Electronic Communications of the EASST
Volume 19 (2009)

Proceedings of the Second International DisCoTec Workshop on Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services (CAMPUS 2009)

Context-Aware Adaptation in DySCAS

Richard Anthony, DeJiu Chen, Mariusz Pelc, Magnus Persson, Martin Törngren

15 pages

Guest Editors: Romain Rouvoy, Michael Wagner
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Context-Aware Adaptation in DySCAS

Richard Anthony[1], **DeJiu Chen**[2], **Mariusz Pelc**[1], **Magnus Persson**[2], **Martin Törngren**[2]

[1] University of Greenwich, Greenwich, London, UK
[2] Royal Institute of Technology (KTH), Stockholm, Sweden

Abstract: DySCAS is a dynamically self-configuring middleware for automotive control systems. The addition of autonomic, context-aware dynamic configuration to automotive control systems brings a potential for a wide range of benefits in terms of robustness, flexibility, upgrading etc. However, the automotive systems represent a particularly challenging domain for the deployment of autonomics concepts, having a combination of real-time performance constraints, severe resource limitations, safety-critical aspects and cost pressures. For these reasons current systems are statically configured. This paper describes the dynamic run-time configuration aspects of DySCAS and focuses on the extent to which context-aware adaptation has been achieved in DySCAS, and the ways in which the various design and implementation challenges are met.

Keywords: real-time embedded systems, dynamic configuration, context-aware behaviour, automotive control, middleware, policy-based configuration

1 Introduction

Distributed embedded systems are traditionally configured at deployment time, i.e. tasks are statically allocated to nodes and settings are fixed for the product's entire life time. It is usually not easy to upgrade the software or add new functions.

A key characteristic of computing systems in the automotive domain is precisely the difficulty in making efficient changes post-deployment during a long vehicle lifecycle for the purpose of software maintenance, customization and personalization, as well as incorporation of technology innovations. For example, it is known that a vehicle as a whole has in general a lifecycle three to ten times longer than its less tightly connected infotainment devices. This gap creates a tension and desire to upgrade both hardware and software of the infotainment devices and have them seamlessly integrated into the vehicle. Under current practice this can only be achieved by directly servicing each vehicle by suitably qualified personnel with specific equipment.

It is expected that future vehicles will offer the following abilities [ALE+06] : performing cost efficient and reliable field maintenance and upgrades of software; integrating external devices, communication with nearby vehicles and composition of their provided services; and allowing post-deployment time optimization of software configuration and resource deployment according to current environment conditions and internal status.

Vehicular control systems are increasingly complex and this complexity impacts on the already long design and development cycles. The automotive industry needs the ability to defer some design decisions so that time-to-market is reduced without compromising the level of functionality

achieved. Upgrades should be supported transparently throughout the lifetime of the vehicle. Vehicle owners' expectations of technology are also increasing and they demand the ability to make customization choices, and to change those choices over the vehicle's lifetime. Flexible vehicle upgrades are also indirectly demanded because vehicular legislation concerning aspects such as safety, emissions, noise etc. are constantly updated and applied differently at national levels; currently such legislation only affects new vehicles as it can not be retrospectively applied.

DySCAS (Dynamically Self-Configuring Automotive Systems) is a middleware technology that facilitates context-aware dynamic reconfiguration of automotive control systems. The deployment of self-management into vehicular systems has the potential to improve robustness e.g. through dynamic fault handling; to improve efficiency, e.g. through dynamic reconfiguration to reduce power consumption; and to make systems flexibly upgradeable and customisable.

This paper focuses on the goals of DySCAS, the need for context-awareness in a system such as DySCAS, the mechanisms by which run-time context-aware dynamic configuration has been achieved, and provides brief details of two of the implemented reference modules.

2 Related work

Context-awareness (CA) enables systems to behave in accordance to their environment and circumstances. This is of high topical importance in a wide variety of application domains; as such CA is a core theme of a number of recent and current projects. MUSIC [MUS] targets the development of an open platform technology supporting self-adaptive / CA mobile applications. MUSIC includes a methodology, tools and middleware. MIDAS [MID] is a middleware which is context-aware in the sense that it automatically adapts to changes in network topology. This is not only to compensate for problems (such as the failure of particular links) but also to exploit opportunities offered (such as when high bandwidth connections to central machines are available). inContext [inC] is investigating new techniques and algorithms for pro-active service aggregation, CA service adaptation and service provisioning, to enable dynamic collaboration. CONNECT [CONa] combines research in several areas including CA, to implement a privacy management platform for pervasive mobile services. CONSEQUENCE [CONb] is concerned with dynamic management policies supported by an architecture that is CA as well as secure and resilient. A Context Awareness Manager is the core component of the LATIS Pervasive Framework (LAPERF) [TN06]. The objective is to provide a framework, and automating tools to support developers of pervasive computing applications. The DySCAS project is differentiated from these other projects fundamentally in the sense that the CA logic itself, as well as the particular selection of context information used in dynamic decision making, is run-time changeable. This yields a highly flexible system such that the functionalities of its applications are not restrained by limited design time vision. [CY08] presents a CA intelligent system architecture for pervasive mobile computing applications. A 'context server' functions as a repository for application-specific context information. The concept of a context fingerprint (a characterisation of the context that a mobile device determines from its sensors) is introduced in [Joh07]. A framework is also provided to allow these 'fingerprints' to be associated with events and actions that are fired on transition between contexts. A Service-Oriented CA Middleware (SOCAM) architecture is presented in [GPZ05]. The architecture provides comprehensive support for building

context-aware services. A formal context model based on ontology is also proposed, to address CA-related issues including context reasoning, context classification and dependency. Similarly, [SWV07] describes a service-oriented middleware which provides a scripting-like method for CA application development; allowing the subscription of rules containing context-based events and conditions and a notification to be sent when the specified context holds. Vehicular control systems are advancing in several directions in response to the requirements of increasingly rich functionality and high configuration flexibility. A component based approach for managing the increasing complexity in modern vehicular systems is proposed in [BCSV08]. Components are used from early design through development and deployment, to support separation of concerns at different levels of granularity. An OWL-based context-model for abstract scene representation of driving scenarios to support driving assistance systems is described in [FRLK08]. The work presented in [FSSC08] is concerned with time-bounded adaptation for automotive system software. Requirements for dynamic software adaptation are identified and a taxonomy of various dimensions of dynamic adaptation in emerging automotive system software is defined. Embedded systems present a challenging arena for the development of adaptive, context-aware systems, primarily because of the special concerns of limited resources and the inflexibility to change (i.e. the difficulty of upgrading code). A resource-oriented embedded system design framework is presented in [KSC08]. Embedded system components are incrementally developed in both a resource-independent model (which is functionality oriented) and a resource-oriented model. The two models are constrained so as to be consistent with each other and in compliance with hardware behaviour. A design framework for real-time embedded systems is discussed in [LSK+08]. The framework takes into account the performance trade-off between code size, execution time, and energy consumption characteristics. The proposed technique generates design parameters such that system cost is minimized while the resource constraints are satisfied. [But06] Discusses some major research needs, to make the next generation embedded systems more predictable and adaptive to environmental changes. Problems with current approaches are identified, along with potential solutions from research in operating systems and scheduling.

3 DySCAS, and its goals

The introduction of context-aware dynamic reconfiguration to the automotive domain requires careful consideration of the technical challenges. These are identified as: **Application characteristics** – many automotive applications have QoS requirements that imply real-time scheduling and worst-case performance guarantees, including communication latency. **Long product lifetimes, long design lifecycle, and high-valued products**. Flexibility is needed to support functions and behaviours not perceived at design time – this is necessary to future-proof products by enabling upgrades at various levels (hardware, middleware components and application components, as well as strategic control logic within these components). This reduces the pressure of taking everything into account at design-time. **High robustness** – applications should continue to function despite localised failures. Systems can have safety-critical aspects and thus must be highly dependable. **Efficient resource utilisation** – in order that several applications can co-exist without QoS conflicts. **Support for a variety of platforms and resource configurations** – this is because the industry has not standardised in these aspects, because of the diverse

set of applications possible, and because future technologies will inevitably be blended in with current solutions. **Support for distributed processing** – the typical automotive control systems comprise a number of heterogeneous electronic control units spread throughout the vehicle.

The key goals and requirements of DySCAS have been derived from the technical challenges above. In addition to the directly mapped requirements (**real-time performance**, **scalability**, **platform independence**, **extensibility** and **functional upgrade**), these also include: **Self-management and automatic run-time configuration** - because the systems are highly dynamic (so cannot have fixed configuration), and are too complex for users to configure manually in a timely way. **Context-awareness** - dynamic adaptation and configuration must take account of current operating conditions and system environment. **Validatable behaviour** - despite dynamic configuration and adaptation, the deployed systems must be predictable and must be trusted to remain within allowable bounds of behaviour and to remain stable despite perturbations.

The following scenario puts several of these requirements into perspective. Consider that a new use case arises which did not exist when the vehicle was built. A handheld navigation device becomes available inside the vehicle - we want to attach the device to the vehicle, and integrate into the vehicle the specific human machine interface of the device. Its functions will become integrated into the vehicle's infotainment system e.g. via wireless Bluetooth communication. Navigation directions should be given out via the sound system of the vehicle while the current entertainment source is muted. The integration and removal of the handheld navigation device must be seamless to the driver and passengers. The DySCAS use-cases are structured into generic classes and specific cases and are discussed in detail in [ARJ+07].

A further motivator for the transition from the traditional static configuration to dynamic configuration is that this provides new opportunities for quality control. The increased use of embedded systems in vehicles implies a growth of complexity both in the products as well as in the development. In many advanced applications (e.g., relating to advanced human-machine interface and active safety), the data, functionalities and behaviours of traditionally independent software and hardware components are being integrated, further characterized by constraints on timeliness, performance, dependability, resource utilization, or technology compatibility. In system development, such product complexity is augmented by the involvement of multiple stakeholders, heterogeneous disciplines and technologies, as well as process and lifecycle concerns. Clearly, such challenges call for new technologies, tools, and methodologies, as well as new technologies for run-time monitoring, analysis, enforcement of qualities and error handling.

Predictable configuration management is a key problem today, even for statically configurable systems. New approaches are required that can handle problems with complex configurations to ensure functional and temporal correctness. The problems will be increasing with ever more software based functions and with the need to handle additional dynamics.

The DySCAS approach to dynamic architecture has considered the needs of advanced configuration management support in the following dimensions: **Product life-cycle management**: The types of product adaptations that could take place here include software upgrades, module replacements (hardware/software) and additions of new functions by adding new software or entire devices. **Optimization of performance and resource utilization**: Many embedded systems, including a large portion of automotive systems, are exposed to varying number of events and applications running at the same time, and with varying resource requirements over time by the applications. Moreover, the applications are in most cases of soft real-time nature, implying

that the timing constraints are subjective (determined by human senses) and not mission critical. All these aspects imply that static, worst-case designs cannot provide optimal performance and resource utilization, and call for a more adaptive configuration according to actual environmental conditions and system internal status. **Supervisory control for runtime system verification, validation, and error-handling**: The intended support of dynamic configuration offers a new opportunity for handling the complexity of embedded systems. It promises for the future a trade-off between the costly development-time and testing effort against advanced run-time support through monitoring, analysis, and enforcement of qualities.

4 Context Awareness in DySCAS

Within a DySCAS system, context-awareness is one of the most fundamental necessities to drive configuration and adaptation that is appropriate for current conditions. A number of factors affect the characteristics of 'context' in such a system. A wide variety in the nature of context information arises because there are many different sources of the information, different levels of precision, up-to-dateness, and data types. Examples of context information in an automotive environment include: user preferences for various service configurations; levels of resource available from instrumentation on processing platforms; system and environment conditions provided by sensors (raw and aggregated values); details of connected devices; and details of operational state, faults detected etc. Over time, in a dynamic upgradable system, the types of context available may change because for example a new sensor might be added, or instrumentation may be upgraded to provide better precision. Additionally the context requirements of a particular self-managing component may change over time. For example policy-logic may be upgraded to a more-sophisticated version which needs a wider selection of context, or increased precision, to make more-optimal configuration decisions.

These factors imply certain requirements on the context management approach used:

- Accurate - context must be sufficiently precise and fresh to support the configuration decisions based on it. These parameters are specific to each type of context information.
- Low latency - context information must be made available to the decision points where is it used without increasing the latency of the configuration decisions.
- Scalable - the context management should scale-up as necessary when new context providers and consumers are added.
- Distributed - to support scalability and low-latency the context management should be distributed with a hierarchical mapping between local and global-level context management.
- Dynamic - the mapping between context providers and consumers must be dynamic to enable incremental upgrades of strategic decision logic within components which in turn can lead to different context information requirements of these components.

The DySCAS component model for run-time configuration is based on embedded 'Decision Points' (DPs) in software components, into which policies are loaded at run time. Thus the configuration logic can be distributed throughout the middleware and application components wherever deferred logic or run-time context-sensitive configuration is required.

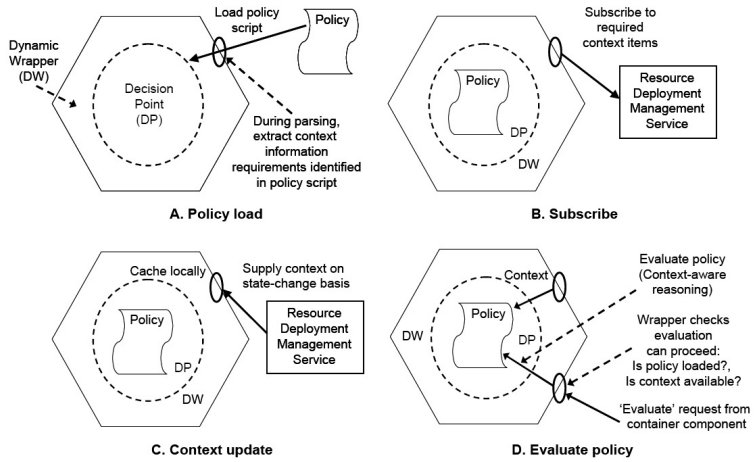


Figure 1: The dynamic context management approach and the context-related interactions between dynamic wrappers, decision points and policies.

The logic modules that are loaded into DPs take the form of policies written using a DySCAS-specific variant of the AGILE [Ant06] policy grammar. This grammar has been specifically developed to enable flexible and powerful expression of self-adaptive, context-aware behaviour whilst keeping the language complexity low.

The grammar explicitly separates (by type) externally provided context variables from internal working variables. This enables automatic determination of the context information requirements of a particular policy during run-time loading and parsing of the policy script.

Decision Points encapsulate a run-time supervisor (the 'Dynamic Wrapper' - DW) which provides a sand-box environment in which a policy operates. In addition to detecting and handling any problems arising from policy evaluation, the DW automatically identifies the context-requirements of a specific policy during the policy-load procedure (Figure 1a). To meet the context provision requirements as discussed above, a publish-subscribe context management service has been incorporated into the DySCAS middleware [DySa] (as part of the Resource Deployment Management Service component). To facilitate that the mapping between component and context information is dynamic, the DW automatically makes subscription requests to the context management system for each required context item (Figure 1b). The context management approach decouples, and avoids redundant communication between, dependant components. This can be achieved by performing context updates to consumers on a state-change, 'push' basis (Figure 1c) - note that this aspect is an implementation decision and is not mandated by the DySCAS specification. The context items are cached locally within the DW (i.e. process-local to the policy evaluation logic) to ensure low-latency access during policy evaluation (Figure 1d).

An automotive embedded system typically comprises applications and application platforms. Consisting of logical functions and software programs implementing such functions, the applications allow a vehicle system to have advanced functionality in the areas of information handling and human machine interactions, active and passive safety, control of chassis and power-train, etc. The application platform provides the implementation and runtime execution support for

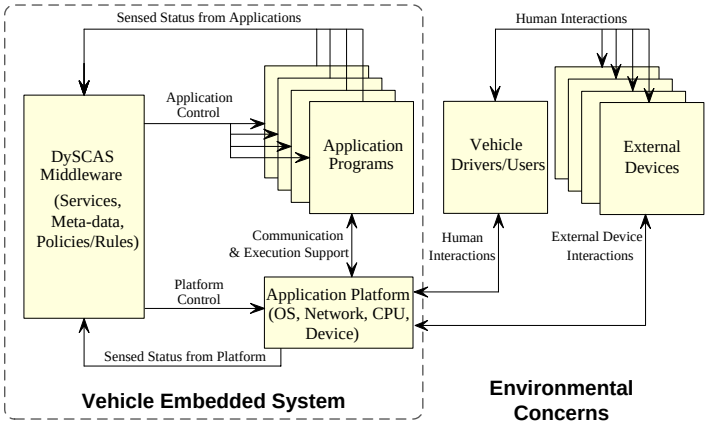


Figure 2: The control scheme of DySCAS, showing the top-level function blocks and their interactions (arrows).

the applications, typically consisting of basic software such as operating systems and device drivers, and hardware resources such as CPU, memory, and infrastructure like CAN, LIN and MOST. The actual content and features of the application platforms may differ among different application areas and vehicle systems.

An embedded system is *self-configurable* if it is able to autonomously adapt to changing environmental conditions or internal status by altering its structures, behaviours, and data to meet its functionality and quality requirements. From a systems perspective, such a feature relies on the following key characteristics:

A) The system's ability to monitor and define its internal status and external conditions (e.g., application modes, CPU and memory utilization, and attachment of external devices);

B) Its built-in knowledge about the configuration variability and related policies/rules for deciding and planning changes; and

C) Its ability to conduct dynamic configuration changes without violating the constraints relating to the overall system functionality, performance, and dependability.

The DySCAS approach to self-configurable automotive embedded systems exploits the fundamental principles of automatic control for the design of middleware interfaces and control functions, while adopting well-known architectural styles and reference models for the structuring of middleware architecture [DySa]. A schematic overview of the targeted self-configurable automotive embedded system is illustrated in Figure 2.

As a fundamental feature in achieving the self-configurability, inroads towards self-awareness in DySCAS means that a vehicle embedded system or component is allowed to be reflective, not only about the execution status and conditions of its environmental entities and resources (referred to as *execution context awareness*) but also about the related architectural design constraints (referred to as *architecture context awareness*). The term architecture here refers to the overall design of an embedded system that specifies the system's environment, application functions and programs, as well as the application platforms. Particular concerns include the design

and configuration of data, interfaces, functions and behaviours, relationships (e.g., composition and communication, and logical and implementation dependencies), and related conformance and quality constraints (e.g., restrictions of hardware binding and robustness).

These two aspects of self-awareness correspond to the above mentioned key characteristics A and B respectively. In DySCAS, they constitute the basis for applying the feedback and feedforward control principles to achieve self-configuration through middleware as in the following:

- *Awareness of execution context*. This type of context-awareness implies measuring the performance of applications and platform resources during run-time for the target system or component under control. The measurements are shown in Figure 2 with the connections from the application programs and application platforms to the DySCAS middleware. This includes for example, current application modes, QoS levels, utilization of memory and network bandwidth, length of task dispatch queues and ratio of missed deadlines. The instrumentation is necessary for being able to detect any deviations from intended system states and then react to the measured conditions as in feedback-control either based on predefined configuration policies/rules or based on the compensation derived according to the built-in architecture knowledge.

- *Awareness of architecture context*. This implies the provision of built-in knowledge about the system configuration and variability. As shown in Figure 2, such knowledge is embedded through the middleware in terms of meta-data for system or components that are subjected to the dynamic configuration. For software components, the meta-data for example specify their logical and resource-specific interdependencies, QoS contracts, platform compatibility, as well as their composability in regards to the overall system functionality, end-to-end performance and dependability. Given such knowledge, it is possible to deliberatively compute how a system should be optimized through restructuring and/or behaviour adaptation to compensate for measured deviations as in feedback-control. A further possibility with such knowledge is to enforce predictability in the presence of uncertainty during online optimization as in feedforward control.

One fundamental concept is variability, which refers to the specification of possible variations that a particular system configuration can have. Traditionally, variability is used for the feature configuration of product lines and dealt with off-line [CCG+07]. In DySCAS, the concept is extended to many more configuration items, levels of abstractions, and product lifecycle concerns. It is assumed that all meta-information and configuration management policies/rules that provide the middleware-based online configuration management are derived from corresponding off-line function and architecture design, verification and validation activities. The issues of particular concern include impacts of changes on the overall system functionality, end-to-end performance, and dependability.

DySCAS provides an information model that stipulates a set of predefined data types for formalizing various architectural and executional concerns in dynamic configuration management. See Figure 3 for an overview of the content and [DySa]for the specification. For a target system, the distinct parts that can be configured separately are referred to as *configuration items*, which can related to the logical design or the design of software and hardware implementations.

The *System Configuration Item Package* provides support for capturing various architectural

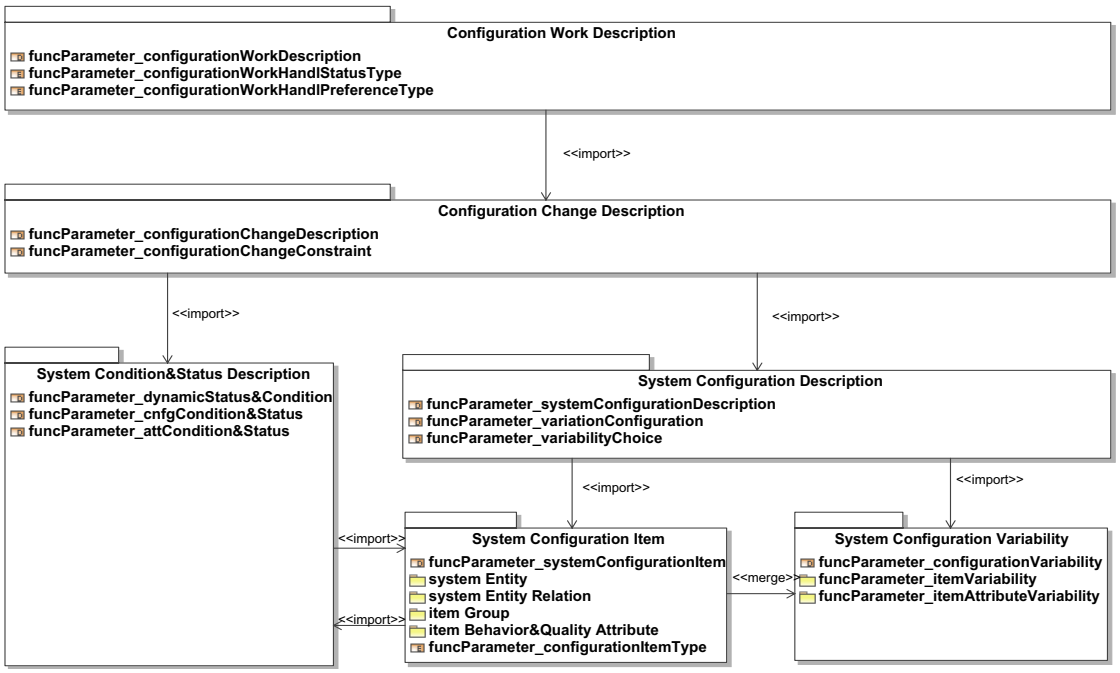


Figure 3: The top-level structure of DySCAS Information Model.

entities and relations, as well as the constraints in terms of behaviour and quality attributes. For capturing the allowed variability of system entities and dependencies, the definition of configuration item inherits the data types for variability description. This package also imports the data types for system condition and status for detailed specification of system behaviours. Figure 4 provides a top-level view of the data types of configuration item for capturing the meta-information in regards to structural, behavioural, and quality characteristics of a system architecture, where the attribute description type provides support for the behaviour and quality characteristics, such as constraints on precedence, timing and performance, and hardware resource utilization.

5 Implementation

DySCAS started by defining some advanced concepts for self-management and context awareness in dynamic, distributed systems with very challenging application and domain constraints and has succeeded in concretising these concepts in the form of reference implementations and demonstrators that have been built, collectively covering most of the DySCAS functionality. Two examples are briefly presented here: DyLite (short for DySCAS Lite/QoS) and GADGET are two of several partial *reference modules* [DySb, DySd].

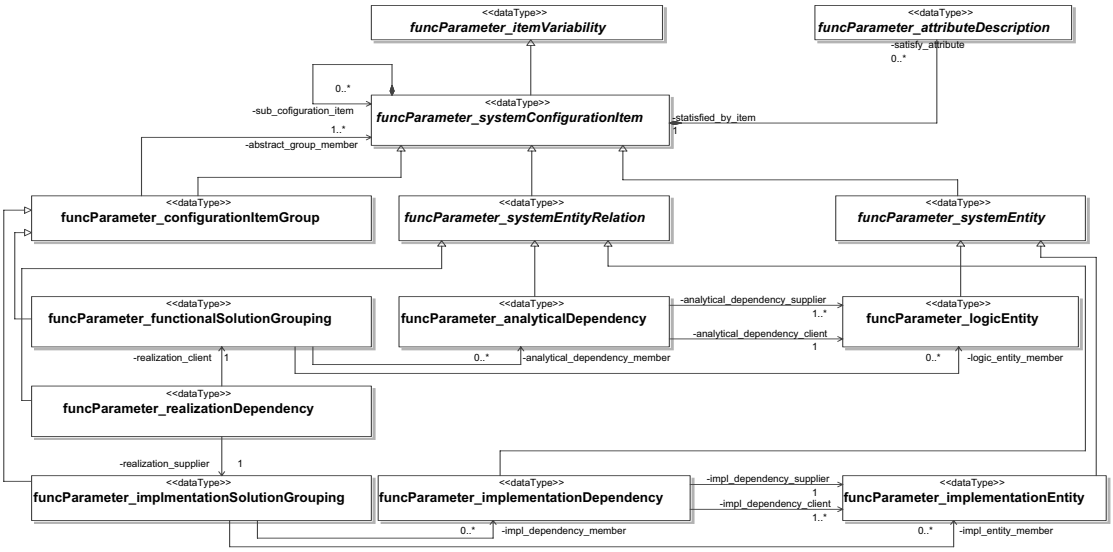


Figure 4: A top-level view of the DySCAS system configuration item definition.

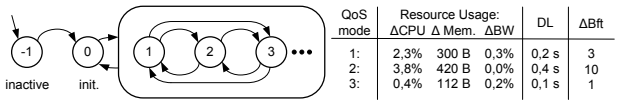


Figure 5: The DyLite task model, an example. State -1 is an inactive (nondeployed) state and state 0 is an initialization state. States 1–3 (and so on) are different QoS modes, annotated with the application's metadata: the resource usage of the processor (ΔCPU), memory (ΔMem) and bandwidth (ΔBW), deadlines/periods (DL) and benefits (ΔBft).

5.1 DyLite reference implementation

DyLite [PGF+09] focuses mainly on reconfiguration and quality of service (QoS) aspects and as such is not a complete implementation of DySCAS. The main intent was not to provide an optimized implementation, but a proof-of-concept to show that a system with aspects of self-awareness doesn't need to be very complex. To allow very compact implementation, DPs were only implemented using traditional code. Middleware services not necessary for the studied features were simplified. Due to technical constraints of the chosen platform, applications were statically deployed. Finally; only a fixed network topology was considered.

The reconfiguration abilities of DyLite have been used to demonstrate scenarios such as hard QoS guarantees, QoS mode optimization, load balancing between nodes and resilience to hardware failure (when a node fails, the applications that were running on it are reallocated to other nodes). All of these are achieved simply by triggering reconfiguration at system changes, such as software or hardware being added or removed from the system.

For the reconfiguration and quality of service decisions, knowledge of the *QoS behaviour* of application tasks (component), captured in delivery notes, is needed. In DyLite, these specify applications' maximum resource needs in each of their QoS modes. A higher numbered QoS

Target	Sensed Status	Related decision activities/algorithms	Related actuation activities
Applications	A QoS mode request from the application	Compare with the allowed maximum QoS mode	Switch application to the highest mode out of the requested and allowed maximum one.
Application Platform: network	Token count in bucket	Leaky bucket algorithm: Remove tokens when sending, periodically replenish buckets.	If tokens are not available then trying to send: block calling application task until replenishment.
	Changes in application set or node connectivity	Find a new configuration using the self-configuration algorithm. It is ensured to meet the Liu-Layland criterion for the communication.	Distribution of new maximum QoS modes for applications. The settings of leaky buckets are updated at QoS mode change.
Application Platform: CPU and memory	Detected Deadline miss	React to offending tasks	E.g. by changing their modes, skipping jobs, killing it, etc.
	Changes in application set or node connectivity	Find a new configuration using the self-configuration algorithm. It is ensured to meet the Liu-Layland criterion for all CPUs and not to use more than available memory.	Distribution of new maximum QoS modes for applications. The deadline monitoring is updated at application QoS mode change.

Table 1: Sensing and actuation in DyLite. Compare with Figure 2.

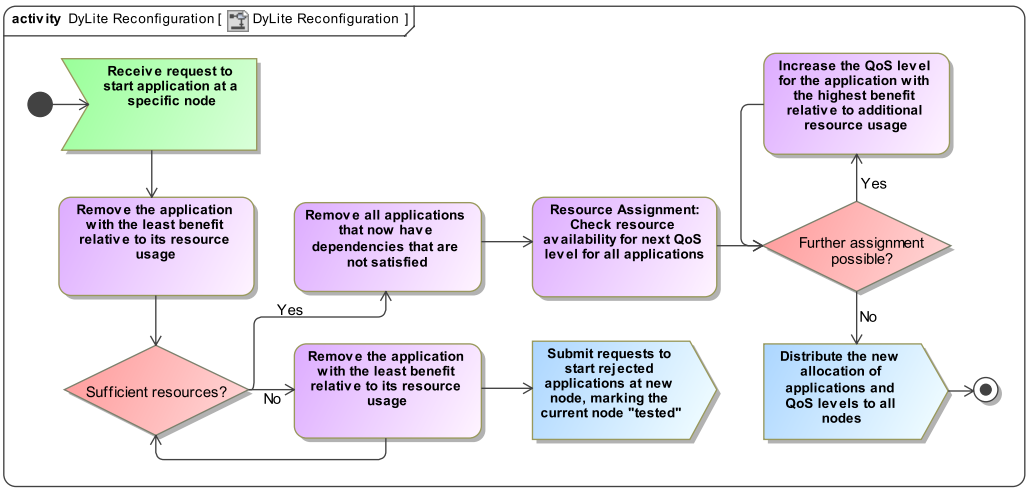


Figure 6: Overview of the reconfiguration algorithm used in DyLite.

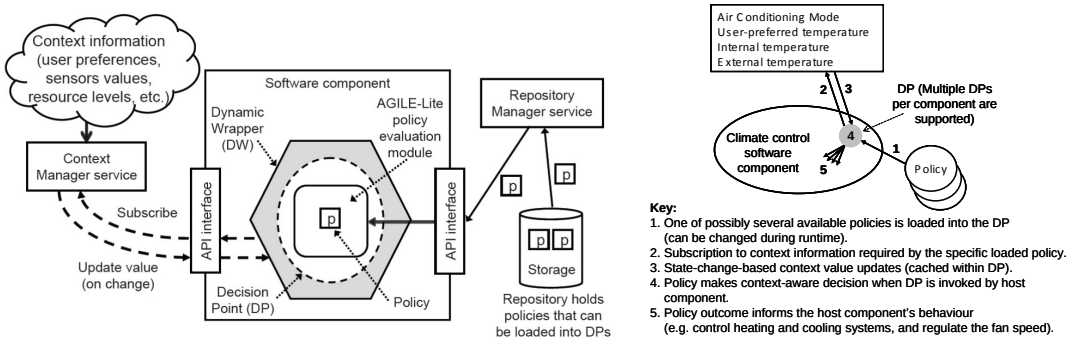
mode is further always assumed to provide better QoS to the user and use more resources.

The delivery notes are stored inside the middleware, as specified in the DySCAS information model and reference architecture [DySa]. Together with each QoS mode, a measure of merit/benefit of the application's functionality in that mode is given. An illustration of an example application with 3 QoS modes, and its delivery note, is given in Figure 5.

Applications are assumed only to have dependencies on each others' existence – one application can't require another application to run in a specific mode, and hence QoS is assumed not to propagate along communicating applications. Each application periodically performs any applicable QoS change itself as instructed by the middleware. To improve performance, this is done by having an intermediate variable containing the maximum allowable QoS mode, which is updated when a new configuration has been found by the reconfiguration algorithm.

With such built-in knowledge of application QoS behaviour, the middleware monitors current application and platform status and thereby derives appropriate *adaptation activities*, e.g. changing an application's QoS mode. A misbehaving application, e.g. one which uses more resources than specified, is acted upon by the middeware – the control strategy is given in Table 1.

For CPU, deadline overrun is detected and handled. For network usage, the leaky bucket algorithm is used to shape the traffic and hence control the traffic flow. To provide schedulability



(a) Implemented components to demonstrate and validate the policy-based context-aware dynamic configuration aspects, showing the interactions with the Repository Manager during policy load, and the interactions with the Context Manager.

(b) Outline of the deployment of the climate control application onto the GADGET platform.

Figure 7: Policies in GADGET.

guarantees, a special self-configuration algorithm has been developed [FCT08] illustrated in Figure 6. Upon configuration changes, this algorithm is triggered to find a new configuration which is then distributed. Assuming correctly set scheduling parameters such as priorities, a configuration (task allocation and maximum QoS modes) with schedulability guarantees for all deployed applications can be found.

5.2 Policy-based Dynamic Adaptation in DySCAS

A reference module (internally named GADGET) was developed to demonstrate and validate the policy-related concepts of DySCAS. Selected parts of the DySCAS specification were implemented to enable the policy-based dynamic self-management to operate in real software components running on automotive-typical hardware platforms (see Figure 7a). Core themes targeted here were run-time configuration and context-aware adaptation, as illustrated in overview in Figure 1. The main mechanisms of interest are the DP which is embedded into components at design time, and is a place holder for a policy subsequently loaded during run time, and the DW which provides run-time supervision of DPs (this guarantees that DP evaluation always returns a 'valid' output despite possible internal errors which are detected and handled silently by either the policy evaluation engine itself, or the DW). Where the policy cannot be evaluated (for example no policy is loaded into the DP, or the required context information is not available) the DW ensures that a design-time decided 'default' value is returned, or in some cases a roll-back to a previously working policy may be performed, depending on circumstances. In effect, the DW ensures that the added dynamic behaviour aspects collapse down to predictable, static outcomes when problems occur, ensuring that the introduction of dynamic adaptive behaviour is not accompanied by complex new failure modes. These aspects are discussed in detail in [AWC+08].

Key components of the reference module include: **The AGILE-Lite library** [PA07] which is responsible for policy evaluation and dynamic decision making. This derivative of the earlier AGILE is optimised for processing speed and deployment on embedded platforms (e.g. it has a small memory footprint). **Communication Service** (CS) responsible for handling communication between services based on socket and message queues mechanisms. **Context Manager**

```
<ReturnValues>
   <ReturnValue Name="RetOK" Value="0"/>
</ReturnValues>
<Actions>
   <Action Name="ReturnAction">
      <Return ReturnValue="RetOK" />
   </Action>
</Actions>
<Policies>
   <Policy Name="ClimatePolicy">
      <Execute Action="ReturnAction" />
   </Policy>
</Policies>
```

The policy above is a simple dummy placeholder for more advanced logic.

The policy to the right uses system context information (the current cabin temperature and the user's preference temperature) to determine the climate control system's settings.

Policies like these can be interchanged at run-time; the automatic context provision system ensures policy-specific required context information is provided to the DP.

```
<EnvironmentVariables>
   <EVariable Name="InteriorTemperatureCelcius" Type="long" />
</EnvironmentVariables>
<InternalVariables>
   <IVariable Name="IdealTemperatureCelcius" Type="long" />
</InternalVariables>
<Templates>
   <Template Name="ClimateTemplate">
      <Assign Variable="IdealTemperatureCelcius" Value="19" />
   </Template>
</Templates>
<ReturnValues>
   <ReturnValue Name="RetIdeal" Value="0" />
   <ReturnValue Name="RetHigher" Value="1" />
   <ReturnValue Name="RetLower" Value="-1" />
</ReturnValues>
<Actions>
   <Action Name="BeginCheck">
   <EvaluateTRC TRC="TempCheck" />
</Action>
</Actions>
<ToleranceRangeChecks>
   <TRC Name="TempCheck" Check="InteriorTemperatureCelcius"
      Compare="IdealTemperatureCelcius" Tolerance="15" ActionInZone
      ="RetIdeal" ActionLower="RetLower" ActionHigher="RetHigher" />
</ToleranceRangeChecks>
<Policies>
   <Policy Name="ClimatePolicy">
      <Load Template="ClimateTemplate" />
      <Execute Action="BeginCheck" />
   </Policy>
</Policies>
```

Figure 8: Diverse climate control application policies, illustrating the flexibility of the approach

(CM) responsible for context information handling, including operations of context provision, subscription. **Repository Manager** (RM) responsible for handling policy repository, including policy update, policy versioning, and policy rollback. **Component Development Kit** (CDK), which is a framework substantially simplifying development of components containing dynamically configurable Decision Points (DP). Several dynamically configuring applications, with accompanying policy logic have been developed to run on GADGET. One of these was a cabin climate control application (outlined in Figure 7b) in which a number of different policies could be loaded into a DP - differently configuring its behaviour - each is accompanied by its unique set of context information requirements, which relates to the sophistication of the policy itself (as illustrated in Figure 8). This ability to dynamically change both the self-management logic and the selection of context information it uses enables the deployment of a generic software component across a range of vehicles (e.g. to account for different equipment levels, sensors / actuators fitted); the component is subsequently customised either at the level of individual models, or at the level of individual vehicles (e.g. to account for different user preferences).

6 Conclusion

The DySCAS project has pushed forward the boundaries in several domains. In addition to producing a detailed specification for autonomic self-management in the very demanding automotive arena, it has also explored a number of new aspects of autonomics and control systems concepts and has concretised some earlier abstract aspects. It has led to a new level of functionality in policy-based computing, pushed forward middleware design concepts and demonstrated how dynamic, scalable and low-latency distributed context provision and context awareness can be achieved.

The feasibility of the DySCAS approach has been demonstrated in several key ways. The DyLite implementation of DySCAS, together with the demonstration activities within the project [DySc], have shown that implementing a middleware supporting self-configurability is indeed

feasible, and the run-time overheads are acceptably small. We have shown that relatively simple models of available resources and deployed applications are sufficient to make the system able to monitor its own behaviour (key characteristic A in section 4), use its in-built knowledge about both execution context, e.g. current QoS mode for applications, and architecture context, e.g. the possible configuration variability, to infer a possible configuration (key characteristic B), and conduct configurational changes, e.g. setting an upper limit on the QoS mode that an application may run at (key characteristic C).

DySCAS represents a first step towards self-managing automotive systems. It is difficult to predict how long it will take for advanced concepts to reach production vehicles. However, we have set a foundation, we have investigated the challenges, identified the tradeoffs, the costs and risks. We have identified realistic use cases and demonstrated how these can be realised.

Acknowledgements

DySCAS was funded by the European Commission's 6th framework program "Information Society Technologies". Project number: FP6-IST-2006-034904. Further details available at [DySe].

Bibliography

[ALE+06] R. J. Anthony, A. Leonhardi, C. Ekelin, D. Chen, M. Törngren, G. de Boer, I. Jahnich, S. Burton, O. Redell, A. Weber, V. Vollmer. A Future Dynamically Reconfigurable Automotive Software System. In *Proc. 26th Automotive Electronics Conf.: Moderne Elektronik im Kraftfahrzeug, Innovationen, Neuentwicklungen, Anwendungen*. Dresden, Germany, June 27–28 2006.

[Ant06] R. J. Anthony. A Policy-Definition Language and Prototype Implementation Library for Policy-based Autonomic Systems. In *Proceedings of Third International Conference on Autonomic Computing (ICAC)*. Pp. 265–276. Dublin, Ireland, June 2006.

[ARJ+07] R. J. Anthony, A. Rettberg, I. Jahnich, M. Törngren, D. Chen, C. Ekelin. Towards a Dynamically Reconfigurable Automotive Control System Architecture. In Rettberg et al. (eds.), *Embedded System Design: Topics, Techniques and Trends*. Pp. 71–84. Springer-Verlag, May 29 – June 1 2007.

[AWC+08] R. J. Anthony, P. Ward, D. Chen, J. Hawthorne, M. Pelc, A. Rettberg, M. Törngren. A Middleware Approach to Dynamically Configurable Automotive Embedded Systems. In *Proceedings of The First Annual International Symposium on Vehicular Computing Systems*. Dublin, Ireland, July 22 – 24 2008.

[BCSV08] T. Bure, J. Carlson, S. Sentilles, A. Vulgarakis. A Component Model Family for Vehicular Embedded Systems. In *The Third Intl. Conf. Software Engineering Advances, Washington, DC*. Pp. 437–444. Oct 2008.

[But06] G. Buttazzo. Research trends in real-time computing for embedded systems. *SIGBED Rev. 3, 3* 1-10, Jul 2006.

[CCG+07] P. Cuenot, D. Chen, S. Grard, H. Lnn, M.-O. Reiser, D. Servat, R. Tavakoli Kolagari, M. Trngren, M. Weber. Towards Improving Dependability of Automotive Systems by Using the EAST-ADL Architecture Description Language. In Lemos et al. (eds.), *Architecting Dependable Systems IV*. LNCS 4615, pp. 39–65. Springer Verlag, 2007.

[CONa] CONNECT Consortium. CONNECT project website. http://www.ist-connect.eu/.

[CONb] CONSEQUENCE Consortium. CONSEQUENCE project website http://www.consequence-project.eu/.

[CY08] N. S. Chang, J., M. Yoon. Intelligent Context-Aware System Architecture in Pervasive Computing Environment, Washington, DC. In *The 2008 IEEE Intl. Symp. Parallel and Distributed Processing with Applications*. Pp. 745–750. Dec 2008.

[DySa] DySCAS Consortium. D2.3 DySCAS System Specification, Part I, 2009. project deliverable. http://www.dyscas.org/doc/DySCAS_D2.3_part_I.pdf

[DySb] DySCAS Consortium. D3.1 Specification of Reference Implementation and Validation Applications, 2009. project deliverable. http://www.dyscas.org/doc/DySCAS_D3.1.pdf

[DySc] DySCAS Consortium. D3.3 DySCAS Demonstrator Application and Specification, 2000. project deliverable. http://www.dyscas.org/doc/DySCAS_D3.3.pdf

[DySd] DySCAS Consortium. D4.3 Evaluation Report, 2009. project deliverable. http://www.dyscas.org/doc/DySCAS_D4.3.pdf

[DySe] DySCAS Consortium. DySCAS Project Website: http://www.dyscas.org.

[FCT08] L. Feng, D. Chen, M. Törngren. Self Configuration of Dependent Tasks for Dynamically Reconfigurable Automotive Embedded Systems. In *Proc. 47th IEEE Conf. Decision and Control (CDC)*. Pp. 3737–3742. Cancún, Mexico, Dec 2008.

[FRLK08] S. Fuchs, S. Rass, B. Lamprecht, K. Kyamakya. A model for ontology-based scene description for context-aware driver assistance systems. In *1st Intl. Conf. Ambient Media and Systems, Quebec, Canada*. Pp. 1–8. FEB 2008.

[FSSC08] S. Fritsch, A. Senart, D. C. Schmidt, S. Clarke. Time-bounded adaptation for automotive system software. In *30th Intl. Conf. Soft. Eng., Leipzig, Germany*. Pp. 571–580. May 2008.

[GPZ05] T. Gu, H. K. Pung, D. Q. Zhang. A service-oriented middleware for building context-aware services. *Netw. Comp. Appl.* 28(1):1–18, Jan 2005.

[inC] inContext Consortium. inContext project website. http://berlin.vitalab.tuwien.ac.at/projects/incontext.

[Joh07] S. Johnson. A framework for mobile context-aware applications. *BT Technology Journal, Springer Netherlands* 25(2):106–111, April 2007.

[KSC08] J. Kim, J. Sim, J. Choi. Resource-Oriented Design Framework for Embedded System Components. *Electronic. Notes Theor. Comput. Sci.* 215:171–189, Jun 2008.

[LSK+08] S. Lee, I. Shin, W. Kim, I. Lee, S. L. Min. A design framework for real-time embedded systems with code size and energy constraints. *ACM Trans. Embed. Comp. Sys.* 7(2):1–27, Feb 2008.

[MID] MIDAS Consortium. MIDAS project website http://www.appearnetworks.com/-EU-Research-Project-MIDAS-.html.

[MUS] MUSIC Consortium. MUSIC project website. http://www.ist-music.eu/MUSIC/about-music.

[PA07] M. Pelc, R. J. Anthony. Towards Policy-Based Self-Configuration of Embedded Systems. *Systems and Information Science Notes* 2(1):20–26, 2007. The Systemics and Informatics World Network (SIWN).

[PGF+09] M. Persson, J. García, L. Feng, D. Chen, T. Naseer Qureshi, M. Törngren. DyLite: Design, Implementation and Experiences. Technical report TRITA MMK 2009:06, ISSN 1400-1179, ISRN/KTH/MMK/R-09/06-SE, Mechatronics Lab, Department of Machine Design, Royal Institute of Technology (KTH), Stockholm, Sweden, 2009.

[SWV07] L. O. Bonino da Silva Santos, R. P. van Wijnen, P. Vink. A service-oriented middleware for context-aware applications. In *5th Intl. Workshop on Middleware For Pervasive and Ad-Hoc Computing: At the ACM/IFIP/USENIX 8th intl Middleware Conference, California, USA*. Pp. 37–42. Nov 2007.

[TN06] C. Tadj, G. Ngantchaha. Context handling in a pervasive computing system framework. In *3rd intl. Conf. Mobile Technology, Apps. & Sys., Bangkok, Thailand*. Oct 2006.