Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

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, Germany
[3]Royal Institute of Technology, Sweden
[4]Bosch GmbH, Germany
[5]Volvo Technology

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 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. The dynamic configuration and control-theoretic technologies used are described in outline and the way in which the demands of highly flexible dynamic configuration and highly robust operation are simultaneously met without compromise, is explained. A number of generic use-cases have been identified, each with several specific use-case scenarios. One generic use-case is described to provide an insight into the extent of the flexible reconfiguration facilitated by the architecture.

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 use 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 exciting 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 DySCAS. 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-managed 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].

For DySCAS, a policy-based approach is being developed to enable very flexible run time configuration. This will apply both at the level of the DySCAS middleware and at the level of applications. The actual technology under development is briefly discussed in section 5.2.

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 is 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 section 6). A sample of these services are:

- 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),
- 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),
- 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.

5.1 Runtime Configuration

To support the dynamic configuration requirements of the project, a policy-configurable middleware architecture has been designed. This section describes those aspects of the middleware that support the policy-based determination of behaviour.

The policy approach allows decisions about specific behaviours to be deferred beyond the point of deployment of the embedded software and facilitates highly flexible run-time configuration. The 'policy' is a set of rules written using the AGILE grammar, available at [17]. This is then stored and transmitted as a small data file and is typically only of the order of a few hundreds of bytes in size. Policies can thus be loaded dynamically from an on-board vehicle repository or even transmitted to the vehicle via a wireless hotspot (for example, a pushed update originating at the vehicle manufacturer's back-end systems). Because policy files are so small, they can be transmitted with low latency even over the slowest on-vehicle bus networks.

The policy engine itself is based on the AGILE policy technology [18], but because the target ECUs are resource-constrained devices, a special lightweight version has been

developed. AGILE-Lite, which is described in more detail in [19].

Each software component within the DySCAS middleware, as well as those that are part of application software, can be run-time configured by embedding one or more policies. To facilitate this embedding in the simplest way for the software developer, special place-holders, called Decision Points (DP) have been devised.

The concept of a DP is that the software developer can identify places at which the actual behaviour needs to be deferred beyond code deployment (to allow flexible upgrades) or needs to be dynamically decided at run-time (to achieve context aware behaviour). Once these places are identified, the software developer inserts an API call to create the DP(s). At run time, policies can be loaded into the DPs to control the logical flow of the component. This concept is illustrated in figure 1.

In addition to dynamically deciding which policy to load into a DP, it is also necessary to be able to automatically decide which context information will be used by the policy. Otherwise the extent to which a particular policy can be changed over time is restricted by the original (static) set of context data types decided by the developer. This problem is resolved by a Context Manager service (CM) to which each DP subscribes (during the load process), with a list of required context items.

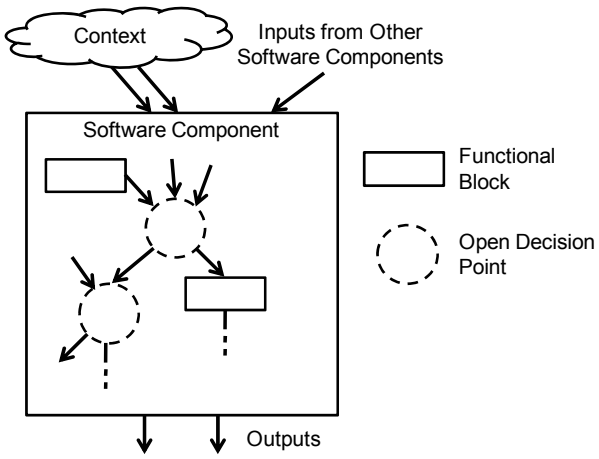

Fig. 1. A software component with two decision points (DP) inserted by the developer. The decision logic, in the form of a policy, is loaded into each DP at run-time.

DySCAS is an automotive middleware platform, and as such must be highly robust. However, the introduction of a dynamic reconfiguration mechanism potentially introduces a number of problems that can occur during the run-time loading and evaluation of a policy. These include: no suitable policy found in the repository, policy load failed, parse error (policy file corrupted), or required context information not available. These problems are all resolved by arranging that, should any problem occur during the evaluation of the dynamic logic, the affected DP silently falls back to statically-decided behaviour. During the specification of a DP, the developer is required to provide a default output decision that should be returned if any problem is detected. Mechanistically the detection and handling of any errors is performed by a silent-operating 'wrapper' that is automatically created around each DP (at compile time). The wrapper monitors the behaviour of the DP (events such as policy load, evaluate policy) and also is responsible for interfacing with the CM, and thus can detect whether the CM is able to provide the full set of required context inputs.

The policy execution framework; comprising DP, wrapper and CM mechanisms is described in more detail in [20].

6. USE-CASES

The technologies under development within DySCAS are based on requirements derived from a rigorous analysis of future use-cases. A layered model describes a hierarchical mapping of use-cases onto system requirements, and ultimately, onto services and technology within the vehicle. Generic use-cases (GUC) represent groupings of specific use-cases (SUC) requiring similar subsets of functionalities. These are defined as atomic 'system functionalities' which can not be further broken down.

To be able to efficiently develop the DySCAS middleware a consistent minimised but complete set of requirements is needed which decouples the middleware design from the concrete use cases. This is because several derived 'system functionalities' may overlap – increasing the development costs if directly targeted by the services layer. The 'system requirements' layer contains such a set of requirements. Normally, a 'system requirement' reflects the information of one or several 'system functionalities', but some 'system requirements' are directly derived from use cases (i.e. the mapping from functionalities to requirements can be many:one or one:one).

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

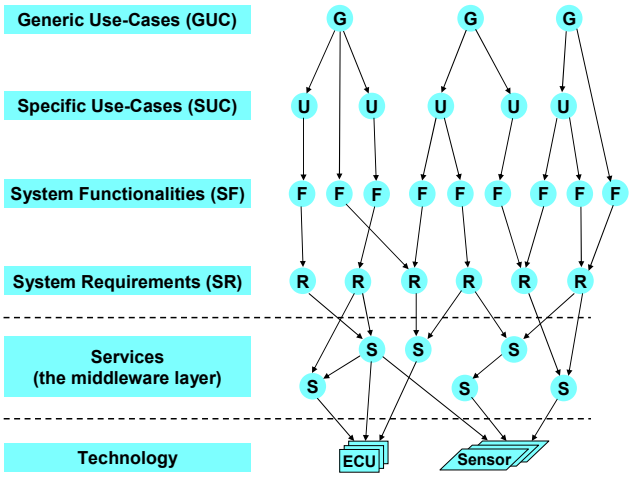

Fig. 2. Mapping Use-cases onto DySCAS Services and Technology.

Each of the use cases considered consists of a sequence of systems functionalities, and thus map down to systems requirements. 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 include:

- New device automatically attached to the car,
- Integrating new software functionality,
- Closed re-configuration.

6.1 Closed re-configuration

This GUC is described to provide an insight into the type and extent of dynamic configuration targeted. The GUC 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 specific use-case is 'Migration of services in the case of HW failure'. The composition of services needed to achieve the reconfiguration is illustrated in figure 3.

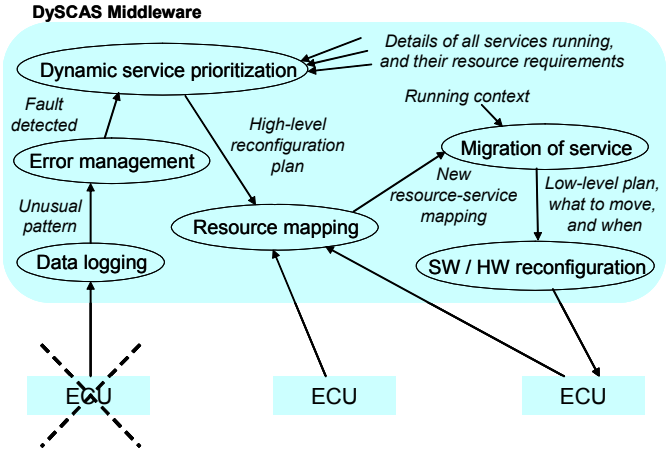

Fig. 3. Composing DySCAS services to satisfy the 'Migration of services in the case of HW failure' use-case.

7. CONCLUSION

The project targets an ambitious collection of use-cases, requiring dynamic adaptation and thus the new middleware must provide a number of new services in a flexible and efficient manner.

A variety of specific services have been identified based on the requirements of the various use-cases. The use-cases have been identified at two levels, generic and specific. Atomic functionalities were then identified, and by removing any duplication, a non-redundant set of requirements are derived for implementation. One of the generic use-cases has been discussed in outline.

In order to meet the dynamic configuration requirements imposed by the use-cases, a run-time configurable middleware has been designed. The configuration is provided by dynamically replaceable policies which can be deployed into any of the software components that comprise the middleware and the applications that will operate over the middleware.

The run-time configuration mechanisms have been devised such that they are extremely robust to any problems arising from the dynamic configuration process, automatically reverting to statically defined default behaviour in such cases.

8. NEXT STEPS

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 [21].

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, and Maguire. L, (2004). Modeling and optimizing run-time reconfiguration using evolutionary computation. Trans. on Embedded Computing Sys., 3(4):661–685.

[2] Götz. M, Rettberg. A and Pereira. C, (2005). Towards Run-time Partitioning of a Real Time Operating System for Reconfigurable Systems on Chip. Proc. Intl. Embedded Sys. Symp. - 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, pp18-25.

[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. S, and Kat. R,, Self-Stabilizing Distributed File Systems, Intl. Workshop on Self-Repairing and Self-Configurable Distributed Systems, 21st Symp. on Reliable Dist. Sys., 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 Centre 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, 25(6), Dec. 2005.

[14] R Sanz, K-E Årzén. Trends in software and control. IEEE Control Systems Magazine, 23(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] AGILE support website: http://www.PolicyAutonomics.net

[18] R. Anthony, Policy-centric integration and dynamic composition of autonomic computing techniques, Proc. 4th Intl. Conf. on Autonomic Computing (ICAC), Jacksonville, Florida, USA, June 2007, IEEE Computer Society.

[19] Pelc M and Anthony R, Towards policy-based self-configuration of embedded systems, System and Information Sciences Notes, 2 (1), pp. 20-26, 2007, The Systemics and Informatics World Network (SIWN).

[20] R. Anthony, M. Pelc, P. Ward, J. Hawthorne, K. Pulnah, A Run-Time Configurable Software Architecture for Self-Managing Systems, Fifth International Conference on Autonomic Computing (ICAC), Chicago, IL, USA, June 2-6, 2008, IEEE Computer Society.

[21] DySCAS website: http://www.dyscas.org