Lisp for Service Oriented Architecture Programs

by

Sheng-Chuan Wu, Franz Inc.

presented at

1st European Lisp and Scheme Workshop, June 13th, 2004

Abstract

A new software component architecture, the Service-Oriented Architecture (SOA), has emerged that promises to deliver the benefits of Information Technology (IT) integration to business. In this paper, we discuss the reasons why we need to embrace SOA today and how it changes the rules of software development and software business. We then review the characteristics of SOA, and explain why Lisp is the ideal programming language for this new software architecture. We conclude by presenting some of the new Lisp tools from Franz Inc. for SOA programming.

♦ ♦ ♦ ♦

One important question facing IT professionals and corporations today is how best to harness computing power to further improve business processes, having already computerized most business functions. This is especially critical in Business-to-Business (B2B) e-commerce. The few companies who have mastered B2B e-commerce, such as Dell, Wal-Mart, Cisco and Intel, dominate their respective markets.

For example, Dell can now manufacture and ship a custom-configured computer within 6 days of receiving the order on their on-line store. They carry zero inventory for parts or finished goods, and they are the lowest cost computer manufacturer in the world. They have accomplished this by completely automating the B2B transactions with all their suppliers and contract assemblers without paper or human intervention. Their sales force have direct access to all their suppliers' up-to-date inventories, pricing and production schedules, allowing them to configure the best computer for their customers in terms of price and capability while increasing the gross margin for Dell. All their suppliers also have access to Dell's sales data and sale forecast so that they can properly plan their inventories and production schedules. Today, Dell leads all computer manufacturers with an 18.6% of world market share, having ceding that position briefly after the HP and Compaq merger. And the gap is widening (HP-Compaq now trails with a 14% market share).

Similarly, Wal-Mart's low cost structure from its B2B system completely alters the retailing landscape, creating the famous Wal-Mart effect – wherever Wal-Mart goes, other stores either drop prices or face extinction. Today, Wal-Mart's annual revenue of US$256 Billion, rivals the GDP of Switzerland and Belgium, and is larger than the next three biggest retailing store chains combined.

However, according to an estimate, 95% of US companies employ little or no B2B e-commerce. Thus the question, "If B2B e-commerce is so beneficial, why don't more companies embrace it?"

The Business Dilemma

Today's B2B e-commerce system, as employed by Dell and Wal-Mart, relies on a hub-and-spoke system using proprietary business systems. Basically, to do business with Dell or Wal-Mart (the *hub*), suppliers or contractors (the *spokes*) must conform to whatever proprietary Electronic Data Interchange (EDI) formats and business processes dictated by the *hub*. Such a B2B infrastructure is neither portable nor scalable. A B2B system built for Dell will need to be rebuilt to work with another company such as HP. And each proprietary B2B system costs anywhere from half a million to many millions of dollars to build. It is little wonder that very few companies actually deploy a completely end-to-end B2B infrastructure. Manufacturers and retailers face a dilemma: embracing B2B means significant up-front investment and lock-in with a particular *hub*; not doing it risks becoming another victim of Wal-Mart.

The same difficulty exists in Enterprise Application Integration (EAI). EAI is intended to connect end-to-end all islands of business automation, which are a disparate group of incompatible software applications from different vendors, to streamline data flow and improve business processes. Traditional one-to-one interfaces between applications quickly exploded into an unmanageable n-to-n complexity. The complexity became even worse when integrating applications across business partners, as B2B would require. Later component architectures, such as CORBA and COM/DCOM, helped mitigate the n-to-n complexity, but still locked users into particular, incompatible middleware platforms without leveraging the Internet for cross enterprise integration. And, they are expensive to develop and deploy. Consequently, IBM found that its 36,000 customers, on average, spent 40% of their IT budget on EAI in 2003.

Service Oriented Architecture

The new Service Oriented Architecture (SOA) holds great promise to simplify implementation of EAI and B2B e-commerce. Basically, an application under SOA is presented to the outside world as a *service* rather than a *function*. To "call" this application, one sends a service request via a text document rather than a function call; in return, the application sends the requested result (service) as another text document, completely bypassing the issues of platform (or language) and implementation compatibility. One emerging SOA is Web Service.

Web Service leverages the open HTTP Internet and XML. A Web Service application is invoked through a declared Application Program Interface (API) – SOAP, described by a service description language – WSDL, and published in a service registry – UDDI. Web Service application components are implementation transparent and loosely coupled, allowing one to compose an application from other Web Service application components readily. One may even swap one component with another component that provides similar but better service at any time without hassles, a feat heretofore not possible with other component architectures. Consequently, It is much easier for one service (application) to be shared by many users (applications), and the user (application) has many more services to choose from. This inevitably increases the utilization and availability of services, leading to lower costs for such services.

New Software Business Model

With Web Service, It is now possible to build a portable and scalable B2B e-commerce infrastructure. The implementation independence of Web Service allows easier separation of business functions from business processes. Basically, all standard business functions, such as accounts receivable, accounts payable, inventory management, order management, etc., can be presented as *web services* with a uniform service API. Business applications, such as B2B e-commerce, can then be built on top of these *web services* for different business processes and individual partners. Since web services are built on HTTP Internet, integrating cross-enterprise business functions over the network is also simpler. Because of separation of business functions from business processes, It is much easier to change an application for a new business process.

Not only do changes in market conditions and technologies demand updates to business applications, increasing globalization, outsourcing and partnering also require constantly adjusting existing business processes. Furthermore, the new ASP (Application Service Provider) business model means a much lower barrier to enter into established markets (for example, Google replaced other entrenched search portals within just 4 years). No more elaborate distribution channel is required to reach customers globally. Only those that keep up with new market demands and employ newer and better technologies will survive and excel. In this new service era, the only constant is *change* and the pace of *change* is accelerating.

While SOA (and Web Services) provides a framework that enables easier development and evolution of applications, one still needs a software tool that allows rapid application prototyping and evolution to meet such demands. And, Lisp is an ideal software tool for such SOA applications.

Lisp Programming Model Ideal for Composing SOA Applications

Typically, Lisp programmers develop an application interactively, building and composing individual functions from the bottom level upward. Such interactive, bottom-up programming style fits very well with SOA application development, where one may want to try out various web services while composing an application. Most Lisp implementations also come with extensive built-in high-level functions and abstraction, making interactive program development productive. Additionally, an interactive programming environment requires a good memory management system. Lisp, having had long experience with automatic memory management, is capable of scaling up to very large and complex applications.

When an application, developed interactively, is ready for deployment, most modern Lisp implementations provide a compiler that compiles Lisp code into machine instructions, running at processor's speed. Most significantly, a deployed Lisp application can include a debugger, a performance profiler and the full development environment. This unique capability makes it very easy to debug, patch and modify an application, a feature very desirable in the service era.

Lisp Dynamism Great for SOA Server Applications

Lisp has the most comprehensive and truly dynamic object system (CLOS). Changes to an object class will update all existing object instances automatically and lazily, while the

application is running. Class methods and functions can also be defined or redefined on the fly. Even the class of an existing object instance can be changed. All are accomplished without recompiling or even stopping the application.

Lisp dynamic objects allow one to compile and load code incrementally at run time, and to debug, edit, compile, patch and test an application without stopping the run. This is particularly significant for server applications, where programmers have full control of running and evolving the applications. Lisp provides the classic Rapid Application Development and Evolution capability coveted by application developers especially in this service era.

Lisp Syntax Great for Web Service Document Manipulation

Comparing to other programming languages, Lisp has a very simple syntax, consisting of only a single expression type – the s-expression. An s-expression represents a data atom (number, character, symbol, etc.) An s-expression is recursively defined by another s-expression, enclosed by two parentheses. Both Lisp code and Lisp data are represented textually using s-expressions, allowing a Lisp application to generate Lisp data, turning the data into Lisp code, then execute the code on the fly. Significantly, s-expression maps very naturally to Web Service XML data. In fact there is a one-to-one mapping between the two. This makes Lisp a natural language to represent XML data and to process web service requests.

Lisp Macro Greatly Simplifies Web Service Programming

Lisp macros may be one of the most useful programming features. Unlike macros in other languages, Lisp macros are not just used for text substitution. A Lisp macro can encapsulate a design pattern, a protocol or a domain policy or rule; all of which will be automatically enforced or embedded in the expanded code, greatly simplifying the application development task. Lisp macros are routinely used to create extension languages specific to particular domains. Using Lisp macros, code tends to be cleaner, more consistent, and easier to read. In many occasions, it results in 10x code base reduction.

Lisp macros are particularly suitable for Web Service programming. Web Service programs are built on top of a long stack of protocols – HTTP, XML, SOAP, WSDL, UDDI, WS-Security, WS-Transaction, WS-Coordination, etc – in order to facilitate a robust and secured web service transaction. All such protocols are important infrastructure but by themselves add no value to the services. All these tedious protocols can be encapsulated and enforced within Lisp macros, totally hidden from programmers. In other words, by using Lisp macros one extends the Lisp language for Web Service application development.

Lisp Perfect for Web Service Based B2B e-Commerce

Lisp is not only very suitable for SOA and Web Service programming, it can be even more useful for modeling Business Process Execution Language for Web Services (BPEL4WS).

Business application integration needs more than just the ability to exchange messages through standard protocols. Business interaction (as in B2B e-commerce) involves "sequences of peer-to-peer interactions, both synchronous and asynchronous, within

stateful, long-running interactions among two or more parties", while SOAP, WSDL and WS-Transaction support essentially stateless and un-correlated interactions. Therefore, enterprises need a business protocol (such as BPEL4WS) on top of SOAP and WSDL to describe the business behavior having cross-enterprise business significance without revealing internal implementation of individual parties. As such, BPEL4WS must include data-dependent behavior, exceptional conditions and their consequences, and long-running interaction with fully maintained state and cross-partner coordination.

Such a protocol for modeling real-world business processes comes awfully close to resembling an executable language. In other words, the protocol language for scripting a business process must at the end be part of program execution. And, business process evolves and changes as market conditions, technology and partnerships change. Being "a programmable programming language", Lisp is ideally suited to be both the scripting language (using macros) and execution engine for BPEL4WS.

Lisp Web Portal Tools

Allegro CL (a Common Lisp implementation by Franz) provides a set of tools for building and deploying dynamic web portals. Web portals can be viewed as part of SOA application family; only in this case, the services are accessed directly by users from a web browser rather than by applications using SOAP calls.

Allegro CL provides a free, open-source HTTP server in Lisp – AllegroServe – running on many other Lisp implementations. It serves both static and dynamic pages, and provides mechanisms for standard access control, logging and secure transactions. A framework based on the model-view-controller paradigm (Allegro Webactions) is developed on top of AllegroServe, to help simplify web portal development and code organization. It provides both session and state support automatically, even with browser cookies turned off. For dynamic web page generation, Webactions framework includes CLP pages. A CLP page consists of standard HTML code and special HTML tags for calling CLP functions to generate dynamic content. It separates HTML content from program code, allowing designers and programmers to work on the same web page without interfering each other. A CLP Lisp function can be as complex as necessary and it runs compiled at processor's speed.

Lisp Web Service Tools

Allegro CL recently included a very speedy SAX XML parser that is the basis for all web service processing. It provides Lisp interfaces for building both SOAP clients and SOAP servers. It has a WSDL compiler to automatically compile WSDL files from other web services into Lisp stubs, and a WSDL generator to generate a WSDL file automatically from the Lisp SOAP server API. Other Web Service infrastructure tools built on top of XML, SOAP and WSDL are to be released soon.

Conclusion

Service Oriented Architecture (SOA) and Web Service are the next IT mainstream, and now is the time to embrace them. Under SOA, one can compose an idea into a software application very quickly. To survive long term, an SOA application must evolve constantly and quickly. Lisp, the 2nd oldest programming language but still very relevant to this new programming paradigm, is the ideal software tool for SOA programming.