# DYNAMIC RECONFIGURATION OF A COMPONENT BASED SOFTWARE SYSTEM FOR VEHICLES

**Xuejun Chen**

*DaimlerChrysler AG*
*Research Information and Communication*
*RIC/TA, HPC T 728*
*D-70546 Stuttgart, Germany*
*xuejun.chen@daimlerchrysler.com*

Abstract: The *component system information and management architecture* is a component architecture to build the application software which is used in modern vehicles. The components are distributed in the system: the individual components are mainly located in different computers in vehicles, while the rest runs on computers in the infrastructure. The components can be dynamically loaded to different places of the component system, executed, removed and updated at runtime. In this article, we describe dynamic reconfiguration of a software component system for vehicles. Furthermore, we propose update strategies for updating components and component deployment strategies for optimizing the distributed system. By the use of dynamic reconfiguration, the flexibility and adaptability of the component based distributed system in vehicles are enhanced. In addition, the dynamic reconfiguration can improve the performance of the component system and reduce the costs of communication between a vehicle and the infrastructure. *Copyright© 2002 IFAC*

Keywords: Mobile Application, Dynamic Reconfiguration, Distributed System, Update Strategy, Component Deployment Strategy

## 1 INTRODUCTION

Modern vehicles are characterized by a steadily increasing amount of software. However, it is hard to decide which application software will be interesting for a customer through the lifecycle of a vehicle. Moreover, a vehicle in the future should not be regarded as an isolated individual system any longer, but should be able to be regarded as an active computation node in a worldwide computer network.

To meet these demands, we have designed a *component system information and management architecture* (Stümpfle, et al., 2000), which supports construction of component based application software. A component system, based on this architecture, is distributed. That is to say, most individual components are located in different computers in a vehicle, while other components run on computers in the infrastructure (see Figure 1) (Chen and Stümpfle, 2001). Such a component system represents at every point of time a certain configuration which describes all components participating in the system and the cooperation among them.

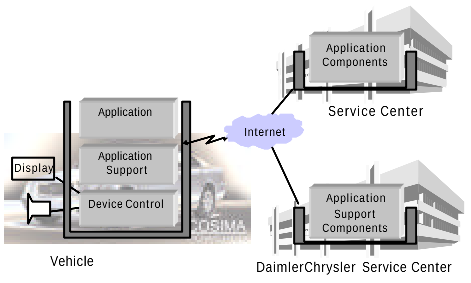

Fig. 1. Vehicle and infrastructure

The components in our architecture can be dynamically loaded into different places of a mobile component system, executed, removed and updated at runtime. With the dynamic loading functionality, the component system can load and keep exactly the amount of software components that are required by the customer. If an application component is not needed any longer, it can be removed from the system. If an application component becomes obsolete, the component can be updated by a newer version obtained from the infrastructure at runtime.

With dynamic reconfiguration the performance of the component system can be improved by usage of dynamic load balancing where the components can be moved to different places within a vehicle and into the infrastructure. At the same time, communication costs between a vehicle and the infrastructure can be reduced. Communication costs emerge when a vehicle communicates with the infrastructure, for instance, a vehicle gets current traffic data or a new version of a component from the infrastructure, or utilizes the high performance of the computers in the infrastructure.

The dynamically reconfigurable distributed component system needs an efficient and robust configuration management that serves to build the component system in such a way, that an optimal executable system can be formed, both statically and dynamically. Therefore, mechanisms of the dynamic configuration management have to be investigated. For example, why and how is a component migrated between a vehicle and the infrastructure? When a component is migrated or updated, which other components are influenced? How must the component system react to this change? In this paper, we will deal with the above mentioned questions.

This paper is structured as follows: the next section describes the structure of the configuration management. Section 3 represents a dynamic reconfiguration. In Section 4 the possibilities to optimize the system at runtime are described. The last section gives a summary of the main points and an outlook to the future.

## 2 STRUCTURE OF THE CONFIGURATION MANAGEMENT

Generally speaking, configuration management means all the activities that serve to build a distributed component system in such a way, that an optimal executable system emerges both statically and dynamically. The tasks of configuration management are static and dynamic configuration. A *static configuration* is a deployment of the components to the places where they function before the runtime. On the contrary, a dynamic configuration is the reconfiguration of deployment of the components at runtime. The distributed configuration managers execute the tasks of the configuration management.

The execution of the activities of the configuration management is distributed in the vehicle and the infrastructure. Some activities can only be carried out in the vehicle, for example, gathering the information about system status, finding out the requirements of users or adjusting the component system by updating components at runtime. The configuration manager who executes the activities in the vehicle, is called *local configuration manager* (LCM). There certainly are further activities that must be carried out in the infrastructure, such as offering an appropriate component from a component database, which is located in the infrastructure. The configuration manager who carries out activities in the infrastructure, is called *central configuration manager* (CCM). Furthermore, there are auxiliary configuration agents located in each runtime environment (see Figure 2).

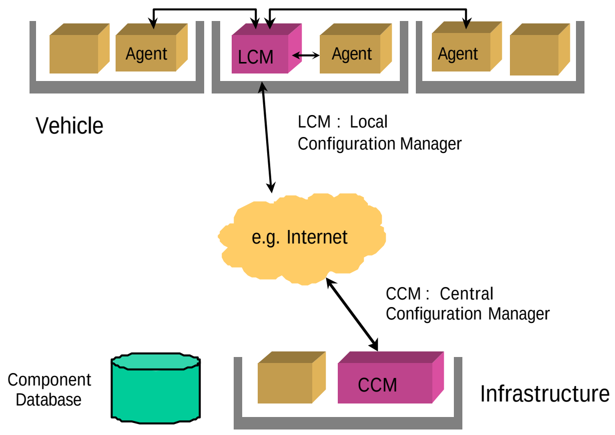

Fig. 2. Structure of the configuration management

### 2.1 Local configuration manager

The LCM manages the software components participating in a vehicle. It is responsible for a dynamic reconfiguration. The LCM analyses the state information of each computer which are collected by the agents (This entity will be explained in Subsection 2.2) and then decides, whether a reconfiguration should be carried out and, if necessary, initiates the reconfiguration process at runtime. Furthermore, the LCM finds out the requirements of users or applications. If the LCM detects either some requirements that cannot be satisfied by the current configuration or change of some conditions (e.g. the communication costs with the infrastructure), it will decide, how a reconfiguration should be performed. A reconfiguration will be carried out, if one of the following cases arises:

- The user needs a new service or information.
- A need for a component update at runtime occurs.
- A component should be migrated between the vehicle and the infrastructure, in order to reduce

the communication costs or to improve the system performance.

- Load balancing in the vehicle must be carried out.

### 2.2 Configuration agents

The LCM has an agent in each computer in a vehicle. Component loading mechanism is realized by agents for loading and unloading of components in a component system. In the current implementation, this mechanism is an extension of Java's ClassLoader. In cooperation with the LCM the agent controls the life cycle of each component in a runtime environment. The agent can initiate, start, stop a component and block or restart its interaction with a certain component. The status information in each runtime environment (e.g. processor load, free memory etc.) are gathered by agents and registered by the LCM. When a component is started, the agent registers it at the LCM. Therefore, the LCM always knows, which components are located in which runtime environment and what is their current status. The agent offers some services to components, for example, the naming service. Furthermore, the agent contains a component list and with this list it knows where each component is. The access to a component is location transparent. For example, if component *A* wants to communicate with component *B*, at first it has to ask the local agent, which is in the same computer, for a reference to component *B*. Then the component *A* can communicate with component *B* by an appropriate remote communication mechanism like Java RMI, or an event mechanism.

### 2.3 Central configuration manager

The CCM is located in the infrastructure. It administrates all the components in the infrastructure and the configurations in all the vehicles that have been registered in the infrastructure, so that the customers can get the personalized service comfortably. When the LCM needs a new component, it sends the corresponding requirement to the CCM. Thus, the CCM checks the current configuration of the vehicle and provides an appropriate component. When a new version of a component exists, the CCM informs the LCM, so that the LCM can update the component.

## 3 DYNAMIC RECONFIGURATION

The configuration management in our component architecture permits a dynamic reconfiguration at runtime. However, two problems arise during a dynamic reconfiguration:

- When a component is migrated or updated, which of the other components are influenced?
- How must the component system react to this change?

This section discusses these two problems.

### 3.1 Intercomponent dependency

A component can offer functionality to other components. We call such components *server components*. At the same time, a component can also require services from other components. A component that uses functions of other components is called *client component*. If a component needs functions of other components, we can say, it depends on other components. The dependency relation among components is described with a directed graph. In addition to the dependency on software components, a component can require certain hardware and other resources. The dependencies on other components, on hardware or on resources are called *dependencies of a component*.

The dependencies among components are stored by the LCM in an XML file. The dependencies, which can be previously defined, are called *static dependencies*. Since the component system is dynamic, the dependencies among concrete components can be dynamically produced and changed at runtime. Such dependencies are called as *dynamic dependencies* that are important for a dynamic reconfiguration. We describe this in Subsection 3.2.

The available component architectures (e.g. Enterprise Java Beans (Sun, 2000), DCOM/ActiveX (Denning, 1997) and CORBA Component Model (OMG, 1997)) offer only very little support for the management of dependencies. If dependencies among components are not clearly specified, it is difficult to shape a robust component system, especially for a dynamic component system. Therefore, a dependency analysis is necessary for shaping a component system. Without the dependency analysis, a component probably cannot work after its installation, or the other components perhaps cannot function after the installation of a new component, because their requirements can be not fulfilled anymore.

If the configuration management knows the dependencies among components at every point of time, it can support a dynamic reconfiguration of the system. For example, by updating a component at runtime, its dependencies have to be analyzed, so that it can be decided, which other components in the system will be affected, which new components must be loaded into the system, and which components must also be updated. If a certain dependency among the updated components exists, an update sequence of those components in a dynamic reconfiguration must be fixed. With the help of the dependency analysis, their update sequence can be correctly determined.

### 3.2 Dynamic reconfiguration

The dependency among the components can be dynamically changed at runtime. For example, component *A* needs a service offered by component *B*. Before component *A* calls a method of component *B*, *A* must ask the local agent for a reference to *B* with the function *agent.lookup(B)*. When *A* invokes a method on the server component *B*, the invocation dependency between the client component *A* and the server component *B* is registered by the middleware. After *A* has invoked a method on the server component *B*, the registration of the invocation dependency between *A* and *B* is removed by the middleware. That is to say, only when an invocation between two components is taking place, the both components are actually dependent. When the LCM decides to carry out a reconfiguration, it collects the dependency information by each agent and makes a global dependency graph. During a reconfiguration of a component we must decide, which components are actually dependent on the goal component.

Because of the consistency of a system, a component is not reconfigurable at arbitrary point of time. Similar to the definition of a reconfigurable state of a node in the work (Kramer and Magee, 1990), we define that a component is reconfigurable, only when the following conditions are fulfilled:

- Its clients initiate no new communication with it.
- The calls of its clients to it have been completed.
- It initiates no new communication with any component.
- Its calls to its server components have been answered.
- Its internal transaction has been terminated, if it initiated an internal transaction.

The following example describes, how a component is updated (see Figure 3). *B* is the target component, which is updated. *B'* is a new version of *B*. Component *A* is a client component of *B*. Component *C* is a server component of *B*.

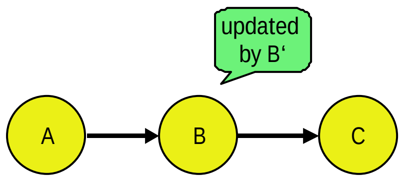

Fig. 3. Update of a component

The update process is described as follows:

1) install(B')
2) unlink(A, B)
3) unlink(B,C);
4) transferState(B, B')
5) link(A, B')
6) link(B', C)
7) remove(B)

However, how to deal with the interactions among components during a dynamic reconfiguration is a challenge. The interaction among the components must be able to be controlled, for instance, block and rebuild the interactions. After a component is reconfigured, the LCM notifies its client components to update the reference to this component. To monitor and treat the interactions among components during a dynamic reconfiguration, we have developed a middleware (Chen, 2002) that supports the dynamic reconfiguration.

Using a dependency management (Chen and Simons, 2002) we can exactly decide which components and which communication activities of those components are affected and therefore, only those communication activities must be blocked during a reconfiguration. This mechanism leads to a minimal interruption during the reconfiguration and at the same time, guarantees the consistency of the system.

## *3.3 Update strategies*

There are two kinds of dynamic updates. One is the update of a component during its execution. This kind of update causes high costs, since the execution of the concerned components has to be broken during the update. Despite of the high costs, the update of a component during its execution is necessary, if:

- it is necessary to replace a defective component during its execution;
- a new service is inserted in system, which requires new versions of other components;
- a component has several implementations of different suppliers. Each implementation has own advantages. In a certain case, the best implementation is applied;
- user wishes lead to an update.

The other kind is the update of a component also at runtime, but not during its execution. Thus this kind of update leads to the same low costs compared to the static update.

There are different update strategies. The update strategies decide when and which components should be updated. There are two extremes in the update strategies (see Figure 4).

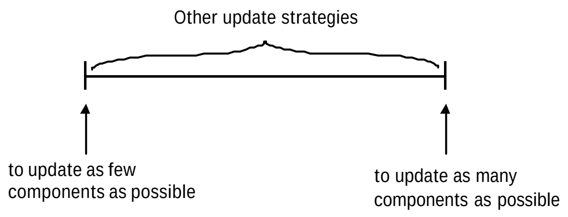
Other update strategies
to update as few components as possible
to update as many components as possible

Fig. 4. Update strategies

One extreme is "to update as few components as possible". In this case a component is updated, just when it is defective or it prohibits the system operation. We call this kind of update *lazy update*. The other extreme is "to update as many components as possible". In this case the components are updated, if they have new versions. This update strategy is called *eager update* here. With the consideration of the update costs, we use a flexible update strategy in our component system. During checking a configuration for an update, the configuration manager registers all components in the system which have a new version. Nevertheless, an update is not executed immediately, but:

- If the components that should be updated *are running* in the system, the strategy *Lazy Update* is applied.

- If the components that should be updated *are still* in the system, the strategy *Eager Update* is applied.

By the use of the flexible update strategy, we can update a component at runtime with as small cost as possible.

## 4 OPTIMIZING THE SYSTEM AT RUNTIME

With dynamic reconfiguration the performance of the component system can be improved by usage of dynamic load balancing, where the components can be moved to different places within a vehicle and into the infrastructure.

Before the execution of one component, the LCM looks for a suitable place for it with the consideration of load sharing. If the load of a computer exceeds its *load threshold*, then the agent informs the LCM and the LCM assigns no more new components to this computer. If the load of a computer exceeds its *overload threshold*, then the agent informs the LCM and the LCM shifts a component of this computer to another one. To relocate a component, the LCM chooses an underloaded computer and takes the costs of the communication between a relocated component and other components into consideration.

Costs of load balancing consist of the communication costs for collecting the state information of computers, the costs of component migration and process costs for execution of the load balancing algorithm. These costs must be taken into account, when the LCM makes decision, if a component should be relocated or not.

The components can be migrated between the vehicle and the infrastructure, in order to improve the performance and to reduce the communication costs. By deployment of components between the vehicle and the infrastructure the question is which components shall run in the infrastructure.

The aspects which should be taken into consideration when designing the strategies for the deployment of components between the vehicle and the infrastructure are the following:

- All components can be executed in the vehicle and thus the communication with the infrastructure is not necessary.
  - ➢ This is very advantageous for the user, however, the case is not very usual.
- If all components are executed in the vehicle, the performance is low.
  - ➢ The user should migrate the components to the infrastructure, in order to benefit from the high computing capacity there and to improve the system performance.
  - ➢ Communication with the infrastructure is necessary.
- The user wants to have current useful information from the infrastructure, such as traffic data or other useful data.
  - ➢ Communication with the infrastructure is profitable.
- An application must constantly communicate with the infrastructure. For instance, a diagnose application needs the support of a large database in the infrastructure.
  - ➢ The application should be migrated to the infrastructure.
  - ➢ If a component is migrated to the infrastructure and executed there, it does not mean that the communication between the vehicle and the infrastructure must always exist. The mechanism „disconnected operation“ (Noble, 1998) can be used to reduce the communication costs.
- A component in the vehicle must intensively communicate with an application that is executed in the infrastructure.
  - ➢ The component should be migrated to the infrastructure or its communication partner should be migrated to the vehicle.

If we use a suitable component deployment strategy, we can improve the system performance, and at the same time, reduce the costs of communication between a vehicle and the infrastructure.

## 5 CONCLUSIONS

This paper presented a dynamic reconfiguration of a distributed component system in vehicles. Such a reconfiguration guarantees the system consistency and minimizes the interruption to the system execution during the reconfiguration. Furthermore, we proposed update strategies for updating components of the system and component deployment strategies for optimizing the distributed system. By the use of dynamic reconfiguration, the performance of the whole component system can be improved and the costs of communication between a vehicle and the infrastructure can be reduced.

Based on the implemented basic mechanisms, the dynamically reconfigurable component system should be further researched and developed. First of all, the focus of the further research is the adaptation of components, where a context model should be introduced. The context influences the semantics of the information or the semantics of processing the information in some way (Heuer and Lubinski, 1996). Since the component system of a vehicle is located in a mobile and constantly changing environment, the characteristics of the component system that originate from mobility must be further investigated.

## ACKNOWLEDGMENTS

The author would like to thank Viktor Friesen for his valuable comments and proofreads.

## REFERENCES

Chen, X. and Stümpfle, M. (2001). Dynamic Configuration Management of a Telematics System for Vehicles, 1st IFAC Conference on Telematics Applications in Automation and Robotics, Weingarten, Germany.

Chen, X. (2002). Extending RMI to Support Dynamic Reconfiguration of Distributed Systems, accepted by the 22nd International Conference on Distributed Computing Systems (ICDCS 2002), Vienna, Austria.

Chen, X. and Simons, M. (2002). A Component Framework for Dynamic Reconfiguration of Distributed Systems, accepted by the First IFIP/ACM Working Conference on Component Deployment (CD 2002), Berlin, Germany.

Denning, A. (1997). ActiveX Controls Inside out, Microsoft Press, second edition.

Heuer, A. and Lubinski, A. (1996). Mobile Information Access Challenges and Possible Solutions, http://www.informatik.uni-rostock.de/~lubinski /artikel, In: Proc. IMC'96, Rostock, Germany.

Kramer, J. and Magee, J. (1990). The Evolving Philosophers Problem: Dynamic Change Management. IEEE Transactions on Software Engineering, SE-16, 11, pages 1293-1306.

Noble, B. (1998). Mobile Data Access, PhD thesis, School of Computer Science, Carnegie Mellon University.

OMG (1997). CORBA Component Model RFP, Object Management Group, http://www.omg.org

Stümpfle, M., Hermes, F., Friesen, V., Müller, F., Chen, X., Bachhofer, S., Jiang, D., Ly, K., Gauger, G. and Stiess, P. (2000). *COSIMA* – A Component System Information and Management Architecture, IEEE Intelligent Vehicles Symp osium, Dearborn, MI, USA.

Sun Microsystems (2000). Enterprise JavaBeans, http://java.sun.com/products/ejb/index.html