conference reports

This issue's reports are on the 6th USENIX Conference on Object-Oriented Technologies and Systems (COOTS '01)

OUR THANKS TO THE SUMMARIZER:

Nanbor Wang

6th USENIX Conference on Object-Oriented Technologies and Systems (COOTS '01)

San Antonio, Texas

January 29–February 2, 2001

Summarized by Nanbor Wang

Best Student Paper Awards

Yi-Min Wang of Microsoft Research, Co-Chair of the Conference, announced the best student award. Many of the accepted papers were written by students this year. Of them, two received identical scores, and both of them were awarded the best student paper award. They are:

"Content-Based Publish/Subscribe with Structural Reflection" by Patrick Th. Eugster and Rachid Guerraoui of the Swiss Federal Institute of Technology, Switzerland and

"Multi-Dispatch in the Java Virtual Machine: Design and Implementation" by Christopher Dutchyn, Paul Lu, Duane Szafron, and Steve Bromling of University of Alberta, Canada; and Wade Holst of the University of Western Ontario, Canada

Keynote Address

The Brave New World of Pervasive, Invisible Computing

Stu Feldman, Institute for Advanced Commerce, IBM T.J. Watson Research Center

In this talk, Feldman pointed out that, with their increasing popularity, portable computing devices have become pervasive in our daily lives. Nevertheless, unlike desktop computers, there is still no single computing model that dominates the mobile world nor should there be one since mobile devices have very different hardware interface designs. With the current trend, the number of mobile devices accessing the Web will shortly exceed that of desktops. There are plenty of opportunities to develop mobile devices, and Object-oriented (OO) technologies and middleware will be required to integrate pieces of software.

Feldman went on to point out that quality of service and immediacy will be important for next-generation mobile applications providing networked services, such as market monitoring, financial transactions, and real-time information broadcasting. The heterogeneity of the environment these applications must run on presents many challenges to developing these applications. The diversities in user demographics, device-specific user interfaces and business models processes of service providers further complicate the issue.

In conclusion, Feldman envisions a world of pervasive devices that are helpful, always on, always available, personal, friendly to use, easily authored, managed, secure, and reliable. It's up to the application and middleware developers to meet these challenges.

Session: Distributed Objects

TORBA: Trading Contracts for CORBA

Raphaël Marvie, Philippe Merle, Jean-Marc Geib, and Sylvain Leblanc, Laboratoire d'Informatique Fondamentale de Lille, France

Raphaël Marvie presented their work on the Trader-Oriented Request Broker Architecture (TORBA), which is a framework and a collection of tools to automate the use of the CORBA Trading Service. Marvie pointed out that locating a service is the essential part of distributed applications. Although the Object Management Group (OMG) already defined the CosTrading Service, it is complicated to use and does not support type safety. The proposed solution is to make the Trading Service an integral part of objects and provide tools to integrate the Trading Service into applications automatically.

TORBA defines the concept of *trading contracts* in TORBA Definition Language

(TDL). With the help of a TDL compiler, TORBA generates the code for *trading proxies*, which hide the complexity of using the OMG Trading Service and provide the necessary type-checking mechanisms. TDL allows application developers to specify the properties of an object and how the object will be offered to and queried for by its clients. Although TORBA introduces some extra operations when connecting and using the trading server, a local library is used to optimize its performance, and other ORB-specific optimizations can be used to improve the performance further.

For more information, contact *marvie@lifl.fr*.

Dynamic Resource Management and Automatic Configuration of Distributed Component Systems

Fabio Kon, University of São Paulo, Brazil; Tomonori Yamane, Mitsubishi Electric Corp.; Christopher K. Hess, Roy H. Campbell, and M. Dennis Mickunas, University of Illinois at Urbana-Champaign

Component technology is gaining popularity in today's computation environments. Software components increasingly collaborate with each other, and how they are configured often depends on the hardware that they run on. Managing software components is becoming complicated as dependencies among components and the number of hardware platforms they can run on grows.

Christopher Hess presented their work on a component configuration framework that provides mechanisms to help:

- Automatic configuration of component-based applications
- Intelligent, dynamic placement of applications in distributed systems
- Dynamic resource management for distributed heterogeneous environments
- Component-code distribution using either push or pull models
- Safe dynamic reconfiguration of distributed component systems

Hess also provided a performance analysis of their component configurator implementation.

For more information, contact *ckhess@cs.uiuc.edu*.

An Adaptive Data Object Service for Pervasive Computing Environments

Christopher K. Hess, Roy H. Campbell, and M. Dennis Mickunas, University of Illinois at Urbana-Champaign; Francisco Ballesteros, Rey Juan Carlos, University of Madrid

Christopher Hess continued to describe their work on the adaptive data object service (DOS). He first pointed out that remote data access is the most essential part in modern computing environments. The traditional file sharing mechanisms, however, are not suitable for many mobile devices and their computing model because they lack the necessary resources, such as bandwidth, memory, software, and CPU power to decipher the raw data stored in files. To address the problem, Hess's team proposes the adaptive DOS, which is inspired by the model-view-controller (MVC) model.

With DOS, data are represented and accessed through *containers* and *iterators*. Containers shield the actual file representations from the application. Data can either be remote or local. Containers can also be instantiated with *filters*, e.g., a GrepContainer, to change the user's view of a data stream. Iterators provide different ways to traverse the data in containers.

DOS is currently implemented as CORBA service and can be linked-in dynamically. XML descriptions are used to describe the capability of DOS objects, so a device knows what a DOS object is and how to use it. DOS is part of the Gaia system, which is an infrastructure that exports and coordinates the resources contained in a physical space and defines generic computational environments for ubiquitous computing devices.

For more information, contact *ckhess@cs.uiuc.edu*.

Session: Infrastructure

HBench:JGC – An Application-Specific Benchmark Suite for Evaluating JVM Garbage Collector Performance

Xiaolan Zhang and Margo Seltzer, Harvard University

In this paper, Xiaolan Zhang pointed out that as Java gains popularity several Java Virtual Machine (JVM) aspects, such as dynamic memory management and garbage collection (GC), can become problematic for performance sensitive applications. Understanding GC performance characteristics in order to select the right GC implementation can significantly affect overall application performance. Traditional GC benchmarking approaches may not measure the behavior of targeted applications accurately because they usually measure and compare the total execution times of a fixed set of applications using different GCs. These applications, however, may not have the same behavior of the application we are interested in.

Zhang presented their work on HBench:JGC, which is a vector-based benchmarking suite that characterizes application memory-usage patterns and the GC implementation independently. It then combines both metrics to evaluate the performance of a GC in the benchmarked application. The experiments show that HBench:JGC can predict the actual GC time within 10% for most applications.

Distributed Garbage Collection for Wide Area Replicated Memory

Alfonso Sánchez, Luis Veiga, and Paulo Ferreira, INESC/IST, Portugal

Alfonso Sánchez presented their work on distributed garbage collection (DGC) for

wide area replicated memory (WARM). He first identified the deficiencies of two traditional DGC schemes. DGC that uses the message-passing model fails to consider the existence of replicated objects. Conversely, DGC that uses the shared-memory model does not scale well, since it depends on sending causal information using the underlying communication channels.

The authors propose a DGC algorithm that adapts the classical reference counting and improves it to take into consideration replications.

For more information, contact *alfonso.sanchez@inesc.pt*.

Multi-Dispatch in the Java Virtual Machine: Design and Implementation

Christopher Dutchyn, Paul Lu, Duane Szafron, and Steve Bromling, University of Alberta, Canada; Wade Holst, University of Western Ontario, Canada

Christopher Dutchyn presented this paper on extending the JVM to support multi-dispatch. Mainstream OO languages like C++ and Java only support uni-dispatch, such as function overloading and virtual functions. Under these programming environments, programmers often must implement double-dispatch—which inspects the type or arguments of the event object—to decide how an event should be dispatched to the proper event handler. Supporting multi-dispatch greatly simplifies the structure of OO application programs.

Dutchyn's work extends the JVM support for multi-dispatch without changing the language definition. His approach also (1) maintains backward compatibility to source code and library and (2) isolates the semantic changes and performance penalty incurred by the extension within the places where it is used. Several other multi-dispatch designs are compared and contrasted with their design. Empirical results show that supporting multi-dispatch through JVM extension has lower latency compared to functionally equivalent handcrafted code.

For more information, contact *dutchyn@cs.ualberta.ca*.

Using Accessory Functions to Generalize Dynamic Dispatch in Single-Dispatch Object-Oriented Languages

David Wonnacott, Haverford College

David Wonnacott presented another paper discussing the applicability of dynamic dispatching using C++ as the programming language. He explained that the single-dispatching strategy used by most OO languages unnecessarily restricts the extensibility of programs. Specifically, the dynamic-dispatch mechanism (virtual functions) that C++ supports inhibits code reuse through inheritance. The Accessory Functions proposed in the paper allow dynamic-dispatching a message to a group of functions with similar signature but one *virtual* argument. This virtual argument is used to determine which function in the group to dynamically dispatch the invocation to, whereas current OO languages use the receiver of the message to determine how the message should be dispatched.

Accessory Functions work with a group of classes without requiring the implementation details of those classes or violating their encapsulation. The cost of dispatching to accessory functions should be no more expensive than existing language constructs for dynamic-dispatch. Like C++, the system must be able to produce errors related to dispatching prior to program execution. The dispatcher extension should dispatch the message based on the dynamic-dispatch semantics existing in the language.

The alternative dynamic-dispatch mechanism decouples the dispatch method from the class membership. This decoupling lets programmers achieve reuse both by inheritance and reuse in a function.

For more information, contact *davew@cs.haverford.edu*.

Guest Lecture

Extreme Programming: A Lightweight Process

Robert Martin, Object Mentor, Inc.

Robert Martin gave a speech on a popular topic – extreme programming (XP) – and on applying XP in software projects. Ken Arnold defines XP as "a lightweight methodology for small- to medium-sized teams developing software in the face of vague or rapidly changing requirements." XP promotes writing test before code, pair programming, collective code ownership, frequent integration of code base, clean and simple coding style, and frequent communication with customers. Adapting XP practice into your environment improves adaptability, predictability, number of options, and humanity in the development process.

Martin explained that the basic principles of XP are:

- *Rapid feedback* – This facilitates a fast learning environment. You should get immediate feedback for schedule, quality, process, and morale.
- *Assume simplicity* – Solve the problem at hand. Treat every problem as if it can be solved with ridiculous simplicity.
- *Incremental change* – Big changes make things break all at once. Thus, problems should be solved via a series of small changes.
- *Embracing change* – The best strategy for dealing with changes is the one that preserves the most options while solving the most pressing problem. See volatility of requirements as an opportunity, not as a problem.
- *Quality work:* – Everybody likes doing a good job. The only acceptable quality levels are excellent and insanely excellent.

Finally, Martin presented some vivid examples through the perspectives of managers, software developers, and customers to demonstrate how software teams can adapt XP practice.

For more information, see *http://www.objectmentor.com*.

INVITED TALK

Running Through the Woods — A Story About Software Engineering

Bjorn Freeman-Benson, QuickSilver Technology

Dr. Bjorn Freeman-Benson talked about software engineering lessons he had learned from his involvement in several successful large software systems. He made an interesting analogy between software engineering and orienteering. Orienteering is a sport of navigation in which, using map and compass to select routes to run through a series of points shown on the map, one tries to reach the destination in the shortest time. Both software engineering and orienteering have four basic variables:

- *Time* – both want to reach the goal in the shortest amount of time.
- *Cost* – both want to deploy competent tools with minimal cost, e.g., quality of map vs. quality of software engineering tools.
- *Features* – both should avoid using tools with over-complicated features that may adversely affect the progress.
- *Quality* – both care much about the quality of the progress in reaching their goals.

After a brief introduction to orienteering, Dr. Freeman-Benson went on to review four projects in which he had been involved. These projects included Visual Age of OTI, Rose+ of Rational, Amazon's front-end software, and Q compiler of QuickSilver Technology. All rather successfully met their goals as originally set, but in retrospect, he could see things that could have been done differently to make the projects optimally successful.

In conclusion, Dr. Freeman-Benson pointed out that, as with orienteering, in software engineering, you can't build the right thing if you don't know the end goal, if you don't have a plan for how to get there, and if you're sloppy about building it. As the business climate has changed, it has become even more important to build something right instead of just building it quickly.

Session: Reflection in Distribution

The Design and Performance of Meta-Programming Mechanisms for Object Request Broker Middleware

Nanbor Wang and Kirthika Parameswaran, Washington University, St. Louis; Douglas Schmidt and Ossama Othman, University of California, Irvine

Distributed Object Computing (DOC) middleware frameworks, such as CORBA, have gained much popularity in recent years because they shield developers from many complex issues in network programming. There are many meta-programming mechanisms available in CORBA that can further shield the developers from other accidental complexities and improve the adaptability of DOC systems and applications. Kirthika Parameswaran presented a comparison between several of these mechanisms.

Meta-programming mechanisms improve application adaptability by abstracting out certain behaviors into replaceable meta-objects. By using meta-objects implementing different strategies and/or behavior, developers can modify different aspects of a system in a non-intrusive way without losing the maintainability of the application. In this paper, the authors show how the meta-programming mechanisms can be applied at different levels in the ORB invocation path and the challenges of implementing them. They also contrast and compare the scope and implications of applying these mechanisms into applications and provide some guidelines on how to select the proper meta-programming mechanisms.

For more information, contact *kirthika@cs.wustl.edu*.

Kava – Using Bytecode Rewriting to Add Behavioral Reflection to Java

Ian Welch and Robert J. Stroud, University of Newcastle upon Tyne, United Kingdom

When reusing code from outside sources, it is often necessary to modify the original implementations to adapt them to new needs. In this paper, Ian Welch presented another approach to use behavioral reflection to modify the existing programs by rewriting the Java bytecode at code loading time. The behavioral reflection added by the code rewriting associates each object with a meta-object that provides operations like `beforeExecution`, `afterExecution`, `beforeInvoke`, `afterInvoke`, `beforeException`, and `afterException` that developers can use to reflect into the associated operations and object. These operations in meta-objects are invoked automatically through the rewritten bytecode. Developers can use these operations to modify the targeted operation in the original code transparently.

Work on Kava revealed insights on the following topics:

- *Strong encapsulation* – It is difficult to bypass the meta-object bound to the base-object.
- *Inherited methods* – Operations redefined in derived classes will not work with the parent class' meta-object.
- *Exception handling* – Exceptions can be intercepted by the associated meta-object. A context object is used by the meta-object to simplify the meta-object protocol and to allow lazy reification.

For more information, see *http://www.cs.ncl.ac.uk/research/dependability/reflection/*.

Content-Based Publish/Subscribe with Structural Reflection

Patrick Th. Eugster and Rachid Guerraoui, Swiss Federal Institute of Technology, Switzerland

Patrick Th. Eugster presented their work on content-based publish/subscribe implementation. The classic topic-based event service groups messages into topics that are relatively static. Several other publish/subscribe services allow grouping the messages based on the contents they carry. The content-based publish/subscribe service described in this paper utilizes Java's reflection mechanism to acquire the type information of a message. Condition objects are used as filters to define the subscription patterns.

The paper also discusses several performance optimizations, such as avoiding redundant invocations and enforcing static filters. Benchmark results show several metrics, e.g., the number of methods supported by messages, which affect the performance of this approach. These overheads are incurred by the use of Java reflection. They are working on a new optimization that will generate code for static filters based on the execution results from dynamic filters.

For more information, contact *Patrick.Eugster@epfl.ch*.

GUEST LECTURE

Language Integration in the Common Language Runtime

Jennifer Hamilton, Microsoft Corp.

.NET is Microsoft's next-generation development platform that provides easy integration within and across languages and platforms. The Common Language Runtime (CLR) is the fundamental building block of the .NET framework. Jennifer Hamilton gave an overview on the design of CLR and how CLR helps support inter-language object sharing. The major features in CLR include:

- *Common Type System* (CTS) –The single-type system CLR supports. CTS is designed to support multiple languages that .NET implements.
- *Metadata* – Used to used to describe and reference the types defined by CTS. The format of metadata is language independent. Using metadata allows applications written in different languages, programming tools, e.g., compilers and debuggers, and virtual runtime to inter-operate. All objects are strongly typed through the metadata.
- *The Common Language Specification* (CLS) – An agreement between language designers and framework designers. As CTS is too broad for most languages to implement, CLS defines a subset of CTS to guarantee language integration. Exception handling is part of the CLS.

Microsoft Intermediate Language (MSIL) specification defines a type-neutral language that other supporting languages can be translated into, similar to Java bytecode. A unit of object deployment is called an *assembly* and contains self-describing metadata that record the modules and files it contains and the dependencies of containing modules with external modules. The execution model uses a "Class Loader" to link in an assembly.

For more information, see *http://www.microsoft.com/net/*.

Session: Programming Techniques

PSTL – A C++ Persistent Standard Template Library

Thomas Gschwind, Technische Universität Wien, Austria

Thomas Gschwind showed that it is impossible to use a Standard Template Library (STL) container to access persistent data simply by implementing an allocator class because of certain inherent constraints, such as the inability to name permanent data storage through a container interface or query an existing object with the allocator interface. Persistent containers are useful for managing and accessing databases. It is necessary to couple the container interface more tightly with the allocator interface to support persistent containers.

Gschwind described a design that added serialization, deserialization, referencing, and locking capabilities into his Persistent STL (PSTL) containers. He also examined the PSTL compatibility with the existing STL containers and compared the performance of PSTL containers with Berkeley DB and `gmdb`. He found that with PSTL, it is possible to replace regular STL with PSTL with minimal modification.

For more information, contact *tom@infosys.tuwien.ac.at*.

Making Java Applications Mobile or Persistent

Sara Bouchenak, SIRAC Laboratory, France

Sara Bouchenak pointed out that while Java provides code and object mobility and persistence, Java does not provide any support for mobility and persistence for control flows and execution states. Her work in thread migration provides a framework to stop the execution states of a thread so the thread can be migrated to another machine, or checkpointed on disk for later recovery.

She described the challenges in designing a thread-state capture/restoration service and its implementation. The performance of the service was also documented and showed it to be rather competitive. The paper also discusses several application examples on thread migration.

For more information, contact *Sara.Bouchenak@inria.fr*.

BEAN MARKUP LANGUAGE: A COMPOSITION LANGUAGE FOR JAVABEANS COMPONENTS

Sanjiva Weerawarana, Francisco Curbera, Matthew J. Duftler, David A. Epstein, and Joseph Kesselman, IBM T.J. Watson Research Center

Being able to compose components together dynamically is a vital part of applying component technology. Most programming languages, however, do not treat components as first-class citizens and, therefore, do not provide enough support for component development. Francisco Curbera presented the Bean Markup Language (BML), which supports component composition in a first-class manner for JavaBean components.

BML is an XML-based descriptive language used to describe inter-component binding, construct aggregates of components, and allow macro expansion for constructing certain types of recursive compositions. Curbera's BML implementation includes support for scripting event adapters to bridge components together. They have implemented a framework to compose component aggregates based on a given BML description. A GUI front end is also available to support visual composition of components.

For more information, contact *curbera@us.ibm.com*.

DESIGN PATTERNS FOR GENERIC PROGRAMMING IN C++

Alexandre Duret-Lutz, Thierry Géraud, and Akim Demaille, EPITA Research and Development Laboratory, France

Thierry Géraud presented their work on applying some Gand of Four patterns in generic programming. In generic programming, higher efficiency is achieved by the use of parameterized classes. This paper presents the following patterns: Generic Bridge, Generic Iterator, Generic Abstract Factory, Generic Template Method, Generic Decorator, and Generic Visitor.

For more information, contact *Thierry.Geraud@lrde.epita.fr*.