Extensible Systems - The Tigger Approach.

Vinny Cahill[*], Christine Hogan, Alan Judge, Darragh O'Grady, Brendan Tangney, Paul Taylor
Distributed Systems Group, Dept. of Computer Science,
Trinity College, Dublin, Ireland

Abstract

The Tigger project is developing a framework for the construction of a family of distributed object-support platforms suitable for use in a variety of distributed applications ranging from embedded soft-real time systems to concurrent engineering frameworks. As no one system can easily meet the varied demands of these different application areas, customisability, extensibility and portability are put forward as the way to handle diversity and are thus the core design goals in Tigger.

1 Introduction

The Tigger project is developing a framework for the construction of a family – the Tigger Pride – of distributed object-support platforms suitable for use in distributed applications ranging from embedded soft-real time systems (actually 3-D arcade and console video games) to concurrent engineering frameworks. Members of the Tigger Pride are expected to be hosted on top of bare hardware, (real-time) micro-kernels and conventional operating systems. Thus customisability, extensibility and portability are major design goals of Tigger in addition to distributed object support.

The baseline for the Tigger project is a set of minimal object-support platforms – known as Cubs – supporting at least four primitive abstractions: distributed objects; persistent objects; activities (i.e. distributed threads of control) and extents (i.e. protected collections of objects). A given object may be both distributed and persistent.

Members of the Tigger Pride may provide additional abstractions supporting, for example, security and transaction services. In addition, members of the Pride may be specialised to support different policies. For example, distributed objects will be supported using both RPC (function shipping) and/or DSM (data shipping) techniques. The result is that a Tigger which provides the necessary services for the target application domain can be constructed.

Each instantiation of the Tigger framework – the Cubs and all other Tiggers – is intended to provide the necessary support for the use of some object-oriented language(s) for the development of distributed and persistent applications. Thus the fundamental interface provided by a Tigger is that provided for the language implementer. The interface used by an application developer is that provided by a supported language.

2 Rationale

The rationale for the Tigger project arose out of the evaluation of the Amadeus object-support platform [5].

Amadeus was a general-purpose object support platform for distributed and persistent programming in multi-user distributed systems. Amadeus was targeted for use in what may broadly be described as cooperative applications concerned with access to shared data in domains such as computer-aided design (CAD), office automation and software engineering. Amadeus has been used to support applications such as ray-tracing, simulation and a CAD framework.

The Amadeus platform was designed to support the use of a range of object-oriented languages for the construction of distributed applications. A language could be extended to support a set of (inter-related) properties including distribution, persistence and atomicity for its objects by using the services of the Amadeus Generic Runtime (GRT), while maintaining its own native object reference format and invocation mechanism [2]. The GRT provided a range of mechanisms from which the language designer could choose those

[*]E-mail: **vinny.cahill@dsg.cs.tcd.ie**

appropriate for the intended use of the extended language.

Evaluation of Amadeus lead to a number of conflicting requirements. While the range of mechanisms supported by the Amadeus GRT was sufficient to support the extension of conventional, non-distributed, non-persistent programming languages, an early conclusion was that additional mechanisms were required to support distributed and persistent programming languages [1, 2]. On the other hand, supported languages typically used only one out of the range of mechanisms provided to support distributed and persistent objects. Moreover, applications used only a limited subset of the functionality provided in any supported language.

In addition, the platform was rather heavyweight including as it did a high degree of functionality. This made the platform unsuitable for many potential application areas including embedded and real-time systems. The platform also placed substantial requirements on the underlying host operating system both in terms of functionality required and resource usage.

Amadeus evolved through a number of versions as functionality was added. These included versions with support for different GRT mechanisms, heterogeneity, transactions and security. In most cases these modifications were somewhat haphazard, propagated through the entire system and sometimes resulted in systems which were not backwards compatible thus requiring modifications to existing applications.

Finally, consideration of the implementation of the Amadeus platform showed that the implementation *itself* could have benefited substantially from the use of a (light-weight) distributed persistent object support platform as its components were often concerned with the manipulation of distributed, persistent data.

3 The Tigger Approach

From the evaluation of Amadeus the fundamental aspects of the Tigger approach became apparent:

- any general-purpose, language-independent, object support platform should be based on a *framework* describing the fundamental abstractions to be supported thereby allowing tailored implementations to be provided;
- this framework should be *self-hosting* thereby allowing instantiations to make use of distributed and persistent objects.

The interface supported by the Tigger framework is that required to support distributed and persistent programming languages and is a refinement of that of the Amadeus GRT. Instantiations of the framework implement specific mechanisms or collections of mechanisms. A key point is that multiple instantiations can coexist. For example, two different instantiations may be used to support two different languages requiring different mechanisms from the underlying platform while still supporting inter-language working.

4 The Tigger Cub

The defining characteristic of a Tigger Cub is that it is any instantiation of the framework which supports the language in which instantiations of the framework are implemented. In this sense the Tigger project is following the micro-kernel philosophy by providing a minimal system capable of hosting more sophisticated systems. Thus each Cub supports a version of C++ extended to provide distributed and persistent objects.

In Tigger the framework is a collection of abstract base classes. The instantiations are implemented in extended C++ by deriving classes implementing the required mechanisms and policies. Likewise a Cub is instantiated from the framework by choosing suitable mechanisms to support extended C++. Of course, not all the classes in the framework have distributed and persistent instances and these classes provide the fundamental support for other classes whose instances may be distributed or persistent.

A Cub may support additional languages if the mechanisms provided by that Cub are sufficient for those languages.

A Cub must provide at least four primitive abstractions: distributed objects; persistent objects; activities (i.e. distributed threads of control) and extents (i.e. protected collections of objects). In fact a Cub need only implement a single extent and the means of trapping accesses to objects which belong to other extents. The framework provides an interface for handling such accesses, the implementation of which in a Cub need only raise an exception, but which can be refined to implement multiple extents. Thus a Cub need only be a single-user system.

To summarise, there is no single unique Cub – any instantiation of the framework which can support the implementation language is a Cub. Where multiple instantiations coexist, one is always a Cub. Services required to support the Cub interface may be implemented in the extended C++ supported by the Cub. For example, the storage system used to store persistent objects is implemented in extended C++ and used by the Cub and other instantiations as necessary.

5 The Tigger Pride

New Tiggers can be implemented in one of two ways: by providing additional implementations of the fundamental abstractions or by supporting additional abstractions.

For example, a new Tigger can be created which supports a different implementation of distributed objects or a choice of implementations. Likewise new Tiggers can be created which support multiple extents (and hence basic security) or transaction support. New mechanisms can be implemented using the language supported by the Cubs i.e. using distributed and persistent objects.

As an example, to implement a Tigger which supports multiple-users requires that the Tigger framework be instantiated to support multiple extents, cross-extent invocation of objects and access checking. In addition, the framework must be extended to include (abstract base) classes, describing the management of multiple extents and users, and instantiated appropriately. These additional components may be implemented in the language supported by the Cubs as Tigger applications in one extent which acts as the "kernel" for other extents and which stores the control information that it uses – user descriptors and access control lists for example – as persistent objects belonging to the "kernel" extent.

One definite disadvantage of this approach is that there is still the danger, when instantiating a new Tigger, that it may not be backwards compatible with existing Tiggers. Although based on experience with Amadeus this is clearly undesirable, it appears to be nevertheless unavoidable.

6 Related Work

Apart from Amadeus, the design of Tigger has been influenced by a number of other projects including Choices [3] – which developed a C++ framework for the construction of operating systems for distributed and shared memory multiprocessors – and Peace [4] – which also addressed the use of object-oriented techniques for the construction of a family of operating systems for massively parallel computers. The Peace family encompasses a range of different members ranging from one supporting a single thread of control per node to one supporting multiple tasks per node. Tigger differs from these systems in the intended application areas, the abstractions supported and in the goal of direct support for language support as the primary interface to the system as well as in the specifics of the mechanisms employed by the system.

7 Conclusions

The Tigger project may be seen as an exercise in practising what you preach: the use of object-oriented design techniques and of distributed, persistent, object support platforms.

At the current time, an implementation of a single-user Cub hosted above the Mach micro-kernel is nearing completion. The next instantiation will be an implementation of another object-support platform providing functionality similar to Amadeus. Another version of the Cub hosted by a real-time kernel will be used in arcade and console video games.

References

[1] S. Baker. *System Issues in Persistent Programming and OODBMS Integration*. PhD thesis, Department of Computer Science, Trinity College, Dublin, July 1992.

[2] V. Cahill, S. Baker, C. Horn and G. Starovic. The Amadeus GRT — Generic Runtime Support for Distributed Persistent Programming. In *Proceedings of the 1993 Conference on Object-Oriented Programming, Systems, Languages and Applications*, ACM, 1993.

[3] R. H. Campbell, N. Islam and P Madany. *Choices*, Frameworks and Refinement. *Computing Systems*, 5(3):217–257, Summer 1992.

[4] J. Cordsen and W. Schroder-Preikschat. Object-Oriented Operating System Design and the Revival of Program Families. In *Proceedings of the 1st International Workshop on Object Orientation in Operating Systems*, IEEE, 1991.

[5] C. Horn and V. Cahill. Supporting Distributed Applications in the Amadeus Environment. *Computer Communications*, 14(6):358–365, July/August 1991.