Design Reuse for Real-Time Systems

Pete Cornwell, Andy Wellings

Department of Computer Science,
University of York,
Heslington,
York,
YO1 5DD,
UK

Tel: (+44) 0904 432711
Fax: (+44) 0904 432767
Email: cornwell@minster.york.ac.uk

Abstract

Real-time code is often structurally dependent on underlying hardware making it a poor candidate for reuse. High-level languages aimed at real-time systems do little to alleviate this problem, indeed they often impose simplistic and inflexible models on development that further prohibits portability and reuse. We believe that a component-based approach to reuse is better achieved by working at the design level, specifically within an object-oriented framework. Ideally we wish to develop a reuse approach that facilitates the specification, evolution and composition of real-time components.

Our chosen notation is HRT-HOOD, an object-oriented method for real-time system design developed at York. Through HRT-HOOD our wish is to develop a fully object-oriented architecture-centred reuse approach, integrating design with formal techniques to model and reuse the collaborative relationships between real-time components. Architectures provide a reusable building plan for real-time system development to which the collaborative real-time behaviour of components must conform in order to participate within the architecture. Adaption of both architectures and components to meet changing functional and non-functional requirements, is made possible through the use of two inheritance schemes that address the incremental specialisation of real-time collaborative structures and individual objects.

Keywords: Reuse, Real-Time, Design, Architectures.

Workshop Goals: Discussion of our ideas; Exposure to interesting and relevant work by others;

Working Groups: Reuse and OO methods, Design Guidelines for Reuse, Reuse and Formal Methods.

1 Position

We believe that code is an inadequate form of representation for the reuse of high performance real- time components. Real-time code often has strict functional and non-functional requirements encompassing such diverse properties as concurrency, fault tolerance and distribution - which are further bounded by hard or soft temporal constraints. A "correct" real-time implementation is where the specified functional and non-functional requirements are met with respect to the constraints imposed by the underlying hardware environment. In cyclic executive based systems that still underpin the vast majority of real-time applications a successful resolution of requirements against constraints, made possible through timing and schedulability analysis tends to result in context-dependent code structured primarily to meet high performance criteria. As a consequence such systems do not adequately address the needs of comprehension and maintainability resulting in code that does not lend itself well to reuse.

High level languages[HLL's], with the abstract modelling facilities they offer, tend to detract from rather than enhance the reuse potential of real-time code. Abstract constructs for expressing real-time characteristics often belie simplistic and inflexible implementation models. To work around these limitations either compiler modification or the use of supplementary low-level code is a very real possibility; such alteration ties code closely to the underlying hardware, limiting the possibilities for reuse. The alternative restricts code to a particular model of real-time behaviour consistent with its underlying implementation model.

In other areas HLL real-time reuse support is correspondingly weak. The classic "plug and socket" approach to composition, matching procedural "plugs" to "sockets" presented in the public interface of an external component is totally inadequate for a real-time approach. The classic "black box" approach to HLL component composition does not address performance issues, which remain hidden.

2 Design Reuse

Our work attempts to address reuse at the design level by identifying those real-time characteristics that are reusable and capturing these at an abstract level, independent of a particular implementation strategy. Object-oriented design is our favoured development paradigm. The advantages of object- oriented components are well known, but they also present a number of key properties useful for the modelling of real-time systems. These are:

- Concurrency : the abstract association of process and object
- Fault Tolerance : An object defines a useful containment boundary for error detection and recovery.
- Timing : temporal associations can be made with be public and internal methods.
- Distribution : The granularity of an object is ideal for distribution.

2.1 HRT-HOOD

Our favoured notation is Hard Real-Time Hierarchical Object-Oriented Design (HRT-HOOD), a variant of HOOD, developed at the University of York [1, 2]. HRT-HOOD defines a set of object types, each of which model a key real-time abstraction, these are:

- Cyclic - an object with periodic activity.
- Sporadic - an object with aperiodic activity.
- Active - a non-real-time concurrent object.
- Protected - a non-concurrent object that enforces mutual exclusion on shared resources.
- Passive - a non-concurrent object.

These abstractions are supported by a rich set of synchronous and asynchronous object interaction mechanisms, allowing a wide spectrum of inter-object behaviours to be modelled. In order to maximise the reuse of detailed real-time design information, we require a scheme of design representation that effectively captures key functional and non-functional real-time requirements in a format that is implementation-independent, and sufficiently concise and abstract in order to promote rapid comprehension. To fulfil this aim we are currently evaluating a wide range of real-time formal notations, as a basis for the capture of detailed design information with HRT-HOOD.

From this basis we are attempting to apply an object-oriented approach to reuse that encompasses real-time characteristics as well as purely functional. We have updated the original HOOD object-based development scheme with an approach that is wholly object-oriented, allowing the derivation of real-time class types from which objects may be instantiated. As HRT-HOOD is a decompositional approach we have designed two complimentary inheritance schemes:

- Architectural Design Inheritance [ADI] - a multiple inheritance scheme designed to allow the incremental specialisation and combination of abstract architectural class types at non-terminal levels of a design decomposition. ADI allows the derivation of new class types that combine the archtectures of each individual parent into a single abstraction. From this basis new child objects may be incrementally added to the derived class type, which may draw upon and use the services and resources inherited from the respective parent architectures.
- Detailed Design Inheritance [DDI] - This is a multiple inheritance scheme for the incremental specialisation of detailed class types at the lowest or terminal level of a design decomposition. In contrast to ADI, DDI is a scheme closely related to inheritance mechanisms at the implementation level. Its functional aspects allow the derivation of new class abstractions conceptually built upon resources and methods provided by one or more parent class types. From this inherited basis a derived class type may incrementally add new resources and services, specialising the new abstraction to encompass extended functional requirements.

For real-time applications, the purely functional orientation of inheritance does not encompass the specialisation and reuse of important real-time characteristics. Through ADI we have introduced a scheme of architectural specialisation that allows the incremental addition of new real-time object abstractions to a non-terminal class type. Extending inheritance at the most detailed level of development to encompass issues such as concurrency, timing and fault tolerance is vital if we are to capitalise on the advantages for reuse offered by this valuable mechanism of class type extension and specialisation. DDI will therefore be the basis for the adaption of real-time components to meet new functional and non-functional requirements. We are currently developing a theoretical basis for the inheritance of concurrent threads of control within the constraints of a real-time framework, and hope to extend this to address both timing and failure issues in the near future.

2.2 Architectures

Ultimately we hope to place these ideas within an architectural reuse framework; "No object is an island". We believe that greater reuse can be achieved by modelling the collaboration relationships between objects as well as the component objects themselves. Complimentary to HRT-HOOD we hope to produce a scheme that is both decompositional and amenable to the inheritance principles outlined above. Our architectures are to be composed of slots, semantic specifications that define both the individual and collaborative behaviours that a reusable component must fulfill in order to occupy that slot. Our architectures will exhibit the following static characteristics:

- Structural - Outlining the slots of the architecture and the paths of interaction between them.
- Compositional - Allowing slots to be decomposed into child architectures, that collectively fulfil the semantic requirements of the parent slot.
- Distributive - How the slots are partitioned onto physical processors.

Overlying the static structure are the following real-time functional and non-functional characteristics:

- Concurrent - Defining concurrent behaviour within and between slots.
- Fault Tolerance - Defining a consistent failure strategy in and between slots.
- Functional - Defining individual and collaborative functional characteristics.
- Temporal - Defining timing requirements in and between slots.

Slots are "typed" according to their real-time characteristics and will conform to one of the HRT-HOOD object types aforementioned. An architecture loosely conforms to a HRT-HOOD object at a non-terminal level of decomposition, containing slots that may in turn be decomposed into respective child architectures until the terminal level of the hierarchy is reached. Architectures at various points on the decompositional hierarchy are ideal units of reuse and may be extracted from their "native" environment and composed to develop new systems and subsystems. Architectures are composed together by common components that demonstrate compliance with the syntactic and semantic requirements of a slot in each subsystem. In other words a component may play different roles in distinct architectures, its very existence uniting them to form a coherent whole. In the highly concurrent, asychronous systems that will come to increasingly dominate real-time development an ideal candidate for architectural composition is the protected object, a monitor-like construct encapsulating shared resources and the basis for communication between cyclic and sporadic components. By capturing collaborative patterns of behaviour we ease the composition problem. Architectures define a building plan only allowing a component to participate if it fulfills the syntactic and semantic requirements of a particular slot. This is a significant shift in emphasis from the classic component-centred approach to reuse and ideal for the modelling of real-time systems where the capture of high performance collaborative behaviour is vital.

3 Comparison

1. Contracts [3] : The authors provide a textual formalism for the capture of collaborative behaviours between inter-communicating objects called a contract. A contract is composed of participants that define the functional behaviour and type obligations that a class must fulfil in order to be incorporated into the contract. The authors provide a formalism, called a conformance declaration for the mapping of class inheritance trees to contract participants. Contracts may be built in a "bottom up" fashion through the aggregation of existing contracts, or modified through contract refinement, a scheme that allows the incremental specialisation of one or more contract participants. Contracts essentially define architectural classes that are instantiated by existing class components. The work of Helm, Holland and Gangopadhyay is fundamental in the area of architectural specification and reuse and contains a number of important concepts that we would hope to build upon in our own work. The contract notation however does not explicitly address real-time architectural specification, an area which we hope to investigate as part of future work.

2. Architectures With Pictures [4] : The authors present a common conceptual framework for the development of real-time architectures through simple graphical notation. Architectures are broadly classified as wired , with static paths of communication between objects or wireless where the links are both dynamic and potentially transitory. Architectures are composed of placeholders which define the semantic and syntactic properties a component must exhibit in order to participate within a collaboration. Placeholders are dynamic in the sense that they may be occupied by different object components over time, furthermore they are subject to hierarchical decomposition. Collaborative behaviour is defined by a contract [3](i), and visualised through a simple timethread notation. Timethreads are paths of execution along which time monotonically advances. These paths are traced through the design and represent an order of execution, terminating when processing is complete. Buhr and Casselman consider the development of real-time object-oriented architectures in some detail. However, their approach does not appear to explicitly address how inheritance, the primary mechanism of object-oriented reuse, can be integrated into a real-time architectural and component based design framework.

References

[1] A. Burns and A. Wellings, "Hard Real Time HOOD: A Design Method for Hard Real-time Ada9X Systems," *Ada U.K. Technical Papers*, 1991.

[2] A. Burns and A. Wellings, "Hard Real Time HOOD: A Design Method for Hard Real-time Ada," *Journal of Real-Time Systems*, vol. 6, pp. 73–114, 1994.

[3] R. Helm, I. Holland, and D. Gamgopadhyay, "Contracts: Specifiying Behavioural Compositions in Object-Oriented Systems," *ACM SIGPLAN Notices*, vol. 25, pp. 169–180, October 1990.

[4] R. Buhr and R. Casselman, "Architectures with Pictures," *ACM SIGPLAN Notices*, vol. 27, pp. 466–483, October 1992.

4 Biography

Pete Cornwell is a graduate from Newcastle University and Bournemouth Polytechnic, where he received a MSc in Software Engineering in 1991. At the beginning of 1992 he began work as a Research Associate in Structured Design Methods for Real-Time Systems at the University of York, working on the HRT-HOOD project with Dr. Andy Wellings and Dr. Alan Burns. His research interests encompass design reuse, architectures, formal methods and reverse engineering. He is also registered as a Phd student at the University of York.

Dr. Andy Wellings is a senior lecturer in the department of computer science at the University of York. His research interests cover a number of aspects of real-time system development including design, programming languages and distributed operating systems. He is the co-developer of the HRT-HOOD design notation (with Alan Burns), and has published over 100 technical papers and reports including three textbooks.