Model Driven Enterprise Integration

Dr. Vladimir Bacvanski (vladimir@inferdata.com)
Petter Graff (petter@inferdata.com)
Outline of the Talk

- Introduction
  - Enterprise Integration
  - Model Driven Architecture (MDA)
- Enterprise Domain Models
  - Structure
  - Behavior
- Enterprise Design Models
  - Refinement and maps
  - Choreography
  - Collaborations and their models
  - Components and their Models
- Enterprise Components Implementation
- Conclusion
Introduction

- Enterprise integration concepts
- Deficiencies of current integration approaches
- Model driven development
What is Enterprise Integration?

- Bringing multiple enterprise parties together into one cohesive application: People, Processes, Information

- Common systems integration forms:
  - End to end
  - Broker based
Deficiencies of Everyday Integration Efforts

- End-to-end and broker solutions:
  - End-points of communication are coded in specific technology
- Integration is done on a very low level
- The same work is repeated as the technology changes
- Knowledge about the integration is tacit and lost with the change of technology
Forces for Integration

- Knowledge reuse
- Standards
- Minimizing manual work
- Traceability
MDA separates specifications of system functionality and implementation on a specific technology

- Generate the implementation
PIM and PSM Separation

- Platform independent (PIM) and platform specific models (PSM) are separate
- Tools generate a set of artifacts from the combination of PIM and PSM
Benefits

- Preserving the investment in knowledge
  - Independent of implementation platform
  - Tacit knowledge made explicit

- Speed of development
  - Most of the implementation is generated

- Quality of implementation
  - Experts provide transformation templates
  - 100% continuity from specification to implementation
Enterprise Domain Models

- Representing Domain Structure and Behavior
Enterprise Domain Models

- UML based specification of a problem domain has two parts:
  - Structural definition specified in UML using **Type Models**
  - Behavior definition using abstract **Events**

- Goals:
  - Capture the domain without regards to
    - Enterprise design (workflow)
    - Technology
  - Provide vocabulary for all implementation components in the enterprise
  - Serve as a glue between different applications and Enterprise processes

Gas Station Structural Model

Gas Station Behavioral Model

```plaintext
purchaseGas( gs: GasStation, c: Customer,
quantity: Fuel, cost: Money)

Preconditions:
-- 'gs' has sufficient fuel
gs.inventory >= quantity

Postconditions:
-- The fuel quantity has been transferred ...
```

Copyright © InferData 2002
Domain Model – Gas Station Structure
Domain Model - Gas Station Behavior

Event:
purchaseGas( gs: GasStation, c: Customer, quantity: Fuel, cost: Money)

Preconditions:
-- ‘gs’ has sufficient fuel
   gs.inventory \geq quantity

Postconditions:
-- The fuel quantity has been transferred from ‘gs’ to ‘c’
   gs.inventory = gs.inventory@pre - quantity and
   c.inventory = c.inventory@pre + quantity and
-- The cost has been transferred from ‘c’ to ‘gs’
   gs.balance = gs.balance@pre + cost and
   c.balance = c.balance@pre - cost

- Event specifications specifying effects of events on the domain
- Template using
  - Name of the event
  - Typed parameters to the event
  - Preconditions – The event never occurs if…
  - Postconditions – If the event occurs, the effect on the domain is…
Role of Domain Models for Enterprise Design

- Domain models provide for:
  - Understanding the real needs of the enterprise
  - Alignment of business goals with software systems
  - Continuity through the development lifecycle
  - Common vocabulary across the enterprise
  - Reuse of knowledge

- Domain models are reusable and adaptable
Enterprise Design

- Critical design decisions
- Refinements in time and space
Critical Design Decisions

- Where and how are the data stored?
  - Who owns the data?
  - How do we improve performance through caching?

- Where are rules of the enterprise implemented?
  - The impact of locality on performance

- How is the system security achieved?
Axes of Refinement

- Enterprise design is a result of a series of refinements of the domain model
- Two possible dimensions of refinement:
  - Refinement in space
  - Refinement in time
Gas Station Domain Model

- **Highest possible abstraction level**
- **Abstract but precise**
Refinement in Time

- **PurchaseGas** refined into *fill* + *pay*
- **Temporal situations requires new state space**

*FuelingTransaction*
Refinement in Space

- Refinement in space introduces enterprise components with responsibilities

Customer | Pump | Gas Station Controller | Cash Register | Cashier

fill | pay
Dependency on Platform Specific Capabilities

- Final refinement of choreography requires knowledge of infrastructure capabilities
  - Synchronous vs. asynchronous, guaranteed delivery, etc.
The enterprise has many actors
- Many systems
- Many different kinds of users

Many technologies used to connect the actors
Platform Independent Choreography expressed through

- UML Activity Diagrams
- OORAM Role Models
- Component Collaboration Architecture (CCA) and Enterprise Collaboration Architecture (ECA)
CCA Example

- CCA enables description of collaborations between parties
The enterprise choreography can be mapped on to multiple underlying technologies.
Platform Specific Choreography Models

- XLANG
- MQ Integrator
- Document Centric WebServices
- RosettaNet
- ebXML
- Free form protocols
Continuity and Mappings

Domain

Deployment

Choreography
Model Driven Integration in the Enterprise

- Starting from scratch is unlikely!
- Start from existing collaborators:
  - Develop their domain models
  - Harmonize the models
    - Define the common areas: the common vocabulary parties share in collaboration
  - Expose the legacy system functionality so that the interfaces are aligned with the domain models
- From platform independent models, generate/write the platform specific models and implementation
Dealing with Legacy Components

- Legacy components are accessed through interfaces that remain stable
- Integration technology change and the adapters with them, but interfaces remain the same
Practical Strategies for Integration

- Domain models are critically important
  - Focus on modeling the areas essential for collaboration
  - Domain models can span several areas and systems
  - Verify the collaborations with stakeholders

- Use metadata to integrate information from different enterprises

- It is critical to design around the characteristics of underlying technologies
  - Today this is mostly manifested in manual coding or definition of templates used for transformation
Tool Support

- Several MDA tools on the market
- InferData MCC: Model Component Compiler
  - A small, practical MDA tool
- Currently focus on:
  - J2EE platform
  - Web Services integration
- Highly modifiable and pluggable architecture
MCC Input Models

Enterprise Context Model

Enterprise Orchestration

Platform Configuration

Platform Independent UML Models

WebSphere

DB2

Struts

Cactus

Platform Dependent XML Specifications
MCC Inputs and Outputs

Model Component Compiler

Analysis Patterns
Architectural Patterns
Design Patterns
J2EE Patterns
Framework Profiles
Best Practices
Optimizations
Idioms

J2EE
Tool Specifics:
- WebSphere
- WebLogic
- Oracle
- DB2
- Struts
- Cactus
- ...

.NET
CORBA
SOAP/SDL/UDDI
ebXML
RosettaNet
BPSS, XLANG, WSFL

...J2EE...

...NET...

SOAP/SDL UDDI/WSFL...

Platform Dependent Generated Implementations
Summary

- Domain models are essential for successful integration
- Refine the models along structure and behavior axes
- Metamodelling is important for integrating information
- Expose legacy systems through business aligned wrappers
- Ideally use MDA tools to generate implementation