TOWARDS A DYNAMICALLY RECONFIGURABLE AUTOMOTIVE CONTROL SYSTEM ARCHITECTURE

Richard Anthony[1], Achim Rettberg[2], Dejiu Chen[3], Isabell Jahnich[2], Gerrit de Boer[4], Cecilia Ekelin[5]
[1]*The University of Greenwich, England*
[2]*University of Paderborn/C-LAB, Germanyy*
[3]*Royal Institute of Technology, Sweden*
[4]*Bosch GmbH, Germany*
[5]*Volvo Technology AB, Sweden*

Abstract: This paper proposes a vehicular control system architecture that supports self-configuration. The architecture is based on dynamic mapping of processes and services to resources to meet the challenges of future demanding use-scenarios in which systems must be flexible to exhibit context-aware behaviour and to permit customization. The architecture comprises a number of low-level services that will provide the required system functionalities, which include automatic discovery and incorporation of new devices, self-optimisation to best-use the processing, storage and communication resources available, and self-diagnostics. The benefits and challenges of dynamic configuration and the automatic inclusion of users' Consumer Electronic (CE) devices are briefly discussed and the self-management and control-theoretic technologies that will be used are described in outline. A number of generic use-cases have been identified, each with several specific use-case scenarios. To demonstrate the extent of the flexible reconfiguration facilitated by the architecture, some of these use-cases are described, each exemplifying a different aspect of dynamic reconfiguration.

Key words: Automotive Systems, Middleware Architecture, Dynamic Reconfiguration, Self-Configuring.

Anthony, R., Rettberg, A., Chen, D., Jahnich, I., de Boer, G., Ekelin, C, 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. 71–84.

1. INTRODUCTION

DySCAS (Dynamically Self-Configuring Automotive Systems) is a European Commission funded project that started in June 2006. DySCAS targets an automotive system that emphasizes the dynamic self-configuration of embedded systems. Unlike the common static configurations, the main goal of the project is the development of next-generation technologies based on existing solutions, namely the development of an intelligent automotive networked middleware system. This platform will have properties that include a high degree of robustness and fault-tolerance due to its self-adaptability at run-time. The cooperation of integrated system objects, primarily automotive ECU's and connected CE devices, results in a dynamic system that adapts itself to its changing environment and conditions.

2. WHY DYNAMIC RECONFIGURATION

The current state of practice is to embed several electronic control units (ECU) into a vehicle. These ECUs are typically based on proprietary hardware and software components and usually have specialized and fixed functionally. A vehicle may have several different ECUs, but since each one has a different non-transferable function, more devices means higher susceptibility to failure.

Upgrades are difficult and expensive; once a vehicle leaves the factory it is not easy to change its functionality, for example to apply the latest engine management configuration to improve fuel economy. If a serious fault occurs a manufacturer may have to undertake an expensive recall because vehicles cannot be easily upgraded in the field.

Owners have increasingly high expectations for infotainment services on the move, for example navigation devices are now common, and some treat their car as a mobile office. Therefore there is significant benefit to be gained by facilitating a three-way dynamic and automatic cooperation between consumer electronics devices that the owner brings into the vehicle, the embedded computing devices of the vehicle itself, and external networks that the vehicle passes within wireless range of.

Such a three-way setup enables an exiting and innovative suite of new applications that can: use location dependent information; share the processing power and other resources of several devices; achieve fault-tolerance by dynamic role allocation; and automatically collect and filter information (news, traffic, entertainment) in context-specific ways depending on who is in the vehicle, where they are and where they are travelling to.

A paradigm shift towards software-centric and network-centric ECU development is occurring. Software is becoming the dominant component of electronic automotive features. New opportunities are arising in the automotive electronics sector with networked and collaborating ECUs, which form an "automotive control grid" in which software components are no longer bound to specific hardware components, but can be executed on various units within the vehicles, and potentially even migrate during run-time. In this way each device can act as on-line replacement for other ECUs. In this scenario more devices means lower susceptibility to failure.

To maximise benefits from this opportunity, proprietary solutions will give way to standardised HW/SW component architectures and infrastructures. This provides developer freedom, better interoperability and service re-use, ultimately leading to lower development cost and higher reliability.

There is a lot of research into the migration of tasks from software to hardware. For example for hardware acceleration of image and video processing, specialized DSPs and FPGAs are used. Reconfigurable hardware/software based architectures are very attractive for implementation of run-time reconfigurable embedded systems. The hardware/software allocation of applications tasks to dynamically reconfigurable embedded systems (by means of task migration) allows for customization of their resources during run-time to meet the demands of executing applications, as can be seen in [1, 2]. DySCAS will focus mainly on the reconfiguration or self-configuration of software tasks in the middleware.

The AUTOSAR effort [3] has the purpose to provide an increasing flexibility with respect to the design time allocation of functions to different ECUs, by standardizing the software infrastructure and its services. Although this would be a large achievement, the AUTOSAR approach does not support dynamic system reconfigurations.

3. FUTURE VEHICULAR CONTROL SYSTEMS

The future vehicular control system takes the form of an embedded network system. This will comprise a wired core-network of processing nodes and sensor nodes spread throughout the vehicle. There will also be opportunities to incorporate mobile devices such as cell-phones, PDAs and other devices carried by the vehicle occupants. Such devices may have external connections to cellular networks and may also form ad-hoc networks between themselves. The automotive system will attempt to take advantage of the processing resources and connectivity of these devices when they are present.

4. SOFTWARE TECHNIQUES FOR SELF-CONFIGURATION

Context awareness and the ability to adapt behaviour to suit current environmental conditions are key requirements for projects such as DySAS. Configurational complexity arises from the number of components, the interactions between these components and between the components and their environment. This can be a barrier to achieving optimal or efficient performance and can be highly problematic for externally-manged run-time configuration [4, 5].

In answer to the complexity problem, the autonomic computing paradigm [6] advocates self-adaptation in which applications modify their behaviour to suit their environment and context, see for example [7] and [8]. There are many different types of adaptation [9], including self-configuration, self-healing, self-optimisation and self-protection [10, 11]; collectively referred to as 'self-management'. Self-management implies that the system is able to adjust some aspect of its own behaviour. When the adjustment is made to enable a new configuration to be supported, for example the dynamic incorporation of newly discovered devices the adjustment is termed 'self-configuration'. Adjustments to improve performance or efficiency are more-specifically referred to as self-optimisation. When the adjustment is made to mask or recover from a component failure it is referred to as self-healing. Adjustments made to deal with threats, such as dynamically detecting a denial-of-service attack by recognizing a 'pattern' in the requests to the system, are classed as self-protection.

The applications that populate the automotive system will have 'pervasive' and 'ambient' qualities. This concept involves embedding context-aware and location-aware services into infrastructure and portable devices. Ideally such services are so embedded that the user is not even aware of it [12]. To support this ideal, self-management moves the emphasis away from manual configuration of components, towards inbuilt learning and discovery capabilities [9, 10].

The Control-Theoretic approach for automatic control is receiving increasing attention [13] [14]. This approach has many characteristics which make it suitable for adoption within an autonomous computing framework: 1. Sensing of the system behaviour and model-based estimation of other interesting (and not measurable) system states; 2. Control based on feedback from sensed and estimated system states. Feedback control provides robustness to system disturbances and to uncertainties about the actual system behaviour; 3. Feedforward control, based on system models, to improve the system behaviour in response to known changes in the environment and changes of the desired behaviour; 4. Systems identification

and adaptive control as means to develop the models of the controlled environment, during design time and run-time.

Computer systems are traditionally modelled using e.g. queuing theory or other discrete-event formalisms. The dynamics differ where central ingredients include delays (due to queues) and dynamics are introduced through sensors. Despite this challenge, recent advances indicate that there is a lot to learn from control engineering. Combinations of feedforward and feedback control are useful and it seems that simple controllers are often sufficient.

Example application uses of a control approach include load balancing and QoS optimization. Typical application areas include multimedia, web storage systems, and network routers [15]. For embedded control systems, issues covered in the research include for example feedback-based resource scheduling, RTOS and middleware support for control activities, dynamic models of real-time systems, control and real-time co-design, and integration of quality-of-service and resource management [16].

5. A DYNAMICALLY RECONFIGURABLE ARCHITECTURE

The architecture needs to be capable of flexible and automatic reconfiguration in order to facilitate the technological advances targeted by the project. These advances include:

Automated fault detection, analysis and reporting: Trends in fault patterns can be learnt, and solutions devised. Over the longer-term solutions can be re-used.

Automated resilience through software relocation: Once a hardware or software fault has been identified it will be possible to automatically relocate the required functionality to another device.

Dynamic reconfiguration based on current resource demands: Processing power and storage capacity of ECUs and CE devices is typically quite limited. It will be possible to dynamically reconfigure resource provision and usage.

Software downloads of plug-and-play components: When passing through hot-spots, wireless connectivity to external systems will be automatically utilized to download new components and services, as well as for uploading fault status reports if necessary.

Automatic support for both push and pull software patching: The external connectivity permits automatic pushing of software revisions from the manufacturer side and faulty software needing replacement can be pulled by a specific system.

Sporadically available resources, as provided by a user's mobile device or accessible via a global network, will be included seamlessly into the vehicle's electronic system. The resources of a PDA or notebook could, for example, be used to improve the rendering of a 3D-view for navigation directions. The middleware needs to provide services that handle the availability and integration of such resources.

Simplification and standardisation of the software developer role: The provision of common interfaces to hardware and software components leads to a set of open standards. This avoids the complexity of having to deal with a multitude of different formats and thus removes the burden of interoperability from developers.

The core of the DySCAS architecture will be a self-managing middleware containing a number of services able to be dynamically composed to provide functionalities required by a wide range of use-cases (see next section).

Services identified so far include:

- Discovery (of location, of physical devices and of the services / capabilities provided by those devices),
- Interface provisioning / negotiation (brokering of service agreements between components),
- Resource mapping (dynamic determination of which resources should be allocated to which services),
- Security (authentication of devices offering or requesting services or resources),
- Storage management (especially when downloading SW; keeping track of versions),
- Rollback management (if a SW update fails, or an untested component configuration arises, a previously stable configuration must be reinstated),
- Reliable download (of SW, configuration parameters, data, or policy),
- SW / policy installer / upgrader (must ensure that the upgrade is performed at a safe moment, and if unsuccessful must invoke the rollback manager),
- Error management (dynamic fault detection, and on-board diagnostics),
- Migration of service (moving code and or data to balance load and to recover from faults),
- Data logging (sensor data is collated for subsequent diagnostics),

- Dynamic service prioritization (based on context, to support gradual degradation in the presence of HW or SW faults, battery-low condition etc.),
- SW / HW reconfiguration (low-level service to facilitate migration of service).

By providing these services the DySCAS middleware facilitates run-time reorganisation to balance workloads, expedite urgent processes, and to reconfigure components for survivability despite hardware failures. Components will also be able to automatically discover new components and establish service level agreements based on the resources and services that they are able to provide to each other.

The DySCAS architecture is specified in three stages: the System Architecture Framework (SAF), the System Architecture (SA) and the System Architecture Specification (SAS), see figure 1.

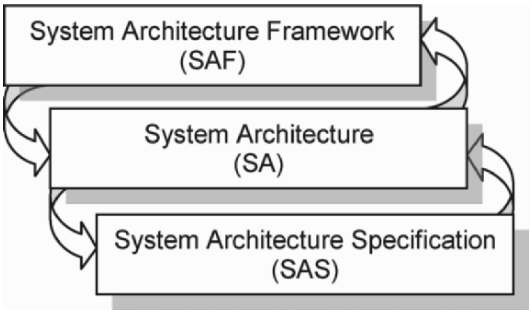

Figure 1. Specification of the DySCAS architecture.

The SAF identifies the system contents and provides the concept at a high-level. The list of system services is also included in SAF. Additionally, the SAF defines the notation for the SA and gives a description of the working process (i.e. how the components / services will interact, composition etc.). Together with the list of available technologies, the SAF defines the draft abstractions for the communication, the used operating system (OS), the underlying hardware resources, possible component models and software patterns. Therefore, the SAF covers all aspects of the implementation constraints.

The system structure (the applied APIs and components) are provided in the SA. Here protocol state machines are used for the description. This is more an abstract description of the functionalities and the APIs, but without a formal model. The formal model for this purpose is defined in SAS.

Components and APIs require an interface with attributes, properties and also timing information. This is also described in SA.

The SAS is as generic as possible to be open and easily adaptable and extendable. The specification described in an ADL provides the classification of services. Here the formal model for the system including an abstract description of the functionalities, APIs and the interfaces are given. Besides this, QoS assumptions and requirements for the used OS are defined. Therefore, the development of an abstract bus description, a so-called virtual function bus, including the communication properties to enable a design is necessary. To summarize, the SAS provides the user of the DySCAS architecture with guidelines for the implementation. This includes also the tool environment. Therefore, the SAS serves as documentation for application developers.

6. USE-CASES

The technologies under development within DySCAS are based on requirements derived from a rigorous analysis of future use-cases. A model describes three levels of use-case, in which generic use-cases represent groupings of specific use-cases requiring similar subsets of functionalities. These are atomic functionalities which can not be further broken down. Functionalities, in turn, map onto one or more services. A service may directly configure resources (in the technology layer) or may require further assistance from, or delegate to, other services. This structure (shown in figure 2) is important in the derivation of services and technology that form the DySCAS architecture, which has to be simultaneously very flexible and highly efficient.

One example of a system functionality is the function which enables a device (for example a smartphone) to connect to the car information systems. System functionalities are abstracted in the sense that they describe the functional requirements but not how they will be realized. The actual realization (below the horizontal line in figure 2) is done by defining services which will be implemented on top of the technical components.

Each of the use cases considered consists of a sequence of systems functionalities. When considering different use cases it was realised that groups of use cases exist which are each built up by a similar suite of system functionalities. To reflect this generic use cases are defined. The generic use cases identified so far are:

- New device automatically attached to the car,
- Integrating new software functionality,

- Closed re-configuration.

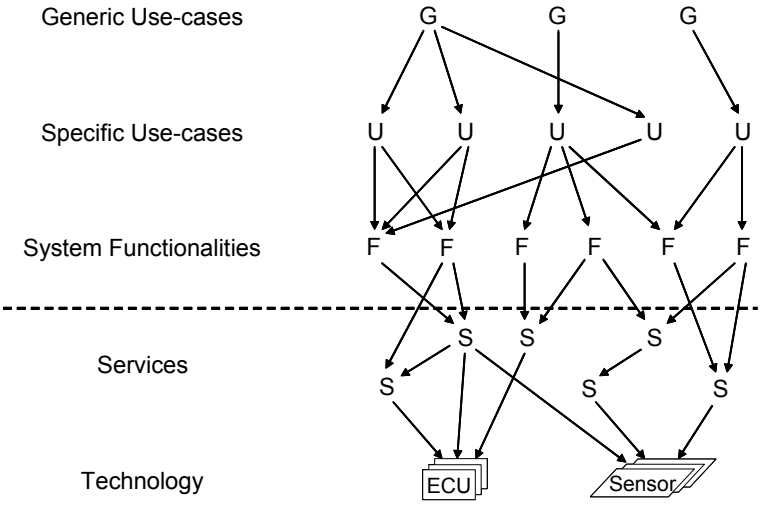

Figure 2. Mapping Use-cases onto DySCAS Services and Technology.

a) New device automatically attached to the car

If an appropriate new device is detected by the vehicle system it is automatically attached to the vehicle IT systems. This can be a mobile device brought into the car such as a PDA or smart phone, but it can also be a Hotspot outside of the car.

In these use-cases the information system of the vehicle is complemented by another device which may open access to external services, possibly extending the functionality of the system by increasing computing resources such as memory, computing capabilities etc, or it may act as a new source of data (e.g. an MP3 player). The task is to integrate the new device seamlessly and for the driver as transparently as possible.

Specific examples of the generic use-case include:

- Accessing a navigation system on mobile devices,
- Connecting a trailer and incorporating the computing and sensor systems the trailer is hosting,
- Internet access via an attached mobile phone (smart phone),

- Sending/receiving traffic data via an attached mobile phone,
- Streaming of entertainment data (MP3, video) from the mobile phone to the vehicle's entertainment system,
- Extracting the destination from the calendar application of the mobile device,
- Vehicle connecting to a wireless network hotspot,
- Determination of location, affecting context. Such as detecting the vehicle has entered the proximity of the owners workplace.

System functionalities include:

- SW is migrated to new HW,
- Wireless connection is established,
- Capabilities are negotiated,
- Authentication and all security mechanisms needed to protect the system.

b) Integrating new software functionality

This generic use-case covers all specific use-cases in which new software is included in the system. Software includes application software as well as operating software. The system must integrate the new software patches and ensure that the new configuration works properly. If not then the old software configuration must be reinstalled. This must be done as seamlessly as possible for the driver and it must not interrupt or impact in any way driving. Therefore the reconfiguration must be performed only in absence of safety related situations.

Examples of system functionalities are:

- Self-test,
- Security features,
- Reconfiguration.

Specific Use Cases include:

- Software download from hotspots,
- Software download from storage media (DVD),
- Replace system functionality,
- Replace applications (e.g. navigation, MP3 software players).

c) Closed re-configuration

In contrast to the other generic use-cases the closed re-configuration includes all use cases which perform any kind of reconfiguration which is not initiated by external events or by the integration of external hardware or software, but is performed to make the system work better or react to unexpected situations. To detect re-configuration opportunities, all components of the system must be monitored. Free devices or devices which do not show expected behaviour must be detected and well working configurations must be identified, migration must be planned and executed.

Specific use case examples are:

- Graceful degradation in case of power problems,
- Efficient usage of redundancy, e.g. ECUs which are used temporally, such as the (normally dedicated) security/immobilizer ECU, unused once the vehicle has been started.
- Migration of services in the case of HW failure.

System functionalities include:

- Detection, monitoring features,
- Re-configuraton features,
- Self-testing / diagnostics.

By means of an illustrated example, consider that an ECU failure is detected. The affected service must be migrated to an alternative ECU. If none are available, running services must be prioritised, so that an ECU can be made available by shutting down some services. This use-case is ‘Migration of services in the case of HW failure’ which is in the ‘Closed re-configuration’ generic class of use-cases. The composition of services needed to achieve the reconfiguration is illustrated in figure 3.

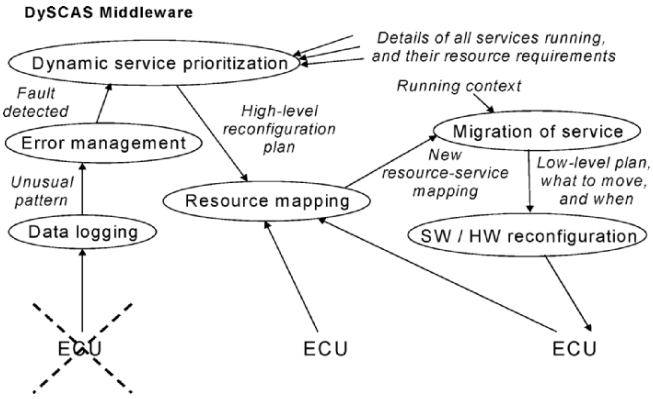

Figure 3. Composing DySCAS services to satisfy the 'Migration of services in the case of HW failure' use-case.

7. CONCLUSION

This paper comes at an early stage in the DySCAS project, and as such, it represents our preliminary design solution for self configuration. The project targets an ambitious collection of use-cases, requiring dynamic adaptation and thus the proposed middleware must provide a number of new services in a flexible and efficient manner.

Due to the timing of this paper, the architecture has been described at a high level. Some specific services have been identified and this set will stabilize as the use-cases are scrutinized during the next stage of the project. It is important that the services are clearly defined and differentiated to avoid duplication and to ensure that flexible composition of these services is possible.

Numerous use-cases have been identified at two levels, generic and specific. This approach facilitates identification of the required finctionalities, whilst once again avoiding duplication. Some of the use-cases have been discussed in outline.

8. NEXT STEPS

Now that the basic architecture has been determined, it needs to be refined. Further detail will be added by analyzing the use-cases in more detail and also by examining the features (capabilities, performance, resource constraints, interface issues) of the various platforms (ECUs, sensors etc.) and mapping the services onto these platforms.

A demonstrator application is under development; this will eventually showcase some of the use-cases in a real-time emulation. Further details of the project are available at the DySCAS website, see [17].

ACKNOWLEDGEMENTS

The authors would like to thank all the people and partners who are involved in the DySCAS project. Without their contributions and the fruitful discussions inside the project this paper could not have been written.

The DySCAS project is funded within the 6th framework programme "Information Society Technologies" of the European Commission.

REFERENCES

[1] Harkin, J., McGinnity, T. M., and Maguire, L. P. (2004). Modeling and optimizing run-time reconfiguration using evolutionary computation. Trans. on Embedded Computing Sys., 3(4):661–685.

[2] Götz, Marcelo, Rettberg, Achim, and Pereira, Carlos E. (2005). Towards Run-time Partitioning of a Real Time Operating System for Reconfigurable Systems on Chip. In Proceedings of International Embedded Systems Symposium - IESS, Manaus, Brazil.

[3] AUTOSAR – Automotive Open Systems Architecture. Available at http://www.autosar.org

[4] M. Ibrahim, R. Telford, P. Dini, P. Lorenz, N. Vidovic and R. Anthony, Self-adaptability and Man-in-the-Loop: A Dilemma in Autonomic Computing Systems, 2nd International Workshop on Self-Adaptable and Autonomic Computing Systems - SAACS '04 (DEXA 2004), Zaragoza, Spain, September 2004, IEEE.

[5] R. Barrett, P. Maglio, E. Kandogan and J. Bailey, "Usable Autonomic Computing Systems: the Administrator's Perspective", Proc 1st Intl Conf Autonomic Computing (ICAC 2004), IEEE Computer Society, New York, May 2004, pp. 18-25.

[6] Kephart J. O, Chess D.M. The Vision of Autonomic Computing. Computer, IEEE, Volume 36, Issue 1, January 2003, pp. 41-50.

[7] Dolev, Shlomi and Kat, Ronen I, Self-Stabilizing Distributed File Systems, International Workshop on Self-Repairing and Self-Configurable Distributed Systems on the 21st Symposium on Reliable Distributed Systems, October 2002, IEEE, pp. 384-390.

[8] N. Badr, A. Taleb-Bendiab, M. Randles, D. Reilly, 'A Deliberative Model for Self-Adaptation Middleware Using Architectural Dependency', Proc 15th Intl Conf Database and Expert Systems Applications (DEXA 2004), August 2004, IEEE, pp. 752-756.

[9] S. White, J. Hanson, I. Whalley, D. Chess and J. Kephart, An Architectural Approach to Autonomic Computing, Proc. 1st Intl. Conf. Autonomic Computing (ICAC), IEEE, New York, May 2004, 2-9.

[10] IBM, An architectural blueprint for autonomic computing, IBM Autonomic Computing White Paper, http://www-03.ibm.com/autonomic/pdfs/ACBP2_2004-10-04.pdf

[11] J. Koehler, C Giblin, D. Gantenbein and R. Hauser, On Autonomic Computing Architectures, Research Report (Computer Science) RZ 3487(#99302), IBM Research (Zurich), 2003.

[12] Waldrop M, Autonomic Computing: The Technology of Self-Management, Woodrow Wilson International Center for Scholars, http://www.thefutureofcomputing .org/Autonom2.pdf, 2003.

[13] J.L. Hellerstein, Y Diao, S Parekh, D M Tilbury. Control engineering for computing systems - industry experience and research challenges. IEEE Control Systems Magazine, Volume 25, Issue 6, Dec. 2005.

[14] R Sanz, K-E Årzén. Trends in software and control. IEEE Control Systems Magazine, Volume 23, Issue 3, June 2003.

[15] K-E Årzén, A Cervin, T Abdelzaher, H Hjalmarsson, A Robertsson, Roadmap on Control of RealTime Computing System, EU/IST FP6 ARTIST2 NoE, Control for Embedded Systems Cluster.

[16] K-E Årzén, JPRA-NoE Integration: Adaptive Real-time, HRT and Control, Activity Progress Report for Year 1, IST-004527 ARTIST2 NoE: Embedded Systems Design.

[17] DySCAS website: www.dyscas.org