Defining Design Guidelines for XML Messages Used in Enterprise Application Integration

Hans C. Arents
senior IT market analyst

I.T. Works
“Guiding the IT Professional”

Innovation Center, Technologiepark 3, B-9052 Gent (Belgium), Tel: +32 (0)9 241 56 21 - Fax: +32 (0)9 241 56 56
E-mail: hca@itworks.be - Site: http://www.itworks.be/ - Home: http://www.arents.be/
Designing XML messages for EAI

- Situating the problem
  - from inside to outside business units
  - using XML message-based EAI

- What are the challenges?

- What we first started to do

- What we eventually did
  - model at a higher level of abstraction
  - using XML Schema + XML Schema Adjuncts

- The modelling process
  - modelling steps
  - people involved

- Conclusions
Situating the problem

Major Belgian bank faced with two major challenges:
- integrating two different IT infrastructures after a recent merger
- opening up their business units to become e-business units

Integration challenge
- keep using and/or joining up the best parts of both IT infrastructures
  - different programming languages, hardware & operating system platforms
  - biggest problem when exchanging data: *data heterogeneity* (the use of different data representations)
- tendency to buy more COTS financial front-office trading systems
  - need to be integrated with middle- and back-office applications
    - bought from specialized vendors
    - are not built to be integrated
    - heterogeneous in nature

Opening up challenge
Inside business units

- business unit A
- business unit B
- business unit C
- business unit D

n using IT services internally, tightly coupled architecture
Opening up inside business units

loosely coupled

n providing IT services externally, loosely coupled architecture
Turning into outside business units
Turning into outside business units

message bus

business unit A

business unit B

business unit C

business unit D
Finding a solution

- Choosing the appropriate architecture
  - supporting message-based application integration
    - asynchronous (message-oriented middleware) and not synchronous (CORBA/DCOM)
    - allowing a “publish and subscribe” interaction model

- Using XML as the message data syntax
  - for all the usual reasons:
    - platform and protocol neutral
    - flexible & extensible but controllable
      - to adapt to ever-changing internal needs
      - to fit in with unforeseen external requirements
  - but also because the outside world increasingly uses XML

- Finding a XML message modelling methodology
  - XML was beginning to be used everywhere, so urgent need for guidelines
What are the challenges?

n Selecting the appropriate message-oriented middleware
   – capable of reliable handling massive volumes of messages
   – XML-aware or XML ready, with good local technical support
   status: selection of EAI tool is completed + pilot project has started

n Developing a methodology for modelling the XML messages
   – leveraging existing in-house standards, following emerging XML standards
   – dealing with message payload only, not with message handling
     • process/task/transaction control, message flow handled by middleware
   status: business models being defined, message models under development

n Setting up a process for enforcing the use of this methodology
   – set of guidelines on how this methodology has to be applied
   – organisational framework to support the use of this methodology
   status: guidelines are being written + process is tested on pilot project
What we first started to do:

- look at in-house existing work
  - data modelling methodologies (ER, UML, …)
  - conventions (naming, capitalization, abbreviation rules)
- find good design patterns for
  - XML instances
    - large set of examples at http://www.xmlpatterns.com/
  - XML Schemas
    - small set of best practices at http://www.xfront.com/
- try to find best practices in existing (financial) XML standards:
  - FixML, FpML, GovTalk, swiftML
  - other XML standards: ebXML, SOAP, ICE, …
- start to write a set of practical design guidelines
  - for how to write XML instances
  - not for how to design XML schemas
XML message standards considered

- **FixML**
  - real-time electronic exchange of securities transactions
  - rich set of business components for bonds, equities and options
  - clean approach to extending the core tag set of the standard

- **FpML**
  - enable e-commerce activities in the field of financial derivatives
  - rich set of business components for “over-the-counter” derivatives
  - design patterns for the exchange of financial information

- **GovTalk**
  - design guidelines for the XML messages to be used as part of the UK’s e-Government Interoperability Framework
  - guidelines for XML Schema design
  - advice on XML message modelling
XML message standards considered

  - improve interoperability of different financial XML implementations
  - by using the SWIFTStandards methodology:
    - model financial business processes
    - store these business models in a repository
    - use these business models to derive messages
  - main design goals of using UML as modelling tool:
    - to remove any dependency on the message syntax
      - current FIN or ISO 15022 syntax, future XML syntax
    - to have the business model serve as the common denominator
  - consistent way of generating XML DTDs from UML business models
    - e.g. business information is expressed as XML elements/values,
      meta-data information is expressed as XML attributes
    - designed in such a way that migration to XML Schemas remains feasible
Developing a methodology

n Lessons learned:
  – existing in-house approaches:
    • oriented towards database modelling
    • modelling process itself is not a shared process
      – business analysts model + their developers build
      – standards department approves or disapproves
  – XML diversity:
    • hard to find a common set of design patterns
    • XML instances/DTDs are the result of a hidden thought process
  – XML backlash:
    • doubt about the usefulness of “XML overhead”
    • hard-to-get developer buy-in from the different divisions of the bank

n Actions taken:
  – move to a higher level of abstraction
  – focus on the development of a shared process
What we then started to do:

- introduce a distinction between
  
  **business modelling** = modelling of the business processes
  
  **message modelling** = building a library of message components

- business models
  
  - well-documented descriptions of business processes
  
  - no formalisation (yet, but UML is being considered)

- message models
  
  - widely used base message components
  
  - common base datatypes
  
  - documentation

  to be managed in a central repository
  
  - under tight control of the central standards department
  
  - open for consultation by local standards setters & use by local developers
Different types of models

- **Business model**
- **Message model**
- **XML Schema**
  - XML Schema Adjunct
    - message documentation
    - message broker format
  - XML Schema Adjunct
    - Cobol copybook
  - XML Schema Adjunct
    - RDBMS tables
XML Schemas as a universal data modelling syntax:

- carefully chosen set of data modelling concepts and datatypes
- significant synergies with other XML standards
- extensibility and programmability
Why use XML Schema Adjuncts?

- Not a challenge: reliable messaging (is handled by broker)
  - messages should not get lost or be resent unnecessarily
  - message content should be secured against spying/tampering

- The biggest challenge of application integration: *mapping*
  - each participating (legacy) application provides/expects a different data format suited to its own needs
  - mapping between data format and message format requires knowledge about the internals of the application

- Need for mapping meta-data
  information about the mapping between two different data formats
  - not tied to a particular data model
  - explicit and exploitable

  Æ XML Schema Adjuncts
Who is responsible for the mapping?

Integrating a legacy system into the messaging infrastructure
- adapter: mapping legacy data format to message data format
- connector: connecting the legacy system to the message bus

Adapter responsibility of the legacy system maintainers
- legacy system generates message data format
- mapping in legacy system
- existing developers struggle with writing good mapping code

Adapter responsibility of the message bus maintainers
- legacy system prepares (special) legacy data format
- mapping in adapter generated from XML Schema + Adjunct
- new developers focus on adapter
What are XML Schema Adjuncts?

XML Schema Adjunct Framework


- for adding processing-specific information to XML Schemas
  - mappings to relational databases
  - business rules for additional validation
  - indexing parameters for native XML databases
  - parameters used for presentation and input forms
- for associating *adjunct data* with XML Schemas and their instances

Not a W3C standard, but already widely used

- e.g. Schemantix

http://www.schemantix.com/

  - Schemantix Development Platform (SxDP)
    - set of open source tools for building Web interfaces
  - uses XML Schemas + XML Schema Adjuncts to generate complex multi-page HTML input forms with built-in validation
<?xml version="1.0"?>
<schema-adjunct xmlns:inst="instance NS URI"
               xmlns:proc="processor NS URI">
  <instance-namespaces default="instance NS URI"/>
  <global>
    <proc:global-meta-data/>
  </global>
  <type type="anyType" context="XPath/to/match">
    <proc:type-meta-data/>
  </type>
  <element type="anyType" context="XPath/to/match">
    <proc:element-meta-data/>
  </element>
  <attribute type="aSimpleType" context="XPath/to/match">
    <proc:attribute-meta-data/>
  </attribute>
  <!-- more type, element and attribute adjunct associations -->
</schema-adjunct>
<?xml version="1.0"?>
<schema-adjunct xmlns:acc="Account" xmlns:sql="SQLdatabase">
    <instance-namespaces default="Account"/>
    <global>
        <sql:databasename>acc_data_tbl</sql:databasename>
    </global>
    <element context="Account/AccountNumber">
        <sql:column>number</sql:column>
    </element>
    <element context="Account/AccountBalance">
        <sql:column>balance</sql:column>
    </element>
    <attribute context="Account/@ID">
        <sql:column>ID</sql:column>
    </attribute>
</schema-adjunct>
Message modelling principles

Message modelling constructs

- objects with properties participating in relations
- “box-in-a-box” modelling approach:
  - attributes are sub-elements of the object element
  - objects participating in a relation are sub-elements of the relation element
- focus on efficiently shipping data around, not capturing complex structure

Technical design principles

- XML attributes only used for technical meta-data
- mixed content model is never used in elements
- use globally defined complex types for elements

No type hierarchy, but definition re-use

- set of base components which have to be reused
e.g. MyPostalAddress includes PostalAddress
- focus on modelling reusable message components
Modelling objects

n With a single property

n With multiple similar properties

n With different properties
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema" elementFormDefault="qualified">
  <!--Property definitions-->  
  <!--Account number property definition-->  
  <xsd:complexType name="AccountNumber_Prop">
    <xsd:simpleContent>
      <xsd:restriction base="xsd:string">
        <xsd:pattern value="\d\d\d-\d\d\d\d\d\d\d\d\d\d\d\d\d\d"/>
      </xsd:restriction>
    </xsd:simpleContent>
  </xsd:complexType>
  <!--Account balance property definition-->  
  <xsd:complexType name="AccountBalance_Prop">
    <xsd:simpleContent>
      <xsd:restriction base="xsd:integer"/>
    </xsd:simpleContent>
  </xsd:complexType>
  <!--Account object definition-->  
  <xsd:complexType name="Account_Obj">
    <xsd:sequence>
      <xsd:element name="AccountNumber" type="AccountNumber_Prop"/>
      <xsd:element name="AccountBalance" type="AccountBalance_Prop"/>
    </xsd:sequence>
  </xsd:complexType>
  <!--Account -->  
  <xsd:element name="Account" type="Account_Obj"/>
</xsd:schema>
Modelling relations

- Without own properties:

- With own properties:
Modelling roles and variants

- A relation with two different roles:

  ![Diagram of two different roles](image1)

- A relation with two variants for the same object:

  ![Diagram of two variants](image2)
Setting up the modelling process

Finding the right balance between local and central

- local activities:
  - business modelling
    - by business analysts
  - local message modelling
    - either by local standards setter
      (if needed supported by central standards setter)
    - or by a central standards setter delegated to the project

- central activities:
  - building the library of message components
  - monitoring and managing the model evolution

Coping with the planned evolution of the modelling process

- now: in-house standards setting department
- in the future: joint standards setting body ("out-house")

The W3C model is designed for efficient joint standards development
The modelling process

1. analyze requirements for business data
2. map requirements to MM XML Schemas
3. review mapping and proposed extensions
4. extend and customize the message model
5. write recommendation + sample XML messages

WD
PR
CR
extended message model
MR + sample messages
The modelling process

Modeling is about defining *shared* standards — modelling process is inspired by the W3C standardization process

- Working Draft (WD)
  - written by business analysts using business model concepts
- Proposed Recommendation (PR)
  - written by local standards setters, helped by the central standards setters
- Candidate Recommendation (CR)
  - after review of the PR by the central standards setters
  - if necessary, central standards setters modify/complete the message model
- Message Recommendation (MR)
  - used by the local developers and the central developers

Responsibility shared by the local business analysts, the local standards setters and the central standards setters

- lots of meetings & lots of discussions, but central dept. has the last word
Conclusions

- Complete design process for XML messages for EAI
  - business model
  - message model
  - message implementation
  - using a W3C-inspired joint modelling process

- Use of XML Schemas to model messages at a high level
  - platform and technology-neutral data modelling standard
  - “runable data models” that can be used for code-generation

- Use of XML Schema Adjuncts for application-specific details
  - business analysts should focus on message content modelling
  - application developers can focus on message mapping/processing

- Open questions:
  - how to handle large numbers of XML Schemas/Adjuncts?
  - how to version an XML Schema and its Adjuncts?
  - started to evaluate XML Schema repositories