ANTS

A Generic Software Architecture
Applied to Driver Assistance Systems

Steffen Görzig
DaimlerChrysler AG
Research and Technology; Software Architecture
steffen.goerzig@daimlerchrysler.com

Abstract. A generic software architecture is useful for driver assistance systems in many ways. First of all it adds flexibility at different system levels. Second, it allows to save space for hardware - which is remarkable for a software architecture. And, as a result of the first two points, a good architecture also helps to reduce costs.

The described architecture is a generic software architecture for multi-agent systems: the "Agent NetWork System" (ANTS). The design description of ANTS includes the core system, tools for component and application development and how ANTS was applied for driver assistance systems.

1 Introduction

Tomorrows driver assistance systems will get more and more complex from the developers point of view. The growing number of sensors (encoder, radar, cameras, etc.) and perception modules as well as the fusion and integration of multiple assistance applications increasingly require concepts to cope with this complexity.

This problem is challenging not only from the hardware but also from the software architecture point of view. The software architecture has to deal with several modules running in parallel, coordinate the different applications and adapt the system to the current environment. The architecture must also consider portability, scalability, reuse and maintenance.

This paper describes a generic software architecture for multi-agent systems. The "Agent NetWork System" (ANTS) allows to realize multi-agent features such as cooperation, dynamic reconfiguration, autonomy, and concurrency. But moreover it has the ability to integrate problem specific components - this allows to adapt the system to given application domains.

The application domain described here are driver assistance systems. ANTS was used as the software architecture in the experimental vehicles UTA (Urban Traffic Assistant) and UTA II. Within these experiments a wide range of driver assistance systems were developed: systems for driver information, warning systems, and systems for autonomous driving. Scenarios for these systems are highway traffic and - for the first time in history of autonomous driving - urban traffic. The results show that the multi-agent approach is able to handle complex requirements for this kind of applications, including the integration of several sensors, data fusion, and hierarchical decision engines.

The paper is structured in two main parts. The first part describes how a generic software architecture is useful for driver assistance systems. That includes topics like flexibility, hardware space consumption, and cost reduction. The second part describes the architecture itself: the design of the system core, development tools, and applications.

2 Software Architecture and Driver Assistance Systems

Usually driver assistance systems are designed as standalone systems. Systems like autonomous stop&go driving, a speed limit assistant, or a lane departure warning system all have their own
sensors, actuators, and ECUs (Electronic Control Units). This has some advantages: the systems
do not directly depend on other systems (this reduces complexity) and responsibilities for failures
are clearly defined.

But on the other hand, standalone systems are not optimal in many ways: they lack of flexibility,
e.g. to use existing hardware such as sensors for additional features. The increasing number of
hardware causes space problems in the car and shifts the complexity from system development to
system integration, i.e. from the suppliers to the car manufacturers. For example conflicts between
two or more systems (e.g. for actuator accesses) cannot be modelled and solved in a uniform way
for the integrated system. And there are also indicators that standalone systems are not optimal
concerning costs.

The first idea one can have is to decompose the assistance systems into smaller software and
hardware components. These components with clearly defined interfaces can afterwards be re-
composed to different assistance systems. This allows to reuse existing software and hardware.

But to have components is only the first step. The next step is to use these components in a
structured way and to support the development of assistance systems itself. This reinforces the
need to introduce a software architecture, which can deal with the following requirements:

– different abstraction levels of action/perception/control,
– sensor fusion,
– integration and cooperation of various software modules,
– economical use of resources,
– scalability for software and hardware modules,
– reuse of software, and
– distributed computing.

The next sections will explain how such a software architecture helps to optimize the use of
components.

2.1 Flexibility

Driver assistance systems can be bound to specific situations. For example a speed limit assistant
is very useful on highways whereas an intersection assistant helps to recognize red traffic lights in
cities. The software architecture supports the reconfiguration from one application to the other,
i.e. it allows to switch applications in real-time depending on the current situation.

It is not only possible to switch between applications. It is also possible to adapt applications
to the current situation during runtime. For example obstacle detection in cities is different to
obstacle detection on highways: the velocity of the obstacles is lower, but the number of obstacle
types is much higher in cities. The software architecture allows to adapt components of applications
during runtime e.g. by adjusting filters or by using sensor fusion.

In contrast to hardwired systems a generic software architecture allows to add new components
without modifications of the existing ones. This makes enhancements of applications and the
development of new applications much easier.

2.2 Space

The reconfiguration of assistance systems according to the current situation as described above
has a remarkable effect: it allows to save hardware space. The software architecture allows to share
the same resources for different assistance systems. This includes not only sensors and actuators
but also ECUs. Now it is possible to perform computational optimizations for a set of assistance
systems which results in less hardware and therefore less space consumption.

2.3 Costs

The indicators for cost reductions are derived from the two previous points. A flexible architecture
allows to reuse existing components and enhance the system with new components. This results
in faster modifications of existing, and faster development of new applications. Saving space has
several advantages: the space can be used for other systems and fewer ECUs mean lower costs.
3 ANTS

Several software architectures were proposed for driver assistance systems [1][2][3][4][5] (for a comparison see [6]). We have pursued a multi-agent system approach to meet the requirements described above. Agent software is a rapidly developing area of research. Since heterogeneous research directions are summarized under this term, there is no consensus for a definition of the word “agent”. A working definition of a multi-agent system (MAS) can be defined as

"a loosely-coupled network of problem solvers that work together to solve problems that are beyond their individual capabilities"[7].

With this definition it is not difficult to see how an MAS can be used for driver assistance systems. Each component contains some functionality which can be useful for assistance systems. The capabilities of these components have to be combined to allow more complex applications.

The “Agent NeWork System” (ANTS) makes an explicit distinction between components (e.g. obstacle detection, vehicle control, etc.) and the connection of components to perform a specific application (e.g. stop&go) [6]. Since the interfaces of the components are fixed, ANTS is a generic architecture [8]:

“A generic architecture can be thought of as a fixed frame with a number of sockets where we can plug in some alternative or extension components. The components and sockets must clearly specify their interfaces, i.e. what they expect and what they provide. In other words, a generic architecture has a fixed topology and fixed interfaces.”

3.1 Core

The ANTS core defines the interfaces of components and how components can be combined to applications. The architecture consists of three layers. The first layer encapsulates the operating system and the hardware drivers. This allows to use different hardware and operating systems.

The next layer defines the core services of ANTS. This includes the coordination of execution (e.g. defining response times, parallel computing, and the priority ceiling protocol). A distributed database is used to store I/O data. The database allows transparent, parallel access without deadlocks[9] [10]. The communication service includes support for different kinds of networks (CAN, TCP Sockets, Myrinet) and the configuration and adaptation service provides the flexibility described above.

The third layer is the application layer. The interface for application development consists mainly of three components:

**Database Object** I/O data for information exchange (e.g. obstacles found by a radar device). This data is typically on a symbolic level.

**Functional Entity** Data processing units (e.g. obstacle detection, traffic light detection, or controllers for actuators).

**Administrator** System reconfiguration during runtime (e.g. switching from highway to city).

3.2 Tools

As described, ANTS distinguishes between the task of developing components and the task of connecting components to perform specific applications. Both parts are supported by tools.
The tool “AntsrGen” (see figure 1) is a graphical user interface (GUI) for ANTS components. It allows to design new components and their interfaces. Source code is generated from this graphical design of a component. “AntsrGen” also administrates existing components. It is possible to edit properties of components. Moreover, round trip engineering is supported: the interface of an existing component can be changed even if a developer has added source code to the component “by hand”. Hand written code and generated code are merged by the tool.

The “Modeller” (see figure 2) is a GUI to combine components to applications. This tool allows to specify a distributed system by defining sequences or parallel sets of components. The components can be configured (e.g. properties and response times) and the tool supports the user by providing standard configurations and by validating the correctness of the application (e.g. when a database object is read by a component but never written).

3.3 Applications

The ANTS software architecture was used in the experimental vehicles UTA (Urban Traffic Assistant) and UTA II. Various driver assistance systems were developed. This includes systems for highways and - for the first time in the history of autonomous driving - urban traffic:

stop&go Follow a vehicle in front autonomously. Switch off when the leading vehicle leaves its lane or a stop sign is detected.
speed limit assistant The driver is provided with information about the current speed limit.
lane departure warning Leaving the lane on highways causes a warning signal.
lane keeping Automatic lateral control on highways.

4 Conclusion

Driver assistance systems will be used to handle increasingly complex driving tasks in the future. The growing complexity of these systems will require software architectures, which can cope with requirements like sensor fusion, economical use of resources, and distributed computing.
This enforces a change from standalone systems to more modular systems. Components are only the first step. The next step is a generic software architecture using these components.

The described "Agent NeWork System" shows that multi-agent systems can be applied as a generic software architecture for advanced driver assistance systems. The advantages compared to standalone systems are flexibility, less hardware space, and reduced costs.

References