Separation of Distributed Real-Time Embedded Concerns with Theme/UML

Cormac Driver, Vinny Cahill and Siobhán Clarke

Distributed Systems Group
School of Computer Science and Statistics
Trinity College Dublin
Ireland
{firstname.lastname}@cs.tcd.ie

Abstract

Model-driven engineering (MDE) addresses platform complexity by abstracting platform-independent models for subsequent transformation to platform-specific models. This facilitates the design of a single system model and the subsequent generation of multiple model transformations targeted towards specific platforms. However, the increasing complexity of distributed real-time embedded (DRE) systems complicates the development of adequate system models by requiring multiple concerns, some of which may be crosscutting, to be modelled. Separation of concerns, a software engineering technique that decomposes systems into distinct features with minimal overlap, can be used to manage complexity. Aspect-oriented software development (AOSD) is an emerging technique to separate crosscutting concerns in software and has been demonstrated to improve modularity and thereby reduce the complexity of software. In this paper[1] *we show how Theme/UML, an aspect-oriented design approach, can be used to better modularise DRE concerns at the model level.*

1 Introduction

Model-driven engineering addresses complexity arising from the need to port systems to multiple platforms by abstracting platform-independent models from which platform-specific models and code can be generated. The MDE focus on high-level abstractions and subsequent concrete transformations affords developers a number of benefits including time savings, code quality improvements and enhanced platform migration [6]. Most importantly, MDE enables the development of systems that are "correct-by-construction" [10].

While MDE can enhance the software development process, the increasing complexity of distributed, real-time embedded systems impacts on the ease with which adequate system models can be created using standard object-oriented modelling techniques. DRE system models are typically required to represent non-functional quality of service (QoS) concerns relating to, for example, time and performance, that have system-wide impact [9, 4, 12]. Such concerns are considered crosscutting because they affect the structure and behaviour of functional concerns and cannot be cleanly decomposed. The presence of crosscutting concerns in a system results in the tangling of behaviours that represent distinct concerns and therefore impacts negatively on system complexity.

Separation of concerns has long been recognised as a useful way to manage system complexity [5]. However, it has been established that total separation of concerns is not possible with the object-oriented soft-

ware development paradigm [8]. Aspect-oriented software development (AOSD) extends the modularisation capabilities of the object-oriented paradigm and has demonstrated effectiveness at modularising crosscutting concerns in DRE systems [7, 12, 1]. AOSD works by decomposing systems into modules that each represent a distinct concern and then binding or weaving the modularised concerns together to form a complete system.

Theme/UML [3] is an aspect-oriented design approach that extends standard UML. In this paper we introduce Theme/UML and show by means of example how it can modularise concerns at the modelling level, thereby reducing complexity in DRE systems.

The remainder of the paper is as follows. Section 2 presents an overview of Theme/UML. Section 3 illustrates how Theme/UML can be used to modularise a timing concern in an in-car driver information system and a memory management concern (discussed outside of any particular application scenario). Finally, Section 4 discusses current work.

2 Theme/UML

Theme/UML is part of the broader Theme approach. Theme is an integrated methodology for AOSD that covers the requirements analysis, design and implementation phases of the software development lifecycle [3]. Theme/UML extends standard UML to explicitly support the modularisation and composition of concerns in design. The associated methodology extends existing object-oriented design methods and provides guidance on how to use these extensions. The extensions to the UML include a new type of classifier called a *theme* and a new type of relationship called a *composition relationship*.

A theme is a package-like structure that encapsulates UML diagrams, e.g., class diagrams and sequence diagrams, that describe the structure and the behaviour of a concern. Themes representing crosscutting concerns are called *aspect themes*, while those that do not are called *base themes*. Theme/UML provides means to compose (e.g., merge or override) concepts and behaviours between themes.

Of particular relevance to this paper is the merging of behaviour from DRE aspect themes with that in base themes. The behavioural flow in an aspect theme is crosscutting, i.e., it is meant to be inserted in the behavioural flow of other themes. The composition of two flows belonging to different themes is achieved by identifying a trigger method in one theme that defines the actual place, known as the join point, where the other flow should be inserted. Join points from base themes are specified as part of the composition process. This behaviour allows crosscutting non-functional DRE concerns to be modularised and subsequently composed with specific points in the base behaviour of a system.

The following section illustrates how Theme/UML can be used to modularise two DRE concerns - timing of system behaviour and memory management.

3 Separating DRE Concerns with Theme/UML

This section shows how, using Theme/UML, it is possible to separate non-functional crosscutting concerns from core DRE system behaviour at the modelling level.

3.1 Timing Concern

A driver information system (DIS) is an automotive vehicle's display and information centre. A DIS is responsible for providing an interface to various aspects of a vehicle's current operating environment e.g., navigation instructions, collision avoidance warnings, climate control information and stereo and telephone data. Such systems can often be controlled via multiple user interfaces e.g., voice, touch and gesture.

DISs are composed of multiple services. Depending on a vehicle's operating context e.g., location and speed, it is desirable to constrain the execution times of certain services in order to favour more important services. For example, in the case that a vehicle is travelling at high speed in the presence of other vehicles, the collision avoidance service should be favoured over services such as the driver's mobile phone or the vehicle's stereo system.

An object-oriented design of the execution time monitoring behaviour in this scenario involves scattering and tangling execution monitoring code throughout the system so that the behaviour of each DIS service can be subjected to timing constraints. The execution timing concern is orthogonal to the core behaviour offered by each DIS service and therefore they should be modelled separately.

Figure 1 illustrates the design of the DIS using Theme/UML. The design consists of two themes. `DriverSystem` (on the right-hand side of the figure) is a base theme that represents the core behaviour of the DIS. The base theme contains a number of classes that represent a subset of common DIS services. Each class in the `DriverSystem` theme contains a method called `processEvent()` that is responsible for coordinating the behaviour of the service represented by the class.

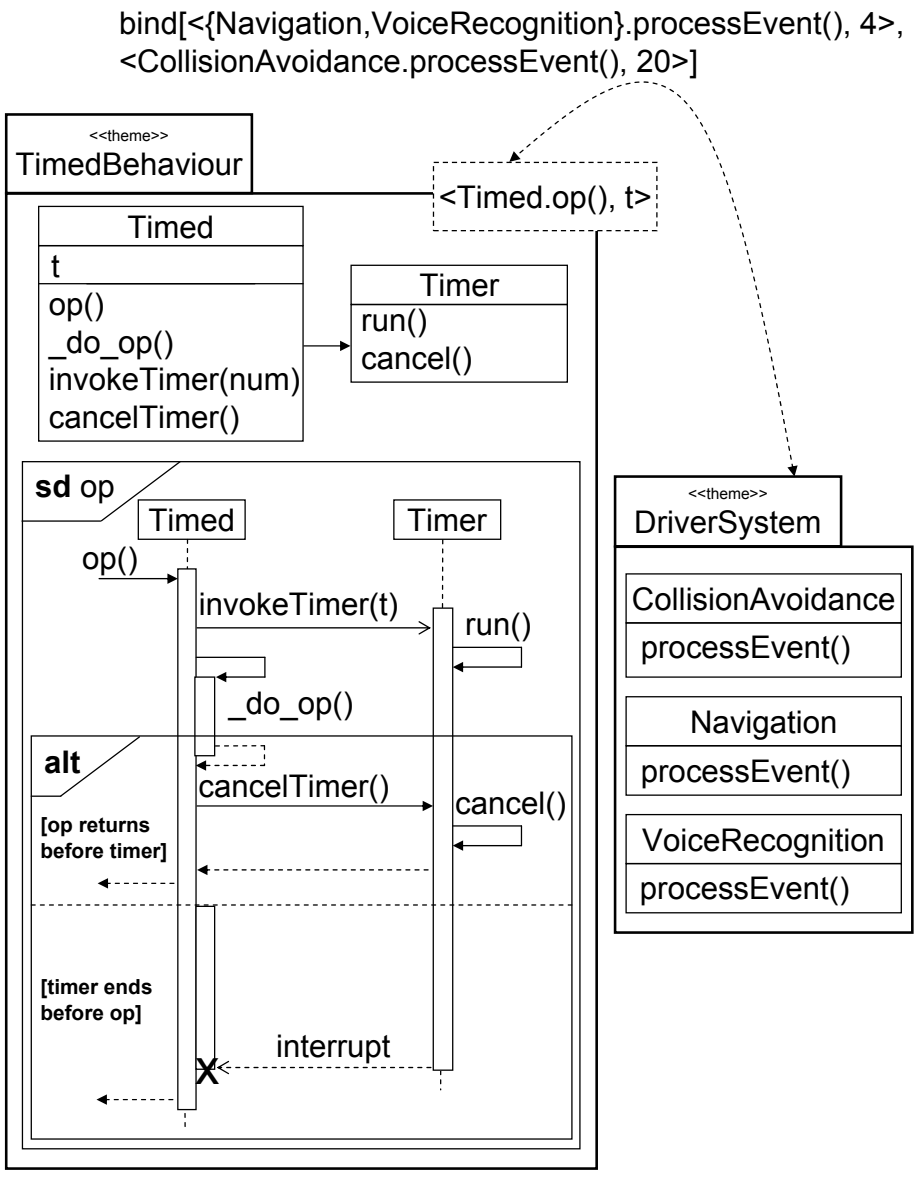

Figure 1. Separation of the timing concern in the driver information system using Theme/UML

TimedBehaviour is an aspect theme that is applied to DriverSystem. The aspect theme describes the structure and behaviour of the timing constraints to which the services in the base theme are subjected. The structure is described by two classes, Timed and Timer. Timed contains a trigger method called op() and timing behaviour to execute before and after the trigger method. Timed uses the services of a timer object which is modelled by the Timer class. Template parameters to a theme (including trigger methods) are illustrated in the top right-hand corner of the theme.

The behaviour of the aspect theme is modelled by a sequence diagram. Later in the development process, during composition, templates are replaced by base elements specified in a bind statement in the composition relationship between themes. The actual execution of this real behaviour is captured by prepending _do_ before the template trigger name. In this example, the TimedBehaviour sequence diagram illustrates two alternate behaviours. When the trigger method op() is invoked the timer is initiated for some period of time. The amount of time can be specified through a further template parameter called t. If the timed method (represented by _do_op()) returns within the specified time period then the timer is cancelled. Alternatively, if the trigger method fails to return within its allotted time then the timer will expire and interrupt the service that is currently executing i.e., the operation that triggered this timing behaviour.

The aspect and base themes are associated with each other through the bind tag of the composition relationship, located at the top of Figure 1. The bind tag identifies which methods in the base theme cause the behaviour in the aspect theme to be triggered i.e., which methods replace op(). The bind relationship specifies that the processEvent() method in each class in the base theme triggers the timing concern. Specifically, it specifies that all services apart from collision avoidance are restricted to 4 units of execution time e.g., milliseconds, while the collision avoidance service is favoured with an execution restriction of 20 units of time.

3.2 Memory Management Concern

Real-time, embedded systems typically cannot tolerate the unpredictable delays associated with garbage collection. Consequently, programming languages targeting real-time embedded platforms provide techniques for accessing memory that avoid the use of automatic garbage collection. The use of these memory management constructs generally involves manually allocating and deallocating memory as necessary, before and after core system functionality. This results in a situation where non-functional memory management behaviour is tangled with the system's functional behaviour. Memory management can therefore be considered a crosscutting concern and has been previously identified as such [11, 12, 8].

Figure 2 illustrates the Theme/UML design of a memory management concern that can be applied to any system behaviour that requires access to memory. BaseSystem is a base theme that represents a collection of core system behaviours. The theme contains a single module that has one method responsible for executing core system functionality involving manipulation of variables. This method requires the application of memory management behaviour to allocate and deallocate physical memory for its variables.

MemoryManagement (on the left-hand side of the figure) is an aspect theme that is bound to the base theme. The aspect theme describes the structure and behaviour of the separated memory management functionality. The theme's structure is defined by the MemMan class which contains methods responsible for the allocation and deallocation of memory. This class also contains a trigger method, op(), that performs the same function as the trigger method in the timing aspect discussed in Section 3.1 i.e., it is bound to a method in the base theme and triggers the execution of the behaviour in the aspect theme. The template parameters to the aspect theme are shown in the top right-hand corner of the theme and specify that methods in the base theme can map to op() in the aspect theme.

Beneath the MemMan class in the aspect theme is a sequence diagram that models the behaviour of the memory management concern. When the template method op() is invoked, the alloc() method is executed to allocate memory. When the actual behaviour from the base theme, represented by the _do_op() method, has terminated, the dealloc() method is invoked to deallocate the memory. The aspect theme's behaviour terminates when the memory has been deallocated.

The base and aspect themes are composed through the composition relationship specified by the bind tag, which is positioned above the base theme in Figure 2. The bind tag illustrates that the sole method in the base theme is composed with the memory management behaviour i.e., method() replaces op(). In cases where the base theme consists of more than one module with one method i.e., the vast majority of cases, the bind tag could be extended in the same manner as is shown in Figure 1 to specify that the memory management behaviour should also be applied to other methods in the base theme.

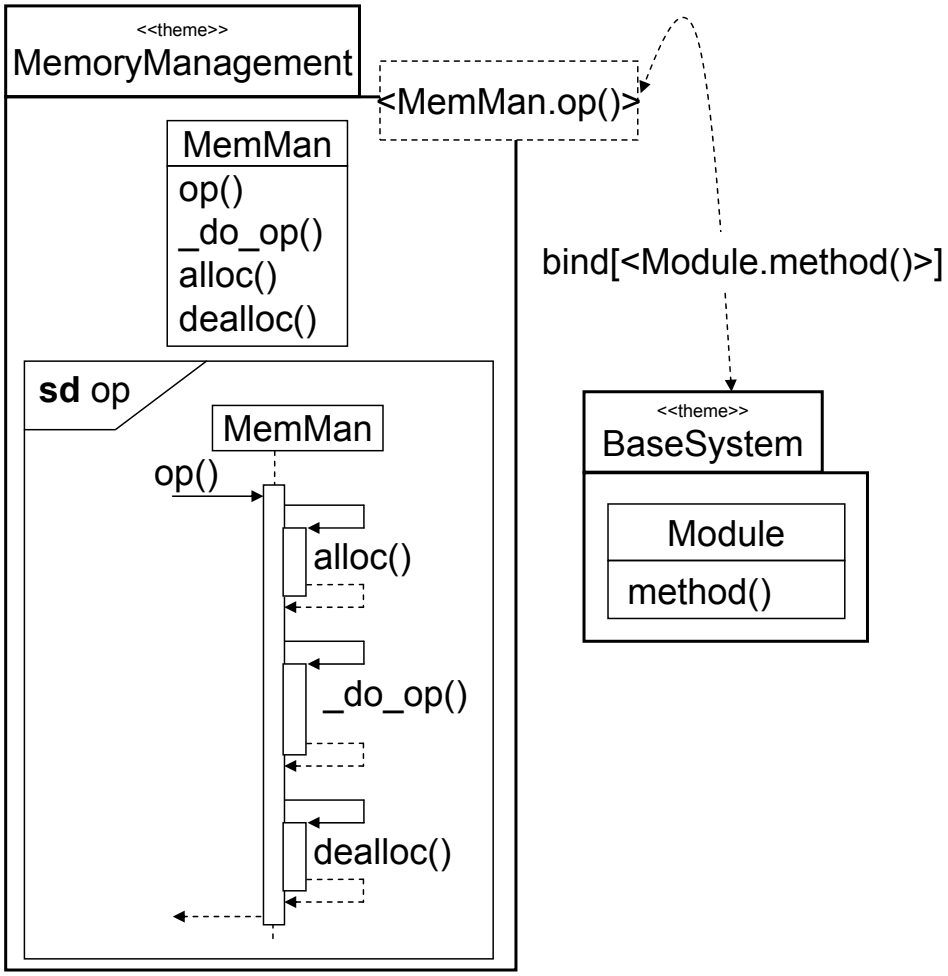

Figure 2. Separation of the memory management concern with Theme/UML

3.3 Summary

The design of the timing and memory management concerns using Theme/UML illustrates that aspect-oriented design can better modularise core behaviour by extracting non-functional requirements and encapsulating them in aspect themes. The crosscutting behaviour can then be merged with the core behaviour in base themes as necessary. The parameterisation of the bind relationship illustrates how it is possible to alter the crosscutting behaviour that is applied to the base as appropriate.

4 Current Work

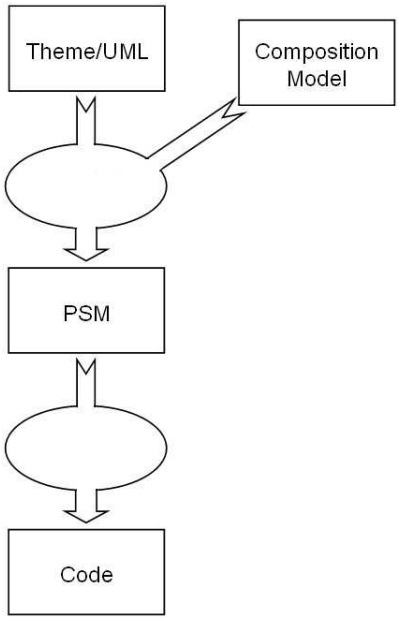

Figure 3. Model-driven engineering with Theme/UML

We have recently developed tool support for Theme/UML model transformations targeting pervasive computing environments, specifically the J2ME and .NET CF platforms [2]. The tool produces platform-specific transformations using the Eclipse Modelling Framework and has been shown to aid separation of concerns during the model-driven development of mobile, context-aware systems. We are currently investigating revisions to this approach with a focus on supporting both the modularisation of DRE concerns at the model level and transformations to embedded platforms.

Figure 3 depicts an overview of the application development process that the will be facilitated by the new tools we are developing. Theme/UML, along with an accompanying composition model, is used to design well modularised platform-independent models. These models are used to automatically generate platform-specific models (PSM) e.g., C, C++ or Real-Time Java models, by transforming the Theme/UML platform-independent meta-model to a platform-specific meta-model. The PSMs are then automatically transformed to platform-specific code.

Our previous work used the Java Emitter Templates (JET), an Eclipse Modelling Framework-based technology, to transform PSMs to executable source code. We are currently investigating alternative approaches to code generation that meet our requirements in terms of transforming models to code for deployment on embedded platforms. The platforms our Theme/UML MDE tool will target include C and C++.

5 Acknowledgements

We would like to acknowledge the support of Lero: The Irish Software Engineering Research Centre, funded by Science Foundation Ireland.

References

[1] F. Afonso, C. Silva, S. Montenegro, and A. Tavares. Applying aspects to a real-time embedded operating system. In *ACP4IS '07: Proceedings of the 6th workshop on Aspects, components, and patterns for infrastructure software*, New York, NY, USA, 2007. ACM Press.

[2] A. Carton, S. Clarke, A. Senart, and V. Cahill. Aspect-Oriented Model-Driven Development for Mobile Context-Aware Computing. In *SEPCASE '07: Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments*, Washington, DC, USA, 2007. IEEE Computer Society.

[3] S. Clarke and E. Baniassad. *Aspect-Oriented Analysis and Design: The Theme Approach.* Addison-Wesley, 1st edition, March 2005.

[4] G. Deng, D. C. Schmidt, and A. Gokhale. Addressing crosscutting deployment and configuration concerns of distributed real-time and embedded systems via aspect-oriented & model-driven software development. In *ICSE '06: Proceeding of the 28th international conference on Software engineering*, pages 811–814, New York, NY, USA, 2006. ACM Press.

[5] E. W. Dijkstra. *A Discipline of Programming.* Prentice Hall, 1976.

[6] A. Hovsepyan, S. V. Baelen, B. Vanhooff, W. Joosen, and Y. Berbers. Key Research Challenges for Successfully Applying MDD Within Real-Time Embedded Software Development. In *SAMOS 2006: 6th International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 49–58, 2006.

[7] A. Iqbal and T. Elrad. Modeling Timing Constraints of Real-Time Systems as Crosscutting Concerns. In *Proceedings of the 9th International Workshop on Aspect-Oriented Modeling (AOM)*, 2006.

[8] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In M. Akşit and S. Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242, Berlin, Heidelberg, and New York, 1997. Springer-Verlag.

[9] M. Panahi, T. Harmon, and R. Klefstad. Adaptive Techniques for Minimizing Middleware Memory Footprint for Distributed, Real-Time, Embedded Systems. In *Proceedings of the IEEE 18th Annual Workshop on Computer Communications*, pages 54–58., 2003.

[10] D. C. Schmidt. Model-Driven Engineering. *IEEE Computer*, 39(2):25–31, February 2006.

[11] S. L. Tsang, S. Clarke, and E. Baniassad. An Evaluation of Aspect-Oriented Programming for Java-Based Real-Time Systems Development. In *ISORC 2004: 7th IEEE International Symposium on Object-oriented Real-time distributed Computing*, pages 291–300, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[12] M. A. Wehrmeister, E. P. Freitas, C. E. Pereira, and F. R. Wagner. An Aspect-Oriented Approach for Dealing with Non-Functional Requirements in a Model-Driven Development of Distributed Embedded Real-Time Systems. In *ISORC '07: Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, pages 428–432, Washington, DC, USA, 2007. IEEE Computer Society.