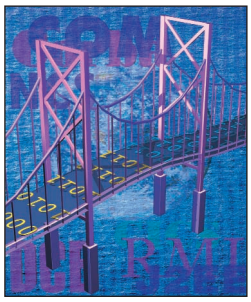

Middleware "Dark Matter"

Steve Vinoski • *IONA Technologies* • *vinoski@ieee.org*

Some astronomers theorize that the universe is filled with "dark matter" that we can't see because it emits too little radiation for our instruments to detect across the vast distances of space. Whether dark matter actually exists, and if so in what quantity, is important to our understanding of the universe. Some theories about what occurred when the universe began are even predicated on dark matter's existence. If it's actually there, its presence is profoundly important to the universe's very existence and operation.

Clay Shirky describes PCs as the "dark matter of the Internet" because a lot of them are connected, but they're barely detectable.[1] We can apply a similar analogy to middleware because the "mass" of the middleware universe is much greater than the systems — such as message-oriented middleware (MOM), enterprise application integration (EAI), and application servers based on Corba or J2EE — that we usually think of when we speak of middleware. We tend to forget or ignore the vast numbers of systems based on other approaches. We can't see them, and we don't talk about them, but they're out there solving real-world integration problems — and profoundly influencing the middleware space. These systems are the dark matter of the middleware universe.

A Dark Matter Scenario

In response to my previous column,[2] reader Norm Katz noted that not all integration efforts revolve around well-known or market-leading technologies or products. As a consultant, Katz regularly works with customers who lack the necessary funds or skills to purchase or use high-cost middleware to solve their integration problems.

He's right, of course. Many integration projects are based on everyday middleware dark matter — languages and systems that work just fine, but are too mundane for those writing for market-oriented technical publications or academic conferences.

Say you're a system administrator in a small company, and your boss informs you that two important departments want their primary systems to talk to each other as soon as possible. The task of integrating them has fallen to you, the company's "jack of all trades (but master of none)." You're not really a programmer, but you have deep knowledge of how to keep networked Unix and Windows systems up and running, and you've written some significant shell scripts and batch files that did the job in the past.

The two systems you're charged with integrating are based on completely different technologies supplied by different vendors. Each system uses its own network protocol, and the two are incompatible with each other. They can almost certainly be bridged using EAI, MOM, Corba, or J2EE middleware, but you're a system administrator, not a distributed systems developer or an EAI expert. Moreover, even if you had experience with such middleware, your boss wouldn't let you use it because of the cost or perceived complexity of applying it to your task.

Several Ways to Skin a Cat

One way to solve this problem is to use the lowliest, but most ubiquitous, middleware dark matter in existence: the humble text file. We can coerce most systems into producing some textual form of the information they process. Thus, by making the text files output by one system available to the other system, you might achieve an integration that's good enough for the two departments involved and, better still, good enough for your boss.

Integration via text files is not only ubiquitous, it's as old as the hills. Unix command-line tools generally read text from their standard input and produce text on their standard output, for example, and Unix shells let you integrate tools via

 1089-7801/02/$17.00 ©2002 IEEE

pipes that feed one command's standard output to the next command's standard input.

This approach does raise the issue of how to send text from one system to another, but straightforward and simple approaches work here as well. The first system could produce its text file in a specified file system directory, for example, and the next system could poll the directory occasionally and act on the file when detected. I've seen this approach applied quite often, but it obviously works only when the systems can share a file system. Alternatively, we could send the text file over the network, using a common protocol such as FTP. The common FTP application is interactive and intended to be human-driven, but tools such as Don Libes's Expect utility[3] can turn just about any interactive application into an automated noninteractive system. In practice, FTP is heavily used for automated system-to-system file transfer for integration.

The other big issue with text files is content format. It is fairly unlikely that a system based on one vendor's technology will produce text files that are directly consumable by another system that uses a different vendor's technology. Thus, we must generally modify one system's text output before it can be input to another system in this type of integration. Even on Unix, for example, you traditionally apply filters based on the sed, awk, or grep utilities to reduce, augment, or modify the text produced by one tool before piping it into another.

One of the most useful tools available for general-purpose scripting, including filtering text files, is Perl (whose full name — the Practical Extraction and Report Language — indicates that it is a full-featured language when it comes to data filtering and integration). A system can use Perl's extensive pattern-matching features to recognize and act on just about any input. Perl also has facilities for reading and writing fixed-format files, and if the built-in features aren't enough, you can easily extend Perl by adding modules to an installation. Perl is well suited to real-world heterogeneous integration tasks because it has been ported to just about every platform in existence.

Together, Perl and Python account for a significant fraction of middleware dark matter. For more on Perl, as well as other tools, such as Python, see the sidebar on "Dark Matter Details" (next page).

Simple, Yet Adequate

Let's say you're accustomed to integrating systems or tools using Perl and text files, and your boss asks you to set up a Web site that allows access to data resulting from one of your integration projects. Your department already has a Web server running, so all you need to do is tie your data into it. A brief search of the Comprehensive Perl Archive Network (www.cpan.org) turns up several common gateway interface (CGI) modules for Perl that let you easily write programs that a Web server can use to respond to browser requests. Given your familiarity with Perl, you can use one of the CGI modules to quickly put together the site your boss is looking for.

I had such an experience several years ago when I used a Perl CGI module to create a Web-based bug-tracking system for a small development team I was leading. The resulting system, which integrated our developers' desktops with our bug-tracking database via a browser-based GUI, was easy to build, maintain, and extend. When the company made our team switch to a vendor-supplied customer relationship management (CRM) system for bug tracking, we found that my 1,000-line Perl system was superior in just about every important way.

CGI's limitations are well known, particularly the fact that a separate operating system process services each HTTP request, which is fairly expensive. Nevertheless, the costs are not prohibitive for many projects. Like text files, CGI often provides a more than adequate solution for simple integration projects. Given that some Perl and other such modules make building CGI applications almost too simple, there's often no need to throw something complex, like J2EE servlets, at the problem.

In addition to CGI modules, the CPAN site has Perl modules for database access, networking, file handling, operating system interfaces, mail, and news. Clearly, these modules collectively supply an extensive and useful middleware layer.

Web services might very well start to illuminate our middleware dark matter.

There are no hard and fast rules to help you decide whether to use traditional middleware or middleware dark matter to solve your integration or distributed computing problems. These decisions depend on your skill set, financial situation, the scope and schedule of your project, and the capabilities of the systems and technologies you're integrating. Middleware dark matter sometimes (but not always) lends itself to solutions that are easy to create but fail to evolve, scale, or perform as well as a traditional middleware solution. On the other hand, using traditional middleware is no guarantee that your system will evolve gracefully, or perform or scale well. Not surprisingly, it comes down to skill and experience. Just as you can write bad code in any programming language, you can develop a poor system using any type of middleware.

The Dark Matter Influence

The situation begs the question of how

Dark Matter Details

The middleware dark matter systems mentioned in this column are used extensively, and each has its own community of users and supporters.

- *Perl.* Larry Wall invented Perl to help him exchange information across a widely distributed and heterogeneous network of computer systems, and to generate reports about each exchange. Perl borrows syntax and semantics from several preexisting tools and appli-cations, including sed, awk, grep, C, Fortran, and the Bourne shell. Starting with the first version — posted in 1987 to the comp.sources Usenet newsgroup — Wall released Perl in source form to allow others to extend its usefulness and help maintain it. That fact has earned him wide recognition as a pioneer of the open source movement. Because of the large number of Web sites that use it to hold their implementations together, Perl is often admired as the "duct tape of the Web." You can find more details about Perl, as well as numerous modules, at www.perl.org.
- *Python.* Guido van Rossum created Python around 1990 to provide a powerful scripting language that could succeed the ABC programming language for the Amoeba distributed operating system. Many view Python as a rival to Perl, and much has been made of the fact that Python's syntax is far cleaner than Perl's. The languages share many similarities and are used to solve many of the same problems, including text filtering, system administration scripting, and Web site implementation. You can learn more about a similar tool called Python at www.python.org.
- *Tcl.* While at the University of California, Berkeley, John Ousterhout invented the Tool Command Language as a general-purpose embeddable interpreter. Tcl was designed to allow developers to use scripts to easily invoke and glue together functions and applications written in C — function-ality that was later extended into the worlds of C++ and Java. Ousterhout also built Tk (so named because it's a toolkit for Tcl), which enabled rapid prototyping of graphical user inter-faces. Tk was originally implemented for X Windows but later ported to Microsoft Windows and other win-dowing systems. Tcl's syntax can be verbose, but it has still been used to develop extensive systems, including a fully functional HTTP daemon. You can learn much more about Tcl at www.scriptics.com.
- *Visual Basic.* Microsoft introduced VB in 1991 to allow developers to visually create Windows applications — particularly those with GUIs, which could be created without writing a single line of code. Indeed, VB's GUI capabilities are probably the biggest reason that Windows applications generally support standard interfaces that let users seamlessly switch between applications. A third-party market for VB add-ons and controls began developing soon after the first release of VB, which in turn helped create the massive VB user community that exists today. Over the years, VB has evolved to provide the same ease of development for database and Web applications as well.

These systems share several characteristics with other middleware dark matter:

- *Extensive community support.* Because a lot of middleware dark matter is open source, it attracts users and developers who contribute to such systems' continued maintenance and evolution.
- *Support for rapid prototyping.* Middleware dark matter is typically based on interpreted languages with flexible type systems that are conducive to rapid edit-run-debug cycles.
- *Highly extensible.* Systems such as Perl, Python, and Tcl allow developers to create modules that build above the language as well as extensions that are embedded into the language interpreter. This flexibility makes it possible for each language's user community to extend it and apply it to an ever-growing variety of problems.
- *Web-related.* Many applications of middleware dark matter involve the Web as many extensions for these systems were built to solve Web site integration and implementation problems. As the Web grew and matured, so did these middleware systems.

Many other systems also classify as middleware dark matter, such as Delphi (www.borland.com/delphi/). I can't detail them all here, but they often have similar histories and share many of the characteristics of the systems I've described.

much, if any, overlap there is between middleware dark matter and traditional middleware. Integration projects tend to evolve and grow over time. Successful dark matter-based systems often reach a point at which administrators can't tune or improve them any further without simply replacing some or all subsystems with more traditional middleware technologies. Do those who solve problems using middleware dark matter ever "cross over" into the realm of traditional middleware?

When you mention J2EE to someone associated with traditional middleware, for example, they normally think of Enterprise JavaBeans, Java Messaging Service, and the Java Connector Architecture – outgrowths of traditional distributed objects, database, MOM, and EAI work. However, I have personally heard from several reliable sources, including respected middleware analysts, that most developers who use J2EE are actually using servlets, Java server pages, and Java database connectivity. Only a small (but growing) number are actually using EJBs. To me, this clearly indicates dark matter's influence.

As system administrator for your small company, you could easily advance to writing systems too large or heavily used for the CGI approach to handle. Given the understanding of CGI and Web applications you would have to possess to reach that point, it's no stretch to imagine that you could turn to J2EE, learn to program in servlets or JSPs, and use those skills to pick up where CGI runs out of steam without ever needing to use EJBs.

Similarly, someone used to achieving integration via text files could use similar tactics to instead use XML, which is, after all, text. Moreover, it's designed specifically to represent structured hierarchical data, filtering and transformation tools are readily available for it, and many of today's systems can already produce it. XML is quickly replacing specialized formats and their associated tools in both the dark matter and traditional middleware communities, as using standard XML and its tools and practices is less expensive than developing and maintaining proprietary tools and formats.

Web services might very well start to illuminate our middleware dark matter. Given that a significant portion of the existing middleware dark matter is probably implemented in Visual Basic, for instance, Microsoft's promise that .Net will put a Web services substrate beneath VB — as well as Perl, Python, and other middleware dark matter languages — is indeed significant. Even as vendors of traditional middleware are moving to augment their Corba, J2EE, MOM, and EAI systems with Web services, the middleware dark matter community is building modules for SOAP, the Web Services Description Language, and other technologies related to Web services (for example, a search of CPAN in August for SOAP turned up 226 hits). Web services might therefore be a genuine point of convergence for traditional middleware and middleware dark matter. If this convergence actually occurs and Web services do manage to illuminate the dark matter, there's no question that the "laws of physics" that currently govern our middleware universe will change.

Acknowledgments

Thanks to Norm Katz of IP Consulting for sending me his thought-provoking e-mail message, and thanks to Doug Lea for reading and commenting on drafts of this and all my previous columns.

References

1. C. Shirky, "PCs are the Dark Matter of the Internet," *Clay Shirky's Writings About the Internet*, Oct. 2000; available at www.shirky.com/writings/dark_matter.html.
2. S. Vinoski, "Web Services Interaction Models, Part 2: Putting the 'Web' into Web Services," *IEEE Internet Computing*, vol. 6, no. 4, July/Aug. 2002, pp. 90-92.
3. D. Libes, *Exploring Expect*, O'Reilly & Associates, Sebastopol, Calif., 1994.

Steve Vinoski is vice president of platform technologies and chief architect for IONA Technologies. He currently serves as IONA's representative to the W3C Web Services Architecture working group.

IEEE Internet Computing

Editorial: *IEEE Internet Computing* targets the technical and scientific Internet user communities as well as designers and developers of Internet-based applications and enabling technologies. Instructions to authors are at http://computer.org/internet/author.htm. Articles are peer reviewed for technical merit and copy edited for clarity, style, and space. Unless otherwise stated, bylined articles and departments, as well as product and service descriptions, reflect the author's or firm's opinion; inclusion in this publication does not necessarily constitute endorsement by the IEEE or the IEEE Computer Society.

Circulation: *IEEE Internet Computing* (ISSN 1089-7801) is published bimonthly by the IEEE Computer Society. IEEE headquarters: 3 Park Avenue, 17th Floor, New York, NY 10016-5997. IEEE Computer Society headquarters: 1730 Massachusetts Ave., Washington, DC 20036-1903. IEEE Computer Society Publications Office: 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, Calif. 90720; (714) 821-8380; fax (714) 821-4010. Subscription rates: IEEE Computer Society members get the lowest rates and choice of media option — US$37/30/48 for print/electronic/combination. For information on other prices or to order, go to http://computer.org/subscribe. Back issues: $10 for members, $20 for nonmembers. Also available on microfiche.

Postmaster: Send undelivered copies and address changes to *IEEE Internet Computing*, IEEE Service Center, 445 Hoes Ln., Piscataway, NJ 08855-1331. Periodicals postage paid at New York, N.Y., and at additional mailing offices. Canadian GST #125634188. Canada Post International Publications Mail Product (Canadian Distribution) Sales Agreement #1008870. Printed in USA.