30. Implementing Models in Applications

DE + IA (INFO 243) - 5 May 2008

Bob Glushko
Plan for Today's Lecture

Model-Based Integration and Applications
Model-Based User Interfaces
Composite Applications
Mash-ups
Model-Based Applications

Creating information and process models is a significant investment in capturing context-specific (or application-specific) requirements in a technology-neutral and robust way.

The abstraction in a good model makes it simpler and easier to work with than the specific technologies of implementation.

A Model-Based Application explicitly uses a model as a specification for generating code or configuring an application.
The Scope of Information Integration
Example: Models in MRP/ERP Cycles

MRP systems (which evolved into ERPs) plan production, procurement, and distribution for an enterprise and you can view them as embodying a generic model of a manufacturing enterprise that is configured for a specific firm.

For each of the products a company manufactures there is a "recipe" or "parts list" or "bill of materials" that lists the parts or components that go into it (and how many of each).

\[
\text{BILL OF MATERIALS} \times \text{SALES FORECAST} = \text{PARTS WE NEED}
\]
\[
\text{INVENTORY ON HAND} = \text{PARTS WE ALREADY HAVE}
\]
\[
\text{PARTS WE NEED} - \text{PARTS WE ALREADY HAVE} = \text{PARTS "SHOPPING LIST"}
\]

"Shopping lists" for each production cycle are then sent to suppliers as EDI or XML documents.
The "Componentized" Bank
"Componentized" Bank Value Chains

Disaggregation of the value chain

Customer

Internet
Call Center
Branch

Internal and external distribution units

Rheinhyp/Eurohyp AG
Other product suppliers

Settlement unit (prompter AG)

(1 - a)
How Else Could You Do It?

I can't imagine why you'd design and implement an application without making the information and process models "first class citizens" but...

There is little separation of information models and the code that handles them in most scripting languages.

Many programs "flatten" and "deconstruct" a hierarchical document model into a set of "attribute-value" pairs.

The iterative, heuristic, and non-deterministic techniques by which most people design user interfaces don't emphasize the information and process models.
Platforms for Model-Based Applications

A software platform solves some class of generic problems so that application developers can focus on the context-specific parts. Operating systems, programming languages and their runtime libraries, databases, software frameworks, middleware "service bus," web browsers ... are platforms...

Ideally, the context-specific parts of the solution that are based on the model(s) remain distinct and inspectable apart from the generic functionality provided by the platform.

In this case, the platform interprets the model to determine how the software behaves -- the model configures or customizes the platform.
Benefits of Model-Based Applications

- Easier to understand the software architecture
- Generated code is of higher quality than hand-crafted code
- Easier to maintain (regeneration when model changes)
The "Old Way" -- A2A Integration

The "old" way to integrate two applications within an enterprise was to write a custom program that fit only between the two of them.

This method has traditionally been called A2A (Application to Application) integration.

The technical approach is usually file transfer or remote procedure call mechanisms.

The low-level granularity of APIs means this tightly-coupled connection isn't model-based.
B2B Integration

In contrast to traditional A2A integration, where neither application interface is visible outside the company, B2B integration is concerned with automating the exchange of information between enterprises.

Since at least one of the endpoints isn't under your control, you can't use a tightly-coupled and granular approach for information transfer.

Loose coupling is a fundamental design principle for the Web, and today almost all B2B integration software uses Internet and Web standards with XML documents as the message format and provides "bridges" to and from legacy formats.

Much B2B integration software has moved to a "hub" or "bus" architecture to replace point-to-point A2A integration with a more scalable approach.
We Want Loose Coupling AND Smart Information

- Context and structure-based text objects (SGML, XML, databases)
- Structured electronic text with intrinsic formatting or “difficult” syntax (HTML, EDI)
- Formatted electronic text (word processing)
- Unstructured electronic text (ASCII)
- Printed Text
Architectures for Transformation: Connectors

It is the sender's responsibility to transform the message (the envelope and its contents) into the model needed by the receiver.

Modern connector software may include predefined data translation solutions (adaptors) for common ERP systems, enterprise databases and enterprise messaging software.
"Foreign Protocol Integration" via Connectors
The receiving application accepts whatever message the sender wants to produce and does the transformation. This is the "be liberal in what you accept" strategy.

This strategy captures existing "transaction flows" by making it easy for legacy systems to join the marketplace or networks.
"Foreign Protocol Integration" via Gateways
Architectures for Transformation: Services

The transformation can be carried out by a third-party service provider - a transformation intermediary.

This is often the most efficient architecture if there is a "hub" or "interoperability" message format (such as xCBL or UBL) because it minimizes the number of transformations that will be required.
"Foreign Protocol Integration" via Service
Model-Based User Interfaces

User interface design started as a distinct activity in the 1980s, and has been dominated by iterative and heuristic techniques ever since.

In the 1990s the goal of model-based UIs emerged with the hope that "automatic generation of window and menu layouts from information already present in the application data model can relieve the application designer of unnecessary work while providing an opportunity to automatically apply style rules to the interface design."

Some people starting calling this the search for the "Big Red Button," and in many cases it involved user interface modeling languages (expressed in XML) from which UIs would be generated.
Many Small Red Buttons?

An alternative to the search for the BRB is the goal of partial automation for user interface generation:

Tools that generate prototype from specifications

Tools that synthesize use cases into sequence diagrams

Tools that merge sequence diagrams to hide states that have no UI implications

Tools that generate UI skeletons or scaffolds while enforcing layout constraints

Tools that generate a family of UIs via "graceful degradation"

UI Design Patterns
The "thin" browser platform freed developers from having to develop "thick" applications that needed to be installed and supported on every user's computer.

But early versions of HTML could only support static structured and hyperlinked narrative document types.

CGI, CSS and HTML forms were introduced for more interactivity, layout control, and transactional content.

Java applets, Flash, DHTML, Java/VBScript, Swing etc. emerged as (often proprietary) alternatives to HTML to build more complex applications.
Can You Be Too Rich or Too Thin or Too Standard?

These so-called "rich" or "smart" client application platforms are also "thick" because they require substantial client-side software.

The dilemma for web application developers is whether to rely on the standard and ubiquitous/universal client of the browser or to take on the problem of installing the "thick" client into the browser.

So the goal is "rich" capability in a "thin" client -- or finding ways to eliminate the problem of installing and managing the required plug-in.

Can we be rich in a standard, model-based way?
Today's Platforms for Web Applications

XFORMS support in browsers to enable client-side validation and XML data interchange

AJAX (asynchronous Javascript, eliminating full-page refresh and enabling much finer-grained interactivity) concepts being built into platforms

Some of these platforms are proprietary (Adobe Flash and Flex, Microsoft Silverlight) but others are open (Firefox XULRunner)

Important emerging functionality is ability to support online/offline transitions (Google Gears, Firefox PRISM)
Model-Based UIs and the Document Type Spectrum

Most platforms for model-based applications are form-based and targeted toward transactional document types

"E-book" readers are a notable exception, designed as platforms for narrative document types to provide "book-like" display and interactive functionality
"E-Books" and Platforms for Structured Publications

The simplest case of structured publication is publishing a single document in a way that lets the user interact with it by exploiting its (content) component structure.

Many e-book platforms exist, and many have adopted XML formats (native or interchange) with Open eBook (but not all of them).

E-book readers differ in how much structural and presentation fidelity they enable and in functionality that "goes beyond the printed book."
Platforms for "Active Documents"

"The document is the application"

Embed processes such as retrieval, data acquisitions, transactions, workflow, or archiving in the document itself

Are highly structured, yet are capable of changing their structure and presentation "on the fly" for different users, contexts, or uses

Become the de facto interface -- the gateway -- into the multiple applications that are related to the information
"Active Document" Platform Examples

IBM Workplace Forms (now "Lotus Forms")
Adobe Intelligent Document Platform
Justsystems xfy
Composite Applications

Create a single interactive user experience by combining multiple business functions or information sources

Reuse the application logic from these existing applications / sources

Do not change the constituent services/sources, which can still function independently in their original contexts
Application Integration "By Eye"
Swivel-Chair "Application Integration"
Composite Application
What are the dependencies among the component services?

Are these dependencies logical/intrinsic or contextual/selective?

Service dependencies determine choreography of transactions and collaborations
Identifying "correlational" or "glue" components needed to interconnect or synchronize services

Harmonizing the semantics of these components

But these components might have different names in different services

Input forms in user interfaces == document assembly models
The Vision: Dynamic Service Discovery and Composition

Business processes are increasingly global and involve widely dispersed parts of an enterprise or multiple enterprises.

An "agile," "adaptive," "on demand" [insert other buzzwords here] enterprise needs to be able to quickly and cost-effectively change how it does business and who it does business with (suppliers, business partners, customers).

So ideally, the component services that come together in composite applications can be dynamically discovered and combined from different firms.
Requirements for Dynamic Composition

Do I know what service I'm looking for?

Do I know where to look? Can I find the service I'm looking for?

Is the service compatible from a business perspective?

Is the service technically compatible? (technical interface compatibility)

Can I transform the service inputs or outputs to achieve compatibility?

Does the service meet my quality and performance requirements?
The Reality -- Hand-Crafted Composition

Most services aren't listed in public registries

Service descriptions are often limited to technical interfaces (WSDL), lacking information needed to assess business model and business process compatibility

Emerging platforms for composite applications are limited in the kinds of document interfaces they can handle (e.g., optimized for services operating on relational databases)

Some platform support for reusing integration semantics, but not sufficiently grounded in ontologies or reference models to enable automated mapping and transformation
Mashups

"A web application that combines data from more than one source into a single integrated tool" (Wikipedia)

"A lightweight tactical integration of multisourced applications or content into a single offering" (DMReview.com)

IS THERE ANYTHING NEW OR DIFFERENT BEYOND "ACTIVE DOCUMENTS" AND "COMPOSITE APPLICATIONS"...
Web Mashups

- Visual centric
- Data relationships are simple
- End-user driven enabled by APIs / tools
- No QoS guarantees – as is
- Simple Web security
Data Mashups

- Data & Information centric
- Deep transformations & semantic relationships
- Interact with Enterprise ecosystem
- Business analyst driven w/ user control options
- Enterprise QoS- Query Optimization, Reliable, Failover, etc.
- Policy-based security
Mashing the Document Type Spectrum

Denodo Platform

- Key Advantages:
  - Diverse Source Types
  - Shared Data Model & Semantics
  - Real-time and Batch
  - Non-intrusive / Agile

Integration Complexity

High

- Enterprise App. Integration (EAI)
- Extract-Transform-Load (ETL)
- Enterprise Info Integration (EII)
- Desktop Tools e.g. Excel

Integration

Low

Structured
Semi-structured / Web
Unstructured

Data Type

Freeware / Shareware

Search

Freeware / Shareware
Enterprise Data Mashup Architecture

**Denodo Platform**
- Multi-mode Access: Data Services, API, Query, Search
- Process Engine: Transform & Relate Information
- 3 Data Engines: Read & Write

**Enable & Extend Applications**

**Build Information Relevance**

**Access Any Data Source Anywhere**

- Databases
- Applications
- Web Services
- Websites
- Web Apps
- Hidden Web
- Files
- Emails
- Documents
For Wednesday May 7

No assigned readings

Work on your course projects!

XForms demos for Courseland

Reports from the OASIS Symposium
For Monday May 12: Final Presentation
Metamodel [1]

Aim for 20 minutes to allow plenty of time for discussion and close looks at your modeling artifacts

Introduction to your project and context (suitable for people who've never heard about your project)

The processes/services and associated documents that you've identified

Results and insights from component harvesting and consolidation activities; any innovations in your modeling artifacts or notations

Results and insights from aggregation activity; any innovations in your modeling artifacts or notations

Results and insights from document assembly activity; any innovations in your modeling artifacts or notations
Final Presentation Metamodel [2]

Most helpful / least helpful document engineering concept or technique

Any other big ideas or lessons learned about your domain

Future plans for the project, if any

Written report and / final artifacts turned in by Friday 5/10