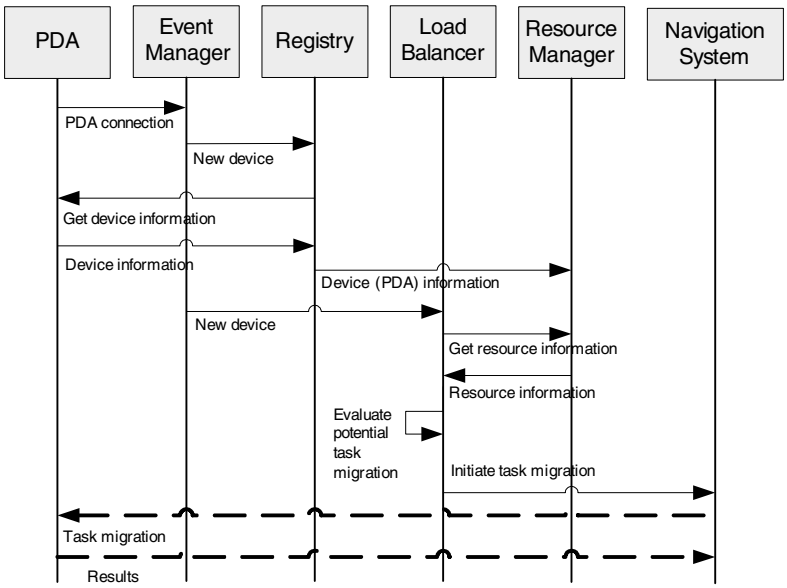

Figure 4. New device connected to the system.

and informs the local Registry about the newly added node. In the following this Registry asks the PDA for device and actual resource information. This information is stored by the Registry. Furthermore the resource details are forwarded to the local Resource Manager. The communication between the Registries of all ECUs makes sure that every ECU is informed about the additional device.

The event of the added PDA is also forwarded to the local Load Balancer by the Event Manager. To decide for a suitable load balancing strategy (see Section 6) it needs resource information about the current status within the network with aid of the Resource Manager.

According to the use case description of Section 3 the navigation system tasks migrate from the navigation system to the PDA and the results are sent back after the calculation.

6. LOAD BALANCING STRATEGIES

There are several possibilities to balance the load after additional devices have been connected to the vehicle network. Initiated by the Load Balancer component the new resources can be used and applications or tasks can be migrated to the additional device.

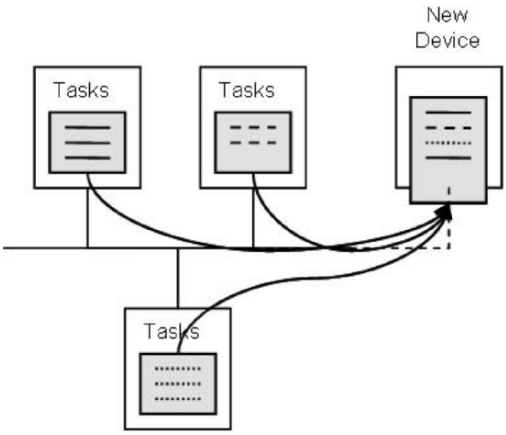

Figure 5. Round-Robin.

In the following three strategies will be explained: Task migration based on the Round-Robin algorithm, a strategy based on a migration from the most loaded processor and a cost based algorithm.

6.1 ROUND-ROBIN

As one of the simplest scheduling algorithms for processes *Round-Robin* assigns time slices to each process in equal portions and in order. All processes are handled without priority.

Based on this idea the Load Balancer assigns tasks to the processor of the additional device for a dedicated time slice. These tasks are migrated to the connected device and all resources of the vehicle infotainment network are relieved in an even manner (see Figure 5).

6.2 MOST LOADED

With the aid of the Load Monitor the current load of all nodes within the vehicle infotainment network will be monitored. The Load Balancer can request that information to decide whether to relieve busy devices by initiating a migration of tasks to the newly connected device. For this it generates a priority list which ranks the tasks from the busiest processor. In that way the tasks with the highest priority will be migrated to the resources of the additional device (see Figure 6).

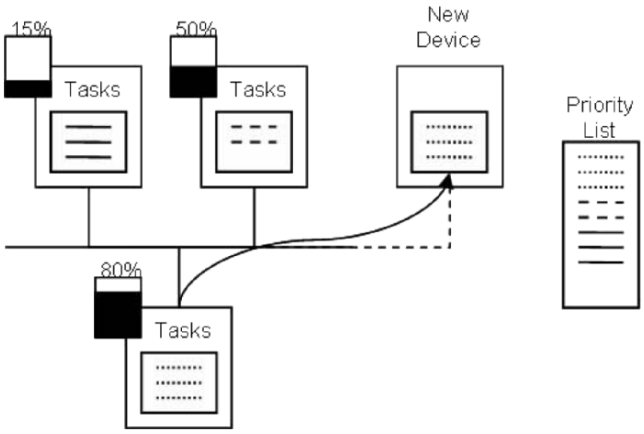

Figure 6. Most Loaded.

6.3 COST BASED MIGRATION

Within the cost based strategy the Load Balancer evaluates possible migration of tasks to the additional device. Migration is only a useful option if the cost of migrating is lower than the cost of keeping tasks with their original device. The cost benefit ratio for tasks of busy devices is computed which helps the Load Balancer to form the decision of whether to migrate or not. The calculation of migration costs of task is realized according to the priority list of the *Most Loaded* strategy.

7. CONCLUSION AND OUTLOOK

We presented a middleware architecture for automotive systems that enables dynamic load balancing. The integration of load balancing is a step towards a self-configuration within the vehicle. We focus on a specific use case scenario whereby an external device like PDA or mobile phone is added to the vehicle. With the help of the requirements, we described the middleware architecture and their enrichment with new services to support the distribution and exchange of tasks. Furthermore, we describe how traditional load balancing strategies could be applied within our approach.

Future work will be a detailed evaluation of the already existing load balancing strategies in the context of automotive systems. Additionally, the extension of existing or the development of new load balancing strategies will be done together with the implementation of the proposed architecture.

ACKNOWLEDGMENTS

This project was funded by the EU Commission within the project DySCAS (**Dy**namically **S**elf-**C**onfiguring **A**utomotive **S**ystems).

REFERENCES

[1] R. Athony, C. Ekelin, D. Chen, M. Törngren, G. de Boer, I. Jahnich, and et. al. A future dynamically reconfigurable automotive software system. In *Proceedings of the "Elektronik im Kraftfahrzeug"*, Dresden, Germany, 2006. Springer.

[2] R. Athony, A. Rettberg, I. Jahnich, C. Ekelin, and et.al. Towards a dynamically reconfigurable automotive control system architecture. In *A.Rettberg, R.Dömer, M.Zanella, A.Gerstlauer, F.Rammig. Proceedings of the IESS'07*, Irvine, California, USA, 2007. Springer.

[3] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. In *J. Parallel Distrib. Comput.*, 1989.

[4] R. Diekmann, B. Monien, and R. Preis. Load balancing strategies for distributed memory machines. In *In H. Satz F. Karsch, B. Monien, editor, Multiscale Phenomena and Their Simulation. World Scientific.*, pages 255–266, 1997.

[5] Y. F. Hu and R. J. Blake. An optimal dynamic load balancing algorithm. In *citeseer.ist.psu.edu/hu95optimal.html*, volume DL-P-95-011, 1995.

[6] Chi-Chung Hui and Samuel T. Chanson. Improved strategies for dynamic load balancing. In *IEEE Concurrency*, 1999.

[7] Balasubramanian Jaiganesh and Douglas. Evaluating the performance of middleware load balancing strategies. In *citeseer.ist.psu.edu/635250.html*, 2004.

[8] O. Othman and D. Schmidt. Optimizing distributed system performance via adaptive middleware load balancing. In *Ossama Othman and Douglas C. Schmidt, Optimizing Distributed system Performance via Adaptive Middleware Load Balancing, ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001), Snowbird, Utah, June 18, 2001.*, 2001.

[9] Ossama Othman and Douglas C. Schmidt. Issues in the design of adaptive middleware load balancing. In *LCTES '01: Proceedings of the ACM SIGPLAN workshop on Languages, compilers and tools for embedded systems*, pages 205–213, New York, NY, USA, 2001. ACM Press.

[10] S. Stoller. Leader election in distributed systems with crash failures. In *S. Stoller. Leader election in distributed systems with crash failures. Technical report, Indiana University, april 1997. 169*, 1997.

[11] S. van der Zwaan and C. Marques. Ant colony optimisation for job shop scheduling. In *S. van der Zwaan, and C. Marques. Ant Colony Optimisation for Job Shop Scheduling. Proceedings of the Third Workshop on Genetic Algorithms and Artificial Life (GAAL 99), 1999.*, 1999.