SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2000; **30**:293–294

Editorial: Persistent object systems

Persistent object systems address the challenging requirements of long-lived, data-intensive applications. At their most ambitious, they aim to support simple, reliable and efficient access to large, shared bodies of structured data and programs, over extended periods of time. Such aims lie at the nexus of an ongoing convergence between historically distinct communities.

On the one hand the research community has concentrated its focus on orthogonal persistence, a programming language abstraction in which the longevity of data, including programs, is independent of the way in which it is created and manipulated. As such, orthogonal persistence integrates the database view of longevity of information with the programming language view of data abstraction and manipulation, all within a coherent system.

Meanwhile, fuelled by commercial interest in object-based systems, industry has worked to bring objects into mainstream products. Whether the context is storage and manipulation of Java™ or C++ objects, or support by vendors for complex data models in pure object or hybrid object-relational databases, the management of long-lived structured objects is now an important commercial opportunity.

The fusion of the persistence abstraction with database functionality such as transactions and crash resilience promises increased programmer productivity, application safety, and data integrity. The recent popularity of application programming languages such as Java™ that emphasise safety over raw efficiency will ensure continued interest in persistent object systems, as applications demand clean and efficient persistence solutions for the objects they create.

In this exciting context, we are privileged to bring this special issue on *Persistent object systems* to SP&E, comprising papers that run the gamut from low-level operating system support through language design and implementation, experimental evaluation of prototypes and evaluation methodologies, on up to high-level software architecture. We received 25 submissions, accepting six for this special issue, having observed the standard SP&E process of peer review. Two other submissions were recommended for separate publication in regular issues of SP&E. Authors of several of the remaining submissions were encouraged to revise and resubmit their papers for similar consideration.

The first paper, *Operating system support for persistent systems: past, present and future*, by Dearle and Hulse, surveys the history of explicit operating system support for the persistence abstraction, gives consideration to modern trends in operating systems, and expounds the philosophy and design of a new 'nano-kernel' operating system called *Charm*, whose explicit goals are flexible support of persistent applications via a minimal set of system primitives.

™Java is a trademark of Sun Microsystems, Inc.

The second paper attacks the problem of obtaining efficient fine-grained lock management for automatic concurrency control over access to shared objects in persistent programming languages. *Implementation of automated fine-granularity locking in a persistent programming language*, by Daynès, describes new techniques for minimising the performance impact of automatic locking, and evaluates their performance in a persistent extension of Java™.

Morrison *et al*., in *A compliant persistent architecture*, systematically address the performance problems often encountered in layered software architectures, by introducing a mechanism for customisation of lower layers in response to the expectations of higher layers. The idea is that the operational needs of each layer are transmitted to *compliant* layers lower in the architecture, which adjust their behaviour accordingly, achieved through systematic separation of policy from mechanism across all layers of the architecture. The paper examines an instantiation of this approach for persistence, demonstrating how it can operate in a manner compliant to a target application.

The fourth paper, *An infrastructure for generating and sharing experimental workloads for persistent object systems*, by Humphries *et al*., describes a trace format to express sequences of logical actions against (single-user) persistent object stores. It includes means to express the object store schema, object creation, and object mutation, including setting and reading object references, etc. The traces are designed to be rich enough to assess performance of different object store implementation mechanisms, including store garbage collectors. The authors also describe a tool that eases construction of benchmark applications to generate traces, which is less error-prone than instrumentation by hand. They include a discussion of their experiences using the tool to generate traces for the widely-known 007 benchmark.

In *Dynamo: design, implementation, and evaluation of cooperative persistent object management in a local area network*, Yang *et al*. present an architecture and protocols for cooperative caching in a decentralised 'server-less' setting. They outline their implementation of the *Dynamo* prototype system, in which they proceed to demonstrate the superiority of their new protocols for object replacement against competing global replacement policies.

Finally, *Fast portable orthogonally persistent Java*™, by Marquez *et al*., focuses on techniques for systematic transformation of Java™ bytecode executables to support transparent extensions of the Java™ programming language beyond its standard semantics. The authors apply these techniques in developing a persistent extension of Java™. The transformational approach imposes no changes on the underlying Java™ virtual machine, resulting in maximum portability of their implementation. The paper closes with a performance study characterising the performance of their persistent extension of Java™ via bytecode transformation, and comparing it to a competing implementation that instead relies on extensive modification of the Java™ virtual machine.

We are proud to bring this snapshot on the state of play in persistent object systems to SP&E, and thank all those individuals who helped it into being. The papers appearing here nicely capture the diversity of this vibrant area.

Antony Hosking
Department of Computer Sciences
Purdue University
West Lafayette, Indiana
U.S.A.

Quintin Cutts
Department of Computing Science
University of Glasgow
Glasgow
U.K.

