JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2004**

Vol. 3, No. 7, July-August 2004

Using Active Objects for Structuring Service Oriented Architectures

Anthropomorphic Programming with Actors

Dave Thomas, Bedarra Corp., Carleton University and University of Queensland
Brian Barry, Bedarra Research Labs.

1 SOA – SOMETHING OLD SOMETHING NEW

Service Oriented Architectures (SOAs), which were previously beneficial in legacy OLTP and Telecom systems, are once again popular, this time for use with web services. SOAs offer language and technology independence, including the important ability to not require every useful program to be in the latest language and/or on the latest platform. SOAs use self described wire formats such as SOAP, which make it easy to communicate between different technologies.

Large grained encapsulation facilitates simpler interface description and use, while avoiding the pitfalls of frameworks. Most importantly, SOAs provide dynamic binding, which supports flexible and even dynamic service provisioning.

2 CHALLENGES IN STRUCTURING SERVICE ORIENTED ARCHITECTURES

There has been much effort in the so-called choreography/orchestration of web services, primarily with special languages and/or visual tools to connect services. However, there has been little work done in the area of structuring complex web services. At this time, workflow languages and coordination protocols are very new and there is little explicit modeling or structuring support for constructing workflow applications.

In business, threats and opportunities result in new models for doing business and these in turn result in new roles, responsibilities and processes. Increasingly these new models required collaborative organizations which many hope can be realized through a technology such as web services. Of particular interest to us is the ability to model a virtual business and populate that virtual business with both human, human augmented,

Cite this column as follows: Dave Thomas, Brian Barry: “Using Active Objects for Structuring Service Oriented Architectures”, in *Journal of Object Technology*, vol. 3, no. 7, July-August 2004, pp. 7-14. http://www.jot.fm/issues/issue_2004_07/column1

and fully mechanized activities. Unfortunately, current software tools provide little high level support for this style of modeling and execution. In particular they maintain and/or increase the gap between business designer and software designer.

MDA and classic Workflow tools support complex state machine descriptions that are often far too complex for composable workflow. State machines and petri nets are difficult to adapt to the needs of dynamically changing business. In particular it is very difficult to cope with the exception handling which is commonplace in asynchronous office systems. They also impose an additional runtime complexity, embedding their own concurrency semantics for states, flows etc. which are opaque to the business analyst.

Web Services use many specialized languages including BPEL4WS, WSDL, XML, and SOAP, each of which is itself focused on one aspect of the problem. While such an approach definitely has its merits, it creates a lot of additional accidental complexity at the seams. It greatly complicates design and implementation choices leading to increased complexity for service builders.

Finally current modern programming languages, for example, are limited to passive objects and associated low-level concurrency machinery. The designer is forced to manually build active objects using low-level thread/process and synchronization primitives. This leads to greater accidental complexity and hence potential performance and concurrency problems. Many of the examples cited by the AOSD community address the complexity of weaving concurrency, transaction, persistence and security aspects in modern application servers.

3 ACTORS – AN ANTHROPOMORPHIC APPROACH TO WORKFLOW

In the early 80s, we worked in the office automation and business process automation space. The work at the time was focused on data flow techniques for transforming paper flow to automated workflow. Those early days in office automation (OA) saw the first application of state machines and petrinets to specify office semantics [Zisman, Ellis 13]. These efforts quickly ran into the classic problems of ill-defined office semantics and exception handling. Further, they lacked modularity as is often experienced with large state machine descriptions.

Smalltalk, the Xerox Star, Programming By Example [14, 16] and Actors [1, 15] provided powerful new metaphors for building complex business systems. Actors in particular provided an extensible active object architect for designing business processes, associated workflows and behavior.

We feel now, as we did then, that Actors provide a simple and elegant language mechanism for SOA applications. Specifically they provide a family of anthropomorphic active classes which meet both the needs of the business designer and those of the software implementer.

Actor Languages

Carl Hewitt (MIT) and others (principally Henry Baker and Gul Agha) developed actor languages. Hewitt's actors are autonomous and concurrent objects that communicate asynchronously and are intended to be a model of an intelligent person.

When an actor receives a message, it executes according to its script and communicates with a well- defined and finite set of other known actors. Actor languages are clean, simple, elegant and powerful.

Actors unify synchronization, message passing and encapsulation in much the same way as a monitor unifies procedure calls and synchronization. The actor allows one to model computations as an organization of communicating active objects and to apply anthropomorphic roles such as workers, coordinators, managers, couriers, notifiers etc. This allows business processes to be expressed using common organizational design principles.

Actor Programming Model

The Actor programming model is to begin with a simulation or animation of the whole system and then build it out, i.e. build an executable model. As the system grows, it takes on characteristics of a solution. Normally this model followed a variant of the interactive Spiral Model that underlies Agile Development approaches. First get the functional requirements right and then focus on the performance.

4 THE ACTRA PROJECT

The Actra Project [9] was a joint research project at Carleton University and OTI for the Canadian Defense Research Establishment during the years 1985 through 1990. The project applied advanced OO and iterative development to the design and implementation of a complex embedded command and control application [10, 11]. Actra sought to show how far Smalltalk could be used in the development of complex embedded applications. The principal research results include: Orwell (aka ENVY/Developer), a team development environment; the Actra actor model and its integration into Smalltalk; a multiprocessor virtual machine implementation – GC in particular; high performance object serialization; and application development using active objects.

Actra combined Smalltalk, Actors and Multiprocessors and used the Harmony multiprocessor operating system as a foundation. An Actor encapsulates cooperating passive (non-Actor) objects. Actors synchronize and communicate by sending messages. Actra ran successfully in both SMP and networked environments, but in 1990 it stretched the state of the art in operating systems and hardware to its limits.

Message Based OS Kernels

Thoth was the archetype operating system that was developed at the University of Waterloo in the late 1970's. It has many descendants including Port, V Kernel, Harmony, and QNX.

Harmony offers a very robust and stable implementation. Harmony has very lightweight tasks (tasks = processes = threads), and a common interface for local and remote tasks.

It has a portable real-time multitasking multiprocessing kernel and is based on simple primitives: Blocking Send, Blocking Receive, Reply, Create, Terminate and special forms for Non-Blocking Receive and Interrupts. It provided standard system services such as ClockServer, DirectoryServer, and LogServer.

Harmony is small and was essentially written by one very smart person. It was an elegant solution with a simple synchronous communication model and a stable performance implementation which was easy to understand and easy to get right.

Process Structuring in Harmony

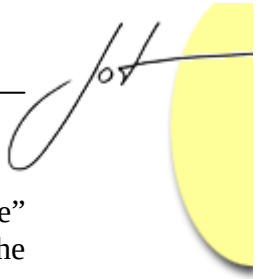
Process Structuring in Harmony includes Anthropomorphic Programming. Tasks are assigned personified roles such as Servers, Administrators, Workers, Couriers, Notifiers, etc. Each of these roles has well known pre-defined semantics.

The process structuring (or architecture in today's language) in Harmony is such that tasks are assigned to processors. Tasks then communicate synchronously, with non-blocking reply. Servers must be responsive, so they delegate most of the work. Processes spend most of their life in a "receive any" loop, while Workers do most computations. The Administrator helps organize this.

Actra Actors

Actra provides a new object class called Actor. This replaces the Smalltalk co-routine process model. Actors execute in pseudo parallel on a single processor and in parallel on multiple processors. Actors have the granularity of lightweight processes (threads/tasks). There are uniform semantics for remote/local processes and these processes have a well-defined life cycle.

The Actra Message Model is designed to allow a serial Smalltalk application to be naturally evolved into a concurrent application. Actors are invoked from another actor via message send which causes an implicit OS blocking ***send.*** Method return executes an implicit OS unblocking ***reply***, releasing the sender. A message is accepted when a target actor is receive-blocked state. Explicit ***send***, ***receive***, ***reply*** are also supported. There is no ***receive_specific*** call (as in Harmony), only ***receive_any***. The resulting coding style is very natural for Smalltalk programmers. Switching between active/passive roles didn't affect most of the code.



Process structuring is with Actors, which execute concurrently and are “large scale” objects with personified roles. The Harmony message-passing model was used and the same generic actors are used: Servers, Workers, Notifiers, Couriers, and Administrators. Programmers create their own application specific actors by specializing the generic ones. The complete taxonomy of known Actors, some generic, many more application specific, creates a vocabulary that populates the programming model and defines its semantics.

5 DESIGNING ACTOR BASED SYSTEMS

Hewitt defined a design process for his actor languages:

1. Decide what the actors are
2. Determine their message protocols
3. Define their behavior

In Actra, we add structure by providing a taxonomy for Actors and introducing OO inheritance. Working Groups of collaborating Actors are defined with only a few Actors in each Working Group. Actors know about Superiors, Subordinates, and Colleagues (who share a common supervisor).

To help with modeling and mapping concurrency, semantics for message passing are similar for actors and objects. Consequently decisions made about concurrent behavior are easily changed. There are uniform semantics for local and remote Actors, so Processor/Actor assignment becomes a runtime optimization. Our rule of thumb is, when in doubt, assume a component is an actor. It will usually become obvious when the assumption is wrong.

Actra has well-defined life cycle policies including initialization (e.g. order of object creation) and finalization (e.g. handling child Actors). There is a standard activation sequence and configurable serialization. Policies can be refined in subclasses. Delegation is used to share responsibility, to task subordinates and to refer to supervisor.

6 ACTOR TAXONOMY

First class active objects are components that encapsulate a set of state and behavior together with a thread of control. Simple inheritance was used because it is an easy model to understand and use. Mixins and multiple inheritance, etc. are more powerful but add complexity (always choose the simplest powerful solution).

Generic Actors include:

From Anthropomorphic Programming - Clients, Servers, Agents, Managers, Secretaries, Couriers, Workers, Notifiers ...

- Workers: send to servers for work, perform computation

- Notifier: event handling Worker
- Courier: messenger Worker, used for delegation and communication
- Server: provides services
- Proprietor: manages resources, mitigates access
- Administrator: manages worker pool
- Dispatcher: provides asynchronous communication

Transactor: adds ACID properties to computation for coordinating distributed transactions across multiple services.

Business Processes = Workflow + Rules + Control (e.g.. Taylor engines)

Agents = Actor where methods are rules

Avitar = Actor where method is script and displayOn: uses VRML

7 CONCLUSION

Conventional SOAs are based on anonymous services. Thousands of anonymous services don't help – we're back to the 10-foot shelf of API manuals. There is a need to group, categorize, organize, and manage services. In our view, the use of actors is a powerful and natural model for describing such systems.

The anthropomorphic approach seems to be very natural. Actor taxonomy is a dense encoding of knowledge. It populates the programming model, defines its semantics, it is easy to customize/extend the model and easy to remember and use. Concurrency and collaboration are built-in and there is an incremental, interactive, navigational programming style. Our motto is: Choose a uniform deep model and then learn to live within it. Code and debug at the level of the abstraction. For example, the send graph allows one to reason about the concurrency of the actor application, showing that simple natural models may also lead to more tractable reasoning about correctness.

Strong models and deep domain abstractions means very dense code, which we believe is a **good** property – dense code means less code, hence greater productivity for skilled knowledge workers.

REFERENCES

1. Hewitt's Message Passing SemanticsGroup: Actors Foundations for Open System, http://www.erights.org/history/actors.html
2. David R. Cheriton, Michael A. Malcolm, Lawrence S. Melen, Gary R. Sager: "Thoth, a portable real-time operating system", *ACM CACM* Vol. 22, 2, pages 105-115, 1979.
3. W. Morven Gentleman: "Message Passing Between Sequential Processes: the Reply Primitive and the Administrator Concept. Softw.", *Pract. Exper.* 11(5): 435-466 (1981)

4. Stephen A. MacKay, W. Morven Gentleman, D. A. Stewart: “Harmony as an object-oriented operating system”. *SIGPLAN Notices* 24(4): 209-211 (1989)

5. David R. Cheriton: "The V Kernel: a software base for distributed systems”. *IEEE Software*, 1(2), pp.19-42.

6. David R. Cheriton: “The V Distributed System”. Computer Science Department, Stanford University, March 1987.

7. David R. Cheriton: “The V Kernel: a software base for distributed systems”. *IEEE Software*, 1(2), pp.19-42.

8. QNX Software Systems, QNX Neutrino RTOS is a true microkernel operating system, http://www.qnx.com/

9. David A.Thomas, Wilf R. Lalonde, John Duimovich, Michael Wilson, Jeff McAffer, and Brian M. Barry, “Actra- A Multitasking/Multiprocessing Smalltalk”, *Proceedings of the ACM SIGPLAN Workshop on Object-Based Concurrent Programming*, San Diego, October 1988. Published as ACM SIGPLAN Notices, Vol. 24, No 4, 1989.

10. Brian M. Barry, D.A.Thomas, J.R. Altoft, and M. Wilson, “Using Objects to Design and Build Radar ESM Systems”, *Proceedings of OOPSLA'87*, ACM SIGPLAN, Orlando, October 1987.

11. Brian M. Barry, “Prototyping a Real-Time Embedded System in Smalltalk”, *Proceedings of OOPSLA'89*, ACM SIGPLAN, New Orleans, October 1989.

12. Brian M. Barry, Active Object Programming Models, 2004.

13. http://www.workflow-research.de/Research/

14. de Jong, S.P., and Zloof, M.M., “The system for business automation (SBA): programming language”, *Communications of the ACM*, Volume 20, Issue 6 (June 1977), Pages: 385 - 396

15. Roy J. Byrd , Stephen E. Smith , S. Peter deJong, “An actor-based programming system”, *ACM SIGOA Newsletter*, v.3 n.1-2, p.67-78, June 21-23, 1982

16. Giuseppe Attardi , Maria Simi, “Extending the power of programming by examples”, *Proceedings of the SIGOA conference on Office information systems*, p.52-66, June 21-23, 1982

About the authors

Dave Thomas is CEO of Bedarra Corp., Adjunct Professor at Carleton University, Canada and University of Queensland, Australia, founding Director of AgileAlliance.com, and founder of Object Technology International. Bedarra works with research labs and commercial partners to transition innovations into products and practices.

Brian Barry is currently CEO of Bedarra Research Labs. From 1991-2002 he served variously as Chief Scientist, CEO, President and CTO at Object Technology International, Inc. Under his leadership OTI developed the Eclipse IDE Platform, IBM VisualAge for Java, and IBM VisualAge MicroEdition for embedded systems. Brian has over 20 years of experience in the design and implementation of object-oriented and component-based systems, including distributed, client/server, embedded and real-time applications. He was a charter member of the ANSI Smalltalk committee and a co-author of the Smalltalk standard. He holds a Ph.D. from Queen's University.