OBTAINING ADA AND UML TO HELP DEVELOP MULTIAGENT SYSTEM SOLUTIONS

Bruce R. Barkstrom

15 Josie Lane
Asheville, NC 28804

1. INTRODUCTION

This chapter provides information on using Ada and Unified Modeling Language (UML) to explore the algorithms presented in Shoham and Brown's text on multiagent systems [2009]. Such systems are interesting because they are easy to visualize as collections of people who converse with each other to solve problems or to develop social consensus. They are characteristic of many areas of social science, including economics [Kendrick, Mercado, and Amman, 2006; Hendrickson, Lave, and Matthews, 2006], standardization of information networks [Weitzel, 2004], and social learning [Chamley, 2004]. In terms of modern IT approaches, this means that multiagent systems appear to fall cleanly into object-oriented design techniques that can easily use UML. Even more to the point, UML can describe the communication patterns between the agents using synchronization diagrams. UML also supports using state diagrams for the individual agents to ensure that the agents respond properly to the communications that trigger changes in their internal states.

The notion that agents are individual active objects that communicate with each other also fits well with the concepts embedded in Ada, particularly with the revisions in 1995 and 2005 that move beyond the simple notions in the original 1983 standard. The Ada structure that is equivalent to an agent is the *task*. In most current Ada implementations, a task becomes equivalent to a thread of control. Communication between threads is standardized in Ada using the notions of *rendezvous* and *protected objects*. Thus, the language does not need additional support from the operating system or special libraries in order to deal with agent-to-agent communication. Because of these features, Ada appears well-placed for designing and implementing efficient multiagent systems even in multi-CPU, multi-core computational architectures.

In the remainder of this chapter, we describe how interested users can obtain an Ada development environment and a UML tool that will support the design and testing of multiagent systems. Our suggestions on these elements are intended to allow individuals to obtain the necessary tools free of charge for their own work or for work in an academic environment. Professionals in multiagent design and development (or in embedded systems work) can substantially extend their capabilities – albeit at a significant cost.

2. OBTAINING ADA FOR PERSONAL USE

Ever since Ada's inception, there has been a strong emphasis on teaching the language. The language itself has always had an International Standard Organization (ISO) document known as the *Ada Reference Manual* (RM) [Ada-Europe, 2007]. This document is rather large (and perhaps formiddable). At this online site, the RM is available both as a PDF and as an HTML document. In addition to being available online, the RM is usually attached to Ada development tools as a library of HTML pages. The RM contains a smattering of useful examples.

In addition, there are a number of Ada textbooks. Barnes [1998] is typical. The concepts needed for dealing with multiagent objects fall into the advanced parts of the language. Probably the best general text for such uses is Cohen [1996]. This text is comprehensive, even on the advanced features of the language. It has many helpful examples in the text. Additional help is available on the features of particular interest for multiagent programming in Burns and Wellings [1998]. These two texts probably provide sufficient guidance for interested individuals to gain experience in concurrent programming in Ada. As background, Klein, et al. [1993] is also quite

helpful, although it is a relatively early text. This book provides a "programmed text" dealing with elements of scheduling theory that are appropriate to concurrent systems. It provides a quite understandable introduction to such subjects as the difference between "hard" schedules (in which the system fails if it misses completion of a task) and "soft" ones. "Hard" schedules appear in embedded systems (such as those that work with weapons or health care equipment). Fortunately, multiagent simulations do not appear to have requirements for this kind of scheduling. Conccurrent programming is hard enough without those constraints.

The GNAT Ada compiler and development environment is available online [Adacore, 2009], under a GNU General Public License [GNU, 2009b]. This version is intended for Open Source software and is supported in the typical Open Source manner. Adacore, an Ada tool company also provides an academic version and a fully supported professional version. Adacore also supports the GNU Compiler Collection [GNU, 2009a]. Other Ada tools are also available, some with academic or free versions.

All Ada code for this text has been developed and tested using the GNAT GPL compiler. This code is freely available under the GNU GPL, which is a "copyleft" license. Organizations interested in developing commercial products should contact the author.

3. OBTAINING A UML TOOL FOR PERSONAL USE

Unified Modeling Language (UML) is a useful tool for describing the design of object-oriented (or agent-based) software. Wikipedia [2009] provides a useful general history of UML, although the article referenced still appears to need some editing. UML started development in the early to mid-1990s. As developers discovered its utility, the language "accreted" and became UML 2.0. The Object Management Group (OMG) adopted this version as the standard in 2005 [see OMG, 2009 for the current version]. Addison-Wesley has a series of books on UML and the related software development methodology known as the "Rational Unified Process," many written by noted development process gurus, Grady Booch, James Rumabugh, and Ivar Jacobson [1999].

Our intent here is not to pursue a full-blown design methodology, but to emphasize the UML diagrams and tools that appear most useful in designing programming examples that provide working implementations of the algorithms in Shoham and Leyton-Brown [2009]. The author has found that Bruce Powell Douglass' books have been very useful for this purpose [Douglass, 1998; Douglass, 1999]. Readers who are new to UML may want to get the first edition of Douglass' *Real-Time UML* as it provides a fairly readable approach to moving from a project's context to a finished design for an agent-based system.

As we will see in the algorithm solutions, it is often feasible to strip down the design of this kind of systems into three steps:

1. Visualize the agents involved and document their properties as UML classes
2. Visualize the communication patterns between the objects as use cases
3. Use state diagrams or state charts to provide a clear understanding of the relationship between an agent's state, the actions of that agent when it is in a given state, and the communications between objects that lead to state changes

The author would also admit to jumping into programming in order to clarify what is going on. Having a reasonably good, interactive debugging tool (such as the one available in the GNAT development environment) is critical in such "interactive design" work. While it is probably true that training in a good design methodology is useful, experience usually comes from the accumulation of mistakes – the corrections of an enthusiastic intuition.

In the end, UML is useful for getting a handle on how agents need to interact and for explaining a design to other people. Based on the author's experience, the BOUML tool (available from [BOUML, 2009]) is quite good. This tool covers most of the UML 2.0 diagrams and is still under active improvement. It comes with an HTML tutorial that is available from its toolbar.

It should be noted that when large projects are developing software, UML may be used to go from design to code. BOUML is not the only tool available for this purpose. IBM's Rational tools

even come with an Ada code writer. However, for the exploratoration of the multiagent algorithms in Shoham and Leyton-Brown (or other textbook exercises), such automation is not really necessary.

Disclaimer: The author's opinions regarding tools or texts that are expressed in this chapter are based on his personal experience with them. He does not have any proprietary interest in these tools and texts and receives no income from the opinions expressed here.

4. AFTER YOU GET THE TOOLS

The set of problems in this text is not intended to teach you how to program (in Ada, even) or how to make UML diagrams. For either of these kinds of texts, you need the documentation that comes with the tools and, probably, one or more of the references in the appropriate subjects.

For the last few years, the author has worked with both the GNAT GPL Ada development environment and the BOUML tool, using Windows NT. Based on that experience, both of these tools are easy to deploy there. Previously, the author had used Ada in a Linux environment. I would anticipate a similar ease – but I haven't deployed the tools into that environment, so my advice is probably to be taken with a grain of salt in this area.

At any rate, getting these tools from the Web should be pretty straightforward. Assuming you're familiar with your operating system, installing them should present you with no difficulties either. Thereafter, it's a matter of getting familiar with the tools.

If you haven't programmed before, you need to find a good programming tutorial and learn to use the languages. Try to learn the tools separately. It's probably best to learn Ada first and then to learn to UML second.

For Ada, you'll want to be able to use the basic data structures (including arrays and records) as well as the basic control structures (like loops and it-then-else constructions). You'll also want to be able to create, open, write, read, and close various kinds of files – noting that the current instantiation of Ada has four basic types of files it can deal with. Text_IO is basic. Sequential and Direct variants of IO are also useful, particularly to simulate database work. Stream_IO is not necessary for most of the examples we work.

For UML, you can follow the tutorial that comes with BOUML. It's available from the Help menu on the usual toolbar (in Windows). It's also helpful to follow through the tutorial, creating exercises as you go. Learning UML, on the other hand, can be a non-trivial exercise in minute distinctions. Many of the texts on UML (particularly those in the Addison-Wesley series of books on the subject, including B. P. Douglass') are devoted to filling in details regarding ways of designing rather abstract objects. At least for the initial work we're doing on multiagent systems, the objects are going to be pretty clear: they're the active players in the games or simulations we describe. That means that you can operate at a pretty elementary level in defining objects. The more important skills relate to designing the communication strategy between the agents and designing the sequence of states each agent goes through.

After you have learned these precursor skills, you can pick up on what this set of examples is intended to teach. As you'll see, once you've settled on the agents (or active objects) in a particular problem, you need to figure out how these agents need to communicate in order to get their work done. Sometimes you'll be able to think of the communication pattern as a "query-response" pattern, i.e., client-server or synchronous exchanges of information. At other times, you'll need a peer-to-peer pattern, which may be asychronous. There are even cases that may be dealt with by "broadcast" messages to many other agents.

It is clear that concurrent programming is much more difficult than strictly serial programming. Agents in a multiagent system are not necessiarily polite about interrupting the flow of computation in other agents. The real design work lies in taming these impolite communication patterns without unduly hindering the overall progress on the total problem being solved. Hopefully, by progressing through the examples in this text, you'll be able to gain some wisdom in how to do this well.

5. REFERENCES

Ada-Europe, 2007: *Ada Reference Manual*, **ISO/IEC 8652:2007(E) Ed. 3**. Available at `http://www.ada-auth.org/arm.html`. Accessed 10 Apr., 2009.

Adacore, 2009: *GNAT Ada*. Available at `https://libre.adacore.com`. Accessed 10 Apr., 2009.

Barnes, J., 1998: *Programming in Ada 95*, **2nd ed.**, Addison-Wesley, Reading, MA, 702 pp.

BOUML, 2009: *BOUML Free UML 2 Tool Box*. Available at `http://bouml.free.fr`. Accessed 10 Apr., 2009.

Burns, A. and Wellings, A., 1998: *Concurrency in Ada*, **2nd ed.**, Cambridge University Press, Cambridge, UK, 390 pp.

Chamley, C. P., 2004: *Rational Herds: Economic Models of Social Learning*, Cambridge University Press, Cambridge, UK, 402 pp.

Cohen, N. H., 1996: *Ada as a Second Language*, **2nd ed.** McGraw-Hill, New York, NY, 1133 pp.

Douglass, B. P., 1998: *Real-Time UML: Developing Efficient Objects for Embedded Systems*, Addison-Wesley, Reading, MA, 365 pp.

Douglass, B. P., 1999: *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*, Addison-Wesley, Reading, MA, 749 pp.

GNU, 2009a: *GCC, the GNU Compiler Collection*. Available at `http://gcc.gnu.org`. Accessed 10 Apr., 2009.

GNU, 2009b: *GPL, the GNU General Public License*. Available at `http://www.gnu.org/copyleft/gpl.html`. Accessed 10 Apr., 2009.

Hendrickson, C. T., Lave, L. B., and Matthews, H. S., 2006: *Environmental Life Cycle Assessment of Goods and Services: An Input-Output Approach*, Resources for the Future, Washington, D.C., 259 pp.

Jacobson, I., Booch, G., and Rumbaugh, J., 1999: *The Unified Software Development Process*, Addison-Wesley, Boston, MA, 463 pp.

Kendrick, D. A., Mercado, P. R., and Amman, H. M., 2006: *Computational Economics*, Princeton University Press, Princeton, NJ, 436 pp.

Klein, M. H., Ralya, T., Pollak, B., Obenza, R., and Gonzalez Harbour, M., 1993: *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*, Kluwer Academic Publishers, Boston, MA.

OMG, 2009: *UML*. Available at `http://www.uml.org`. Accessed 10 Apr., 2009.

Shoham, Y. and Leyton-Brown, K., 2009: *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*, Cambridge, Cambridge, UK, 483 pp.

Weitzel, T., 2004: *Economics of Standards in Information Networks*, Physica-Verlag, Heidelberg, Germany, 292 pp.

Wikipedia, 2009: *Unified Modeling Language*. Available at `http://en.wikipedia.org/wiki/Unified_Modeling_Language`. Accessed 10 Apr., 2009.