USING ASPECT-ORIENTED SOFTWARE DEVELOPMENT IN REAL-TIME EMBEDDED SYSTEMS SOFTWARE

A Review of Scheduling, Resource Allocation and Synchronization

Pericles Leng Cheng
Department of Computer Science, Cyprus College
6, Diogenes Street, Engomi, Nicosia 1516
pcheng@cycollege.ac.cy
Department of Computer Science, University of Cyprus
75 Kallipoleos Str., P.O.Box 20537, CY-1678 Nicosia
pcheng@cs.ucy.ac.cy

George Angelos Papadopoulos
Department of Computer Science, University of Cyprus, 75 Kallipoleos Str. PO Box 20537
george@cs.ucy.ac.cy

Keywords: Aspect-oriented programming, AOP, Aspect-oriented Software Development, AOSD, Embedded Systems, Real-time Systems.

Abstract: Timeliness and criticality of a process are the two main concerns when designing real-time systems. In addition to that embedded systems are bounded by limited resources. To achieve these concerns while at the same time using a minimal amount or resources, real-time embedded systems use different techniques such as task scheduling, resource management and task synchronization. These techniques involve a number of the modules of the system which makes the use of Aspect-Oriented Software Development imperative. AOSD is a programming technique which uses the notion of join points to capture specific locations in code execution and then use advices to insert new code. This paper examines existing work in the development of schedulers, resource allocation agents and synchronization techniques using Aspect-Oriented Software Development in real-time embedded systems. An analysis of the existing research is used to describe the advantages of using AOSD over conventional OOP methods and to identify areas where further research may be required.

1 INTRODUCTION

An embedded system is a combination of software and hardware in such a way that it performs a specific operation. In most cases an embedded system needs to be able to perform in real-time. Real-time systems are developed with two important areas in mind: Timeliness and criticality (Cooling, 2002). Timeliness is the ability of a process to complete within a specified timeframe whereas criticality has to do with the importance of a process to complete and what might occur if the process is not able to complete. In addition, real-time embedded systems require more efficiency due to the limited amount of resources available. When designing real-time embedded system software, the developer needs to take into account concepts such as concurrency, system resource usage and task structuring and implementations (Cooling, 2002). This is why embedded systems utilize real-time schedulers to control processes and their execution. Resource usage also plays an important role in real-time embedded applications and that is why it is important to have a good way in which to manage resources. Finally, tasks in such systems need to be synchronized therefore a good concurrency control or synchronization policy needs to be implemented.

Embedded systems have a functional perspective which can be developed using conventional Object-Oriented programming but there is also a real-time perspective which includes scheduling policies, and synchronization mechanisms, which is better implemented using Aspect-Oriented Software

Development concepts. AOSD (Kiczales et al., 1997) is a programming technique which enhances existing programming techniques by allowing the description of components that cross-cut numerous modules of the system. Instead of developing code for each module where a functional component is encountered, an aspect is developed and then code is injected in the appropriate locations using an aspect weaver. The aspect weaver may increase the code size but the gains obtained by removing redundant code from the different modules are sometimes more than the addition of the aspect weaver as described in (Kiczales et al., 1997).

When applied to the realm of real-time embedded systems, a developer that wants to use AOSD techniques needs to take into account the characteristics of such systems as described previously. This paper is an overview of research done in the field of real-time and embedded systems using Aspect-Oriented techniques and more importantly in the realm of task scheduling, resource management and synchronization.

The rest of the paper is organized as follows. In Section 2, the paper discusses research done on task scheduling in real-time and embedded systems and how AOSD may be utilized to enhance the scheduling aspect. Then, in Section 3, the resource allocation aspect is discussed and various research papers that approach that aspect are examined. In Section 4 the paper investigates synchronization using AOSD. Section 5 summarizes conclusions made in the review of the research done in the area and also presents opposing views to the use of AOSD in real-time and embedded systems. It also provides information on future work in the area investigated by the authors.

2 SCHEDULING

As discussed earlier, Cooling (Cooling, 2002) addresses scheduling in a real-time system as an important feature in any real-time software system. Real-time operating systems run a number of processes and these processes need to be scheduled correctly to achieve both timeliness and criticality issues. There are several problems with designing schedulers for embedded and real-time systems like the need for more flexible policies which are not hard-coded in the system. There are a number of research papers that deal with scheduling using AOSD techniques (Nyström et al., 2003, Tesanovic et al., 2004). This section introduces the existing research on the area of scheduling in real-time systems using AOSD and then discusses the results obtained from these papers. Since most of the research done in this area deals with real-time systems and not with embedded systems a discussion of how this research may be applied in embedded systems and what are the advantages and disadvantages will also be presented.

2.1 Existing literature

2.1.1 COMET and ACCORD

The Component-based Embedded Real-Time Database (COMET) (Nyström et al., 2003), is a real-time database application which uses a design method developed by the authors to demonstrate the concept of "aspectual component-based real-time system development", ACCORD (Tesanovic et al., 2004). ACCORD aspects are divided into application aspects which change components to suit a particular application, run-time aspects that provide information to the run-time system and composition aspects which provide information about components and how they can be extended or combined with other application aspects. Because components are black-box implementations but aspects require white-box implementations to capture certain join points in the code the authors developed "grey" components which allow aspects to alter their behaviour. The authors implement seven basic components that are used in real-time systems and present the different aspects that crosscut these components with real-time scheduling being one of them. The Transaction Scheduler that COMET implements can support various scheduling policies such as EDF and RM. In their conclusions, the authors state the advantages and disadvantages of using aspects and components in developing an application such as COMET. Advantages include the high reusability of code which is apparent in the definition of the system's Concurrency Control as an aspect and its ability to interact with other components such as the Transaction Scheduling Component, the Locking Component and the Transaction Management Component. Additional advantages include the ability of a component to fit specific requirements and the good separation of parts into functional components and aspects allowing the reconfiguration of the aspect part. The authors state that good composition rules need to be followed because of the large number of components and aspects which may be combined in numerous ways. Another disadvantage is the large code overhead that may appear due to the number of

mechanisms implemented in the components to support the weaving of aspect code.

2.1.2 BOSSA

In (Barreto and Muller, 2002) the authors identify the demand for new scheduling policies and implement a new Domain-Specific Language named Bossa. Because of the crosscutting of the scheduling process to numerous parts of the operating system kernel, the authors suggest the process's evolution to a modular component. To modularize scheduling, the authors identified scheduling points, such as the creation and destruction of processes or the change in the priorities of a process. An AOSD framework may capture these join points and control the Bossa scheduler by adapting the code to represent a certain scheduling policy. An event-based AOP framework EAOP (Douence et al., 2001) is identified by the authors as being able to "smoothly" integrate the Bossa DSL. This framework allows dynamic weaving of code. In this way Bossa may use EAOP to capture events such as process creation, termination or blocking and then execute scheduling code as part of the aspect. The paper concludes that AOSD techniques may be utilized in a variety of kernel subsystems and therefore they can help in developing an Aspect-Oriented operating system. A number of schedulers have been developed using the Bossa DSL such as Borrowed Virtual Time (BVT) (Duda and Cheriton, 1999) and Best (Banachowski and Brandt, 2002).

2.1.3 Open Layered Aspect Moderator Framework

In (Netinant et al., 2001), the authors discuss how AOSD can be used in designing Operating Systems. In their paper, the authors introduce a layered way of modularizing operating systems. They further identify the fact that operating systems implement numerous crosscutting concerns such as synchronization, scheduling and logging. Code for addressing these concerns is spread throughout the modules in an operating system restricting the developers' ability to easily change the functionality of such a concern. The authors introduce an Open Layered Aspect Moderator Framework which will allow developers to modify policies for crosscutting concerns. This is done through the use of an Aspect-Moderator Framework (AMF) where components have no aspectual properties, a proxy captures methods and sends them to the Aspect Moderator which in turn selects the appropriate aspect rules and strategies from an Aspect Bank and weaves that code during runtime. When considering the scheduling concern, developers can change from one scheduling policy to another by simply choosing a policy from the Aspect Bank or adding a new policy. Even though the paper discusses operating systems in general most of the ideas presented in the Open Layered Aspect-Oriented System Framework apply to real-time operating systems as well. One should be careful when working with real-time systems because any change in code could negatively affect other areas the system which could prove disadvantageous.

2.2 Analysis

Implementing a scheduling algorithm in any operating system is hard because of the immensity of modules that interact with the kernel and the requirement of rebuilding the kernel and those modules with aspects in mind. In real-time embedded systems though, the number of modules are reduced and most of the time operating systems for real-time systems are custom-built allowing the designers to utilize aspects to represent schedulers. Additionally, the uniqueness of each embedded system requires a way of modularizing various parts of the system so that the production is faster and cost-effective. There exists a wide variety of research on scheduling that can be studied and implemented using AOSD in order to come up with an efficient aspect-based scheduler for real-time embedded systems.

3 RESOURCE ALLOCATION

Real-time embedded systems and all systems in general do not involve one resource and one process but several resources with varying availability such as network, CPU, memory and energy. Resource Allocation Management deals with the system's ability to allocate resources to processes. A good resource manager should be able to manage resources, guarantee minimum demands, and smooth variations in streams of process arrivals. High performance real-time applications have to face challenges including dynamic changes in the environment, and limited resource availability (Rosu et al., 1997). The implementation of a resource allocation manager using AOSD is clearly apparent since this involves a number of resources which need to be efficiently managed to ensure timeliness of process execution and to reduce interference between processes competing for the same resource.

3.1 Existing Literature

3.1.1 Xelha

Introducing Quality of Service in middleware platforms is the objective of the authors of (Duran-Limon et al., 2003). In order to provide QoS, the authors present an architecture description language (ADL) called Xelha and a resource configuration description language (RCDL). The authors divide their resource model into resource managers and resource factories. In the design of the RCDL, the authors used aspect-oriented languages which allow the configuration and reconfiguration of distributed multimedia systems. An example of the aspects in RCDL is the Resource Description Language (RDL) defining resource templates which include worst-case-execution times, typical execution times, and amounts of resources used. Using this information the system knows what resources a certain service will require. Aspects provide the authors with a better separation of concerns giving the whole framework a more readable and reusable specification. Compared to Quality Objects project (QuO), the authors target an open framework rather than targeting CORBA and Xelha handles both fine-grained and coarse-grained interactions whereas QuO only handles fine-grained interactions.

3.2 Analysis

Resource allocation is a very important feature of any real-time and especially embedded system due to the scarcity of resources. To this extent the research done in (Duran-Limon et al., 2003) works in developing resource management techniques utilizing aspect-oriented paradigms. RCDL managed to separate the cross-cutting concerns in a system and allowed the straightforward definition of various services. One of the problems apparent in the design of both Xelha and RCDL was the need of a language interpreter whenever a change was made which added some overhead and another overhead was incurred when developing the middleware. In order for AOSD to prove beneficial these overheads need to be dealt with or at least prove insignificant to the overall gains of the system.

4 SYNCHRONIZATION

One important concern in the development of embedded systems is the synchronization or concurrency control function. Most of the embedded systems work in a distributed way and processes from various systems might require a specific synchronization policy to execute efficiently. There are several problems that can be tackled using synchronization policies such as a deadlock. This is more important in embedded systems which cannot recover in the same way as any other system. Due to the fact that processes from different modules need to be synchronized both in uniprocessor and distributed systems it is apparent that synchronization can be represented as an aspect.

4.1 Exiting Literature

4.1.1 Synchronization in CAN-Based Systems

In (Su and Singh, 2004) the authors discuss Controller Area Networks (CAN). A CAN is a communications bus which is responsible for sending and receiving short real-time control messages. Due to the increasing complexity of embedded systems, the authors define and implement aspect-oriented synchronization code which enables designers of CAN-based applications to develop functional components and then applying the necessary synchronization policies. The methodology used allows the developer of the system to implement the functional part of the system, identify the areas where synchronization will occur and finally set a global invariant which will control the synchronization policy that is to be used. In developing the system, the authors developed the code without taking into consideration any synchronization aspect. Then they identified synchronization regions which are code segments which depend on certain events to occur. For each of these regions global invariants are specified which will enable the incrementing of in and out counters. Finally, an aspect weaver was used to weave the synchronization code in the base code by intercepting the appropriate regions. An aspect-oriented methodology was used to develop two invariants; one that is centralized in nature and one that is decentralized. The authors found that the first approach was simpler and with less code in the CAN nodes but the second approach was more efficient due to a lower number of messages being transmitted and lower latency. One major advantage of using aspects in this project was the ability to change synchronization policies whenever the global invariant was changed. The aspect changed the code and weaved it back into the system with the new

policy allowing the development of more complex policies.

4.1.2 PURE system

AspectC++ was used by (Mahrenholz et al., 2002) to develop an aspect-oriented interrupt synchronization strategy. This synchronization strategy handles interrupts in an operating system by injecting calls to the synchronization primitives in the operating system code. Since code needs to be inserted in various areas in the operating system code it is clear that interrupt synchronization is a crosscutting problem that can be solved using AOSD. To eliminate the need of an aspect weaver the authors used compile-time aspect weaving and not runtime. In the paper the authors developed two variants of strategy using aspects; one which weaves code in every method that needs to be synchronized and one which uses methods grouped in subsystems and synchronization code is weaved only when a subsystem boundary is crossed. The second variant provides more robustness over the first variant because the use of subsystem groups offers higher abstraction. After implementing the variants a comparison of memory consumption was performed and the authors found that code size did not increase but rather decreased due to the fact that classes that were only used for synchronization were removed since the aspect handled those calls. This lead to more compact code which is very important in the development of embedded systems. The interrupt synchronization method described was used in the PURE operating system (Mahrenholz et al., 2002) and now the authors are implementing it in the CiAO project (Lohmann et al., 2005).

4.1.3 Application-Tailored Database Systems

In an embedded database, synchronization is a very important crosscutting concern which spans all of the database system. Data manipulation needs to be synchronized correctly to keep the database consistent therefore semaphores are used to protect data consistency. (Tesanovic et al., 2004b) presents the different crosscutting concerns in embedded databases and discusses how aspects can help in the development of an embedded database management system. The authors use the Berkeley DB, an embedded database system, and reengineer the application using aspect-oriented programming to tackle failure detection, synchronization and error-handling. The results of using aspects include a 42% code reduction and more comprehensibility since the synchronization aspect is no longer spread throughout the modules but centralized in as a single aspect code.

4.2 Analysis

Process synchronization is spread throughout the code in various modules therefore synchronization can be viewed as a crosscutting concern that can be expressed using aspect-oriented programming. It is apparent to the authors that aspects provide a lot of benefits to the developers of embedded systems and overall to real-time systems. These benefits include code size reduction, more comprehensible code as well as a more modular design because aspects can be seamlessly changed without interfering with the remaining code.

5 CONCLUSIONS AND FUTURE WORK

This paper shows how current scheduling, resource management and synchronization implementations can be enhanced by the use of AOSD. In schedulers, AOSD allows code reuse throughout several modules in the system and also provides the developer with the ability of adapting the scheduling policies used depending on the situation. Using AOSD techniques in the development of resource allocation managers can prove beneficial since resource usage is a crosscutting concern that spans all the modules of a system. Resource policies can be implemented as aspects and can be injected whenever they are required to determine efficient resource usage. Finally, since the synchronization aspect is spread throughout all the modules of a system, consolidating all the code in a centralized area provides a reduction in code in those modules that may prove beneficial even after the overhead of adding the aspect code and the aspect weaver needed to integrate the code.

In contrast to the advantages, the authors of (Tsang et al., 2004) evaluated aspect-oriented software development for use in java-based real-time systems development. Based on the seven areas of concern addressed by RTJava (Real-time Java) the paper designed a system both using Object-Oriented programming and Aspect-Oriented Programming and determined that using AOP improved modularity of the code but negatively affected system properties. This paper brings forth one of the main concerns of using AOSD in the design of software. If the trade-off between weaving

crosscutting concerns and the overhead they cause is not balanced or tilting towards the gains then AOSD will not be able to influence the real-time and embedded areas. Therefore more work needs to be done in identifying ways of using AOSD without inducing too much resource utilizations.

Trade-offs such as the increase in method calls may negatively affect the performance of a system. Nevertheless, the ability to remove unwanted scheduling code from within the various processes themselves could increase the effectiveness of the system in other areas, such as code size. Another advantage is the ability to change from one scheduling technique to another by simply replacing the aspect code and not interfering with other modules. This leads to better reusability of code since it can be used in a lot of areas depending on how it is developed.

Using AOSD in embedded systems is also beneficial due to the development of unique device drivers without prior knowledge of what non-functional concerns are going to be implemented. An embedded system can then combine the different modules and then implement an aspect to add further functionality to the system.

The authors of this paper feel that more research needs to be done for embedded systems and experiments should be performed to prove the benefits of using aspect-oriented software development. Furthermore, the trade-offs performed will be investigated to prove that an overall gain is observed.

REFERENCES

Banachowski, S. A. & Brandt, S. A. (2002) The BEST Scheduler for Integrated Processing of Best-Effort and Soft Real-Time Processes. Multimedia Computing and Networking (MMCN 2002).

Barreto, L. P. & Muller, G. (2002) Bossa: a language-based approach to the design of real-time schedulers. 10th International Conference on Real-Time Systems (RTS'2002). Paris, France.

Cooling, J. (2002) Software Engineering for Real Time Systems, Addison-Wesley.

Douence, R., Motelet, O. & Sudholt, M. (2001) A Formal Definition of Crosscuts. 3rd International Conference on Metalevel Architectures and Separation of Crosscutting Concerns. Springer-Verlag.

Duda, K. J. & Cheriton, D. R. (1999) Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. 17th ACM symposium on Operating systems principles. Charleston, South Carolina, United States, ACM Press.

Duran-Limon, H. A., Blair, G. S., Friday, A., Sivaharan, T. & Samartzidis, G. (2003) A Resource and QoS Management Framework for a Real-time Event System in Mobile Ad Hoc Environments. 9th IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2003F). Capri Island, Italy.

Huang, E.-H. & Elrad, T. (1998) Scheduling control mechanisms for managing indeterminate object behavior. ACM symposium on Applied Computing. Atlanta, Georgia, United States, ACM Press.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M. & Irwin, J. (1997) Aspect-Oriented Programming. European Conference on Object-Oriented Programming. Finland, Springer-Verlag.

Lohmann, D., Spinczyk, O. & Schröder-Preikschat, W. (2005) On the Configuration of Non-Functional Properties in Operating System Product Lines. 4th AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS 2005). Chicago, IL, USA.

Mahrenholz, D., Spinczyk, O., Gal, A. & Schröder-Preikschat, W. (2002) An Aspect-Oriented Implementation of Interrupt Synchronization in the PURE Operating System Family. 5th ECOOP Workshop on Object Orientation and Operating Systems. Malaga, Spain.

Netinant, P., Elrad, T. & Fayad, M. E. (2001) A layered approach to building open aspect-oriented systems: a framework for the design of on-demand system demodularization. Communications of the ACM, 44, 83 - 85.

Nyström, D., Tesanovic, A., Norström, C. & Hansson, J. (2003) The COMET Database Management System. MRTC Report. Mälardalen Real-Time Research Centre, Mälardalen University.

Rosu, D. I., Schwan, K., Yalamanchili, S. & Jha, R. (1997) On Adaptive Resource Allocation for Complex Real-Time Applications. 18th IEEE Real-Time Systems Symposium.

Su, Y. & Singh, G. (2004) Synchronization in CAN-based Embedded Systems. Embedded Systems and Applications. Las Vegas, Nevada, USA.

Tesanovic, A., Sheng, K. & Hansson, J. (2004) Application-Tailored Database Systems: a Case of Aspects in an Embedded Database. 8th International Database Engineering and Applications Symposium (IDEAS'04). Coimbra, Portugal, IEEE Computer Society.

Tsang, S. L., Clarke, S. & Baniassad, E. (2004) An Evaluation of Aspect-Oriented Programming for Java-based Real-time Systems Development. Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'04). Vienna, Austria.