Model-Driven Development of a C# RIMBAA API

Hans Jonkers
Philips Research
May 27, 2009
Context

- Philips Research
  - Healthcare
    - HSA
    - ABCD project
ABCD Goal: Clinical IS

To bring the RIGHT INFORMATION at the RIGHT TIME customized to the CLINICIAN in the given CLINICAL CONTEXT to IMPROVE EFFICIENCY and PATIENT OUTCOMES

1. How to bring together clinical data scattered on “information islands”?

2. How to determine what information subset is relevant for a given clinical context?

3. How should this information be presented to the clinical user?
ABCD Data Model

- The HL7 RIM and Clinical Statement pattern were chosen as the data model for representing and storing patient data.

- First implementation uses a hand-coded object model covering a subset of the RIM and Clinical Statement pattern.

- An API for creating, accessing, modifying, serializing and storing/retrieving patient data was generated from the object model.
Scope

- Implementation Technology Specification (wiring format, usually XML)
- Hierarchical Message Description
- Refined Message Information Model
- Domain Message Information Model (ClinicalStatement)
- Reference Information Model

API

Data Types

Vocabulary

RIM

DMIM

RMIM

HMD

ITS
Technology Matrix
Problems

- HL7 Version 3 is big, complex and evolving
- Hard to deal with complexity without appropriate tooling
- Most tools are written in Java (e.g. RIMBAA Reference Implementation) and/or require EDF (Eclipse Development Framework)
- We need a .NET-based solution, but few mature .NET-based tools seem to be around

- Develop .NET-based tool support based on our previous experiences with model-driven development
Goals

Concrete goal:

- Provide a high-level C# API (including implementation) to create, access, modify, store and retrieve HL7 v3 based objects in order to
  - simplify the encoding of clinical data
  - simplify the development of applications that process clinical data
  - be independent of the way the data is stored

The aim is *not* to

- Develop fully-fledged support for the HL7 Version 3 methodology
- Provide support for the messaging part of HL7 Version 3
How?

• Use model-driven development to *generate* the API from the XML and MIF sources of the HL7 standard → makes it ‘easy’ to keep up with changes in the standard

• Keep XML (‘structured ASCII’) and MIF completely under the hood

Sources of the HL7 v3 standard

- XSD
- MIF

Generator

C#

API
Sources of the HL7 Version 3 Standard

Sources of the HL7 v3 standard:
- Microsoft Visio files
- Microsoft Access databases
- Rosetree files
- XML schemas
- MIF files

MIF (Model Interchange Format)
- Defined by HL7 for tool interoperability
- Can be viewed as a metamodel of HL7 v3
- Defined in terms of XML schemas

Mapped by the HL7 V3 Generator to
MOM (Meta Object Model)

• We use the MOM (Meta Object Model) as our language for defining object models
• The MOM Core object model is similar to other metametamodels such as MOF and Ecore, but has a few special features
• Object model = collection of object types + properties, defining ‘is-a’ and ‘has-a’ and ‘part-of’ relations
• All objects are first-class citizens, including types and properties
• Main object types supported by MOM:
  – Object, Class, List, Value, Enum
  – Built-in types such as Boolean, Int32, String (all .NET CLR types)
  – Union types
    • Mathematical unions, allow one-to-one modeling of XML schema choice particles and HL7 v3 choice boxes
    • Support a limited form of multiple inheritance
Vampire Tool Set

Provides (among other things) the following tools:

• VIDE: integrated visual model editor
• Validation tools
• Large collection of generators:
  – XML schema generator
  – SVG generator
  – documentation generator (html)
  – normalizer
  – multiple general-purpose C# code generators (heavy-weight, light-weight)
• Several predefined object models (XML, XHTML, XML Schema, MOM itself, etc.)
Development of the Generator: Step 1

What
• Develop a MOM model + (de)serializers for the full XML Schema language, so we can deal with the MIF definition in MOM terms

How
• Use reflection on the Microsoft XmlSchema classes and some hand optimization to generate XmlSchemaModel.mom + (de)serializers
VIDE Snapshots of XmlSchemaModel.mom

union type

inherits from Schemaltem and Type
Development of the Generator: Step 2

What
• Develop a generator that maps an XML schema to a corresponding MOM object model

How
• Read XML schemas as instances of XmlSchemaModel.
• Implement a model-to-model transformer that maps these instances to instances of the MetaObjectModel.
• Include transformations to raise the level of abstraction.

Notes
• This tool can be used as a general-purpose tool for reverse engineering XML schemas.
• High-level XML schema to object model/UML transformers do not seem to exist (?); Microsoft’s xsd.exe and Altova’s code generation tool were tried first, but generate relatively low-level code.
Simple Example (1)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="hoot" type="Noot">
    <xs:annotation>
      <xs:documentation>The root element</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:complexType name="Aap">
    <xs:sequence>
      <xs:element name="noot" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Noot">
    <xs:complexContent>
      <xs:extension base="Aap">
        <xs:sequence>
          <xs:element name="mies" type="Mies"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="Mies">
    <xs:choice>
      <xs:element name="aap" type="Aap"/>
      <xs:element name="noot" type="Noot"/>
    </xs:choice>
  </xs:complexType>
</xs:schema>
```
Simple Example (2)

Altova representation of schema:

Generated MOM model:
Development of the Generator: Step 3

What
• Generate the MOM object model of MIF from the XML schemas defining the MIF language

How
• Flatten the XML schemas
• Apply the XML schema to MOM generator
VIDE Snapshot of MifModel.mom
Development of the Generator: Step 4

What
• Implement a MIF to MOM deserializer so MIF files can be read as instances of object types defined in MifModel.mom

How
• Extend the XML schema to MOM model generator such that it adds serialization metadata to the generated MOM models
• Implement a generator that maps a file X conforming to a schema S to an instance X’ of the object model S’ generated from S
• Apply it to MIF files

Note
• The first two steps provide a general-purpose extension of the XML schema to MOM model generator
From MIF Files to MOM Instances

Artefact.mif

LoadXml

instance of XmlModel.Document

MifModel + serialization metadata

Generator\textsubscript{4}

Instance of MifModel.Artefact

XmlModel is ‘MOM’s DOM’

Generator\textsubscript{2}

MOM domain
Development of the Generator: Step 5

What
• Develop a generator that maps MIF artefacts, in particular static models, to MOM object models

How
• This requires understanding of the way static models are defined in MIF
• Given this understanding, the generator can be implemented as a straightforward model-to-model transformer

instance X of MifModel.StaticModel \[\rightarrow\] Generator_5 \[\rightarrow\] MOM object model of static model X
Final Step (the easy bit)

What
• Generate the C# API from the generated MOM object models of the RIM and ClinicalStatement using one of the existing code generators

How
• Select a code generator and apply it to the MOM object model for artefact X (X = RIM, ClinicalStatement):