A Model-Driven Approach for Building Customized Distributed Applications

By John Pompeii and Scott Danforth

Secant Technologies, Inc. -- April 25, 2001

A Model-Driven Approach for Building Customized Distributed Applications

By John Pompeii and Scott Danforth
Secant Technologies, Inc. -- April 25, 2001

Overview

Today's businesses are presented with the ever-increasing pressure of quickly and successfully converting business vision into technical reality.

At one end of the spectrum, the Internet provides highly effective avenues for reaching potential customers and for operating businesses more efficiently. Today, customers and vendors alike expect to be welcomed into, and be able to actively participate in, the overall operation of business relationships as never before.

Yet, it is at the other end of the spectrum that most projects end up. Statistics published by the Standish Group show that nearly 75% of all software development projects and almost 90% of n-tier application projects fail to meet their objectives. Application software customized to the specific requirements of the business and its users remains very difficult to build.

Fortunately, model-driven software generation coupled with a high-performance, scalable run-time infrastructure offers a way to bridge the chasm between corporate vision, and the timely delivery of advanced software solutions that are essential to streamlining business processes and building competitive advantage.

Great Expectations

In the last few years, a number of high-profile web-oriented businesses have pioneered use of the Internet for business processes. Numerous patent applications have been filed in this regard, which only serve to underscore the pioneering aspects of this integration of the Internet with the world of business. Examples of interaction types already utilized in extending business processes on the Internet include the following:

- In the *shopping cart* paradigm, customers browse product offerings, add their selections to a virtual shopping cart, and then "check out" by choosing between different shipping options and providing credit card or other payment information. This interaction style supports businesses that offer products for sale to the public.
- In the *auction paradigm*, users can offer items for sale, or bid on items. A business might utilize this model to offer limited amounts of outdated or used merchandize and allow customers to bid on them.
- In the *service-tracking* paradigm, customers are provided with the means for identifying themselves and a service they've contracted for, and for accessing information concerning the current state of that service. This is a useful customer support model for

businesses whose services take place over a period of time and move through various phases of execution.

- In the *continuous service* paradigm, customers identify themselves and update information that influences the execution of an ongoing service for which they've contracted. This model is useful for just-in-time supply chain management.
- In the *discussion group* paradigm, a web site provides a page that contains threads of discussion that take place over a period of time. Users can create new threads, add messages to previously created threads, and can send themselves an email message whose content records all of the messages currently in a thread. Threads are typically presented in order of their creation (youngest first), and are purged according to a policy determined appropriate for the site. This is another very useful paradigm for customer support.
- In the *chat room* paradigm, customers interact with other customers, and, often, business representatives, by sequentially adding messages to a common blackboard that is continuously updated and seen by all participants. This interaction style is a good way of providing immediate responses to customer questions and issues. Often customers themselves can provide assistance to those less experienced, thereby offloading work from business representatives.

Variations on these and many other models of interaction have become familiar to the public as they have become integral to the B2C environment. For a while, the Internet was exploding with businesses for which such interactions were the sole source of revenue. However, because it makes such good sense, customers have come to expect all businesses to provide Web-based applications that support efficient and appropriate interactions, hence the emergence of B2B.

Of course, most businesses would love to extend their competitive advantage by utilizing technology. A well-designed Internet-based application that extends advanced business processes can provide dramatic cost savings when compared with a similar service, supported, for example, by trained office personnel. In many cases, it is only the possibility of a computerized Internet-based solution that makes such services economically and operationally feasible.

Customizing the Approach to the Business

The best way for a business to provide advanced Internet-based interactions is to begin with a model that represents the overall business. This involves an extensive strategic analysis of the business processes and the relationships between those business processes. Such processes have consistently defined operational competitive advantage for leading organizations and will continue to define this advantage as they are extended over the Internet. This business process model allows designers and business managers to focus on identifying portions of existing or new business processes that can be computerized and, perhaps, executed over the Internet. Because they are part of the overall business model, and more importantly competitive advantage, the resulting applications are easily integrated into the overall functioning of the business. A well-defined business process model clarifies how common business objects manipulated by these applications relate to and are used by other business processes that may not be computerized.

This "custom" approach allows new levels of customer interaction to be understood and integrated into the current functioning of a business. And, it offers an obvious migration path into the future, allowing additional computer-based automation to be added over time. The more detailed and appropriate the model of a business is, the more likely it is that applications based on the model will fit naturally within the operation of the business

As implied above, a model will sometimes suggest the possibility of new processes based on existing objects. As a business evolves over time, it may deal with new objects, or specializations of previously understood objects. For these and other reasons, a business model will naturally evolve over time, incorporating new objects and processes. Assuming a customized approach to computerized support, it follows that the computer applications used to support a business must also evolve over time.

All of this suggests the absolutely fundamental importance of modeling. And, it also suggests the value of carefully defining as much of an application as possible from a model, so that when the model changes, so can many aspects of corresponding applications. Today's organizations struggle within this evolutionary environment and need solutions that to some degree automatically deliver the application code, thereby minimizing the need for additional programming.

This does not mean that automatic programming and reprogramming based on iterations of a general object model is yet a reality. But, an important and very real differentiating aspect for software vendors lies in the degree to which their products approach this objective. For example, a top-level question for a middleware product intended to assist in the production of software would be whether it supports any modeling language at all. And, if so, how is iterative development and ongoing maintenance supported?

Time to Market is Crucial

Aside from the fact that model-driven software better supports and defines the integration of computerized applications into a business's evolving operational procedures, model-driven software can also be more efficient to create. In today's volatile marketplace rapid application deployment is the crucial advantage that a model-driven approach provides.

Clearly, a number of possibilities exist for integrating programming tools used for modeling into the software production process. However, existing and future model-driven tools only solve the problem of quickly and accurately defining the application solution. Many of the most important demands placed on today's applications have very little to do with specific aspects of the business model they support. Just as customers expect functional and appropriate applications, they also expect good response times and high availability.

So, applications need to scale in order to maintain rapid response times as usage increases, and they need to be fault tolerant in order to satisfy availability requirements. And, while it may be possible to specify response time and availability constraints in a model, there is little hope at the present time of automatically generating application code that somehow assures scalability and fault tolerance in a way that is derived from these constraints. Furthermore, if time to market is crucial, the idea of requiring software developers to come up with application-specific solutions to these problems seems counter-productive.

If we can't generate the solution from the model, and we don't want to design the solution into the application, what is the alternative? Leading edge software developers believe in a two-pronged approach: use model-driven tools to generate customized software; and support this software at runtime with a model-driven infrastructure that provides scalability and fault tolerance.

Specifically, this approach is best characterized as follows: First, to the extent possible, provide a model-driven runtime infrastructure that offloads important aspects of the application solution from the programmer. And, second, to the extent possible, generate remaining software for the programmer in a way that supports iteration and evolution of the model over time.

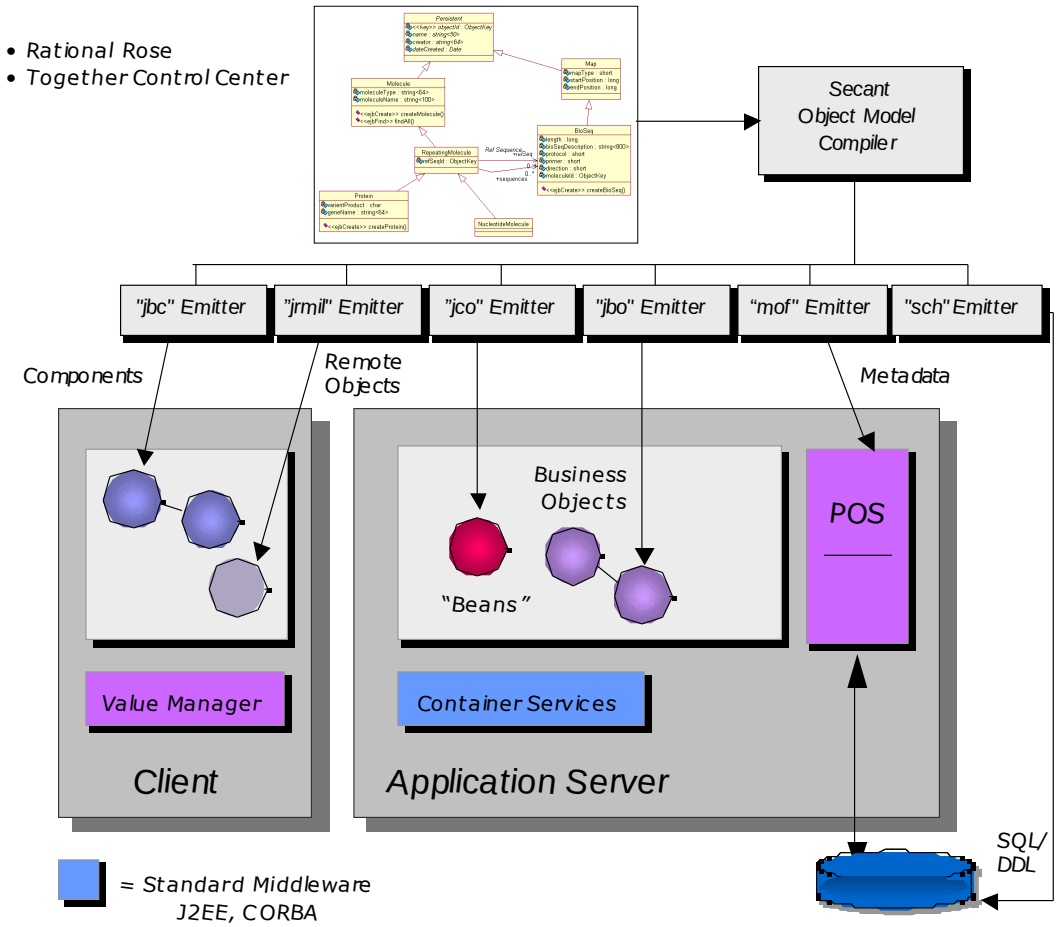
The advantages of this approach are compelling. It minimizes the amount of code that needs to be written by application developers; supports iteration during development allowing for application evolution over time; moves some of the most complicated code needed by the application into a portion of the system that is provided by the tool vendor; auto-generates code written by expert programmers, alleviating command line code generation; and has the advantage of years of testing. The result minimizes time-to-market and maximizes the value of the applications that are produced.

Overall a model-driven runtime infrastructure is an important piece of the overall time-to-market equation that many developers overlook because they want to write code. But, if you want to get to market, you don't want to write code. The more code you write, the worse off you are – an important corollary of which is the following: The smarter you think you are, the longer it will take you to release your application. Nobody wants to hear this, but it is true. The most important aspect of application code development is the code you don't need to write, because the runtime infrastructure provides the required support based on model information.

ModelMethodsä Delivers

Secant's infrastructure-based approach to model, code generation, and runtime support offers a representative picture of the current state of the art in this regard. For example, Secant's model-driven runtime support includes distributed transactions, security, and persistence. These runtime services include requirements specified by the J2EE standard for Enterprise Java Beans, as well as relevant OMG CORBA standards. Other important APIs and architectural support not directly related to any specific business model include support for servlets, JSPs, web site management, distributed locking, identity management, scalability, fault tolerance, and load balancing.

On the code generation side of things, Secant tools generate complete implementations for persistent and transient objects (with stubs for business logic) from an object modeling language that includes full support for inheritance and relationships between objects, as well as essential deployment decisions. The code generation tools support incremental development and evolution of generated code as the model evolves over time. The diagram below illustrates the code generation approach:



The technology works by first defining a UML language domain for distributed applications that includes standard UML extensions for specifying the details of remote objects, object-to-database mappings, client components, and distributed application packaging and partitioning. These extensions are used by application designers to specify the details of their system in a UML model.

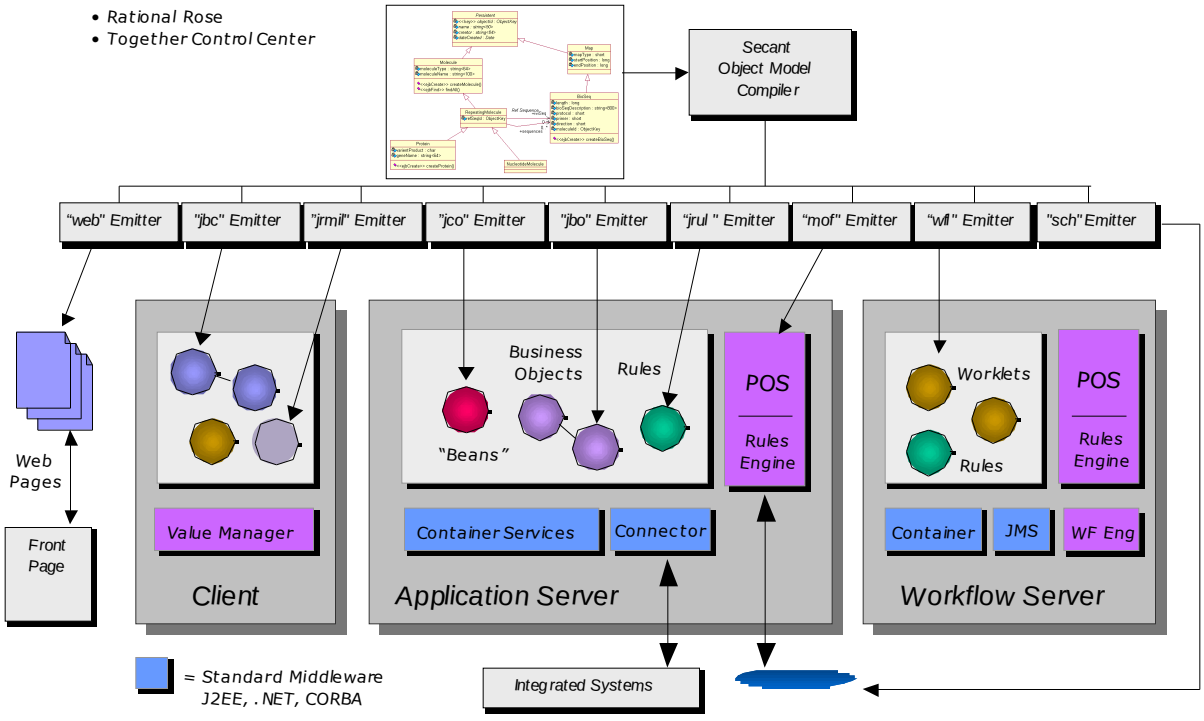
When the model is ready for code generation, the user accesses the Secant code generation wizard from within the case tool and clicks the "Go" button. The model integration technology then reads the model and sends it to the Secant Object Model Compiler for analysis. Depending on the selected target system (J2EE or CORBA), the compiler will load a suite of code emitters where each in turn generates its portion of the software. The modular emitter approach allows for different application models to be selected. For example the system may only be a two-tier application or it may require Java clients to access a C++ CORBA server.

The system diagrammed above represents the current shipping version of the ModelMethodsä

Enterprise Server. This technology has been proven to generate up to 80% of a project's source code and reduce the overall initial development time by 25-50%. For follow-on releases, the reduction in development time is further reduced to 60-80%. These figures offer significantly better return on investment figures compared to all other infrastructure technologies in place today. Further, from statistics published by the Standish Group, nearly 90% of all n-tier application projects are cancelled, or failed on budget, delivery, or objectives. The ModelMethods technology can significantly reduce the risk of project failure because of the drastic reduction in hand coding required to build a distributed system.

Direction for Future Releases of ModelMethods

Although these figures are impressive, it is the goal of Secant to increase the code generation percentage to 100%, support all components of a distributed system, and increase our platform support to other J2EE environments and for .NET. The diagram below illustrates the next generation of the ModelMethods technology:



This diagram illustrates an extended reference model for distributed computing that includes workflow, process management, business rules, systems integration, XML, and web applications. To support this extended model, additional UML extensions, code emitters, and runtime frameworks are required and are being developed. The result though will be a very comprehensive coverage of a system that can be specified in a UML model and then ultimately generated. One of the key additions is the specification of business logic and rules in the model

that allows the implementation of the application to be targeted towards any platform or object oriented language.

Conclusion

In contrast with the approach described so far, some vendors provide code generation from object modeling tools, but don't include the model-driven runtime infrastructure that is necessary to complete the time-to-market equation. Others provide the basic, model-driven runtime infrastructure required by the EJB standard, without including model-driven code generation that supports iterative development. All of these technologies can be and are used to build and successfully deploy applications. But we believe that the most reliable and rapid routes to success are those that begin with an object model and then combine model-driven code generation with model-driven runtime infrastructure support to produce a customized application that can be evolved over time and is tolerant to changes in middleware.