Methods for configuration and composition of adaptable system software

Extended Abstract

Michael Schulze
Otto-von-Guericke-Universitat Magdeburg
Faculty of Computer Science
Department of Distributed Systems (IVS)
Universitatsplatz 2
D-39106 Magdeburg, Germany
mschulze@cs.uni-magdeburg.de

Our known daily environment is subjected by important changes. The environment is more and more equipped with electronical devices becoming smarter. However, they are mostly not interlinked and therewith the next logical development step is from single smart artefacts to networked devices, whereas the sum enables more than the part. Therefore, new possibilities arise like smart homes, habitat monitoring, emergency warning systems to list a few only. However, the smart environment, consisting of sensors and actuators, has to be aware of the context defined by the surrounding physical world and the challenge is to enable predictable operation under a constantly changing environment and open and dynamic cooperation.
Related to the task of the individually devices, different hardware is used ranging from 8 Bit microcontroller to 32 Bit control systems. The heterogeneity concerns not only the CPU but a broad variety of communication media have to be provided also. Obviously, development of application-oriented, easy usable and adaptable systems software (e.g. Operating System Kernel, Middleware) is a challenge even though several systems providing at least an operating system. Other systems like some embedded microcontrollers which are not equipped with an operating system have to be provided with an at least very thin hardware abstraction layer how our PURE (Portable Universal Runtime Executive [1]) operating system family it is.
To overcome the differences in communication abstractions on operating systems (Linux/Windows on PC and PURE on microcontroller) our publish/subscribe based middleware COSMIC [3] is used. COSMIC provides a unique communication interface and hides the local peculiarities of the operating system and therewith the location of the application can be chosen freely. A migration of applications between platforms is enabled.
In general, building system software for such heterogeneous system networks as it is envisaged requires a flexible and adaptive approach to meet the largely different requirements resulting from the diversity of hardware platforms, performance requirements, energy constraints and memory size. Additionally, the resources (e.g CPU time, RAM/Flash usage) needed by the fundamental software should be minimal. Clearly, the amount of resources is tightly related to the functional and quality properties of such software. Thus, the only promising way to achieve minimal resource consumption is to restrict the functionality to just those components that are absolutely needed by the application. Some steps in this direction were already made and other are ongoing. My research interests has three major objectives:

- Firstly, identifing and structuring of the necessary functional and non-functional requirements,
- secondly developing a family of suitable programming abstractions and
- thirdly providing an adequate framework by which the system software can be tailored and configured according to the specific application needs.

Determining functional and non-functional requirements for a broad variety of applications isn't easy and often the requirements are intermixed or they are not explicit. As an example, real-time behaviour is a requirement and is apparently non-functional. However, real-time has also a functional part and can be realised in different kinds. On the other side, functional requirements of applications can be e.g. analog-digital-converter device, multi threading or communication facilities.
The abilities of our system software are structured in feature models [4, 2]. The models are used also to configure the system software by selecting some features. Thereby, application requirements are mapped to features, and elected features called feature set describing the tailored system software consisting of operating system and middleware.
We assume it is necessary to design the system software OS-Kernel as well as Middleware as a family [6] to achieve flexibility, adaptability and dependability. For example to deal with the limited memory (e.g. few kilobytes in sensor nodes) the software should provide only the functions actually needed by the application. In order to reach this goal, the functionality of the software has to be presented as a collection of configurable components and functions. Design decisions about the needed properties have to be deferred as long as possible and determined by application needs.
Using a family approach enables a fine-grained selection and adaptation of functional requirements. Selection refers to the common requirements that apply to a certain class of system components, e.g. the existence of process scheduling, synchronization and/or real-time communication. Adaptation relates to the specific needs within such a class, for

example non-blocking synchronization primitives or a real-time scheduling policy. Apart from the functional properties, the encapsulation of non-functional requirements, like dependability issues and real-time properties need a separate treatment. These requirements, termed crosscutting concerns, are often fundamental system policies and refer to issues as robustness, fault-tolerance and real-time. Since crosscutting quality features may apply to multiple classes it is impossible to implement them as independent encapsulated entities because this would restrict the above-mentioned freedom of selection and adaptation. Aspect-oriented programming (AOP) seems to be a suitable possibility to deal with crosscutting concerns [5]. AOP allows separating of the functional system components and non-functional system components called aspects. Aspects are woven into the system software during build time. Thus there is no extra runtime overhead to dynamically introduce these aspects. Clearly, there is the added effort to execute the respective mechanisms but that has to be executed too, if the crosscutting code is intermixed with the component code. There are many examples in the context of system software about the benefits of aspects and a component model to configure such software like synchronization and profiling.
Our testbed of the developed software is a quadratic robot platform, called Q, on one side and on the other side a sensor network which will be installed in the near future. On the corners of Q are mounted microcontrollers contolling two motors one for direction and one for driving and two infrared distance sensors for obstacle detection. The microcontrollers are networked via CAN (Controller Area Network) and a laptop that is also part of the CAN acting as gateway. The microcontrollers run our PURE operating system and the PC works under Linux (in near future under Xenomai a real-time Linux). The communication is publish/subsribe and is handled by COSMIC.
The major aim of our working group is to develop suitable programming abstractions which provide the needed requirements and allow code reuse on application level. Enabling the migration of applications from development system to deployment system should be reached as long term objective.

Keywords
Configuration and Composition, System Software, COSMIC Middleware, Family-based Design, Feature Models, AOP

1. REFERENCES

[1] D. Beuche, A. Guerrouat, H. Papajewski, W. Schröder-Preikschat, O. Spinczyk, and U. Spinczyk. On the Development of Object-Oriented Operating Systems for Deeply Embedded Systems—The PURE Project. In *Proceedings of the 2nd ECOOP Workshop on Object-Orientation and Operating Systems (ECOOP-OOOSWS'99)*, pages 27–31, Lisboa, Portugal, June 14–18 1999.

[2] D. Beuche, H. Papajewski, and W. Schröder-Preikschat. Variability Management with Feature Models. In *Proceedings of the Software Variability Management Workshop*, pages 72–83, University of Groningen, The Netherlands, Feb. 2003. Technical Report IWI 2003-7-01, Research Institute of Mathematics and Computing Science.

[3] A. Casimiro, J. Kaiser, and P. Verissimo. An architectural framework and a middleware for cooperating smart components. In *ACM Computing Frontiers conference (CF'04), ISCIA*, Italy, April 14–16 2004.

[4] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, November 1990.

[5] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In M. Aksit and S. Matsuoka, editors, *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP '97)*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer-Verlag, June 1997.

[6] D. L. Parnas. On the design and development of program families. *IEEE Transactions on Software Engineering*, SE-2(1):1–9, March 1976.