IBM’s Model-Based XML Methods and Techniques

Table of Contents

Notices

This information was developed for products and services offered in Europe. IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785

U.S.A.

IBM grants limited permission to licensees to make hardcopy or other reproductions of any machine-readable documentation, provided that each such reproduction shall carry the IBM copyright notices and that use of the reproduction shall be governed by the terms and conditions specified by IBM in the license agreement. Any reproduction or use beyond the limited permission granted herein shall be a breach of the license agreement and an infringement of the applicable copyrights.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you. This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

The product described in this document and all licensed material available for it are provided by IBM under terms of the IMCL or IMSL agreement or any equivalent agreement between us.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information on softcopy, the photographs and colour illustrations may not appear.

Trademarks and service marks

The following terms are trademarks or service marks of the IBM Corporation in the United States or other countries or both:

- IBM
- Insurance Application Architecture
- IAA

Rational Rose® is a trademark of Rational Software Corporation in the United States or other countries or both.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

About This Book

This document provides guidance on the use of IBM's Model-Based XML architecture (MB-XML).

Who Should Read this Book

This book is intended for people who are using or planning to use the MB-XML architecture as the basis for designing XML messages.

What This Book Contains

This document contains:

- An introduction to MB-XML and the message creation process. This part provides an overview of MB-XML and a summary of its role within a message exchange network.
- XML message design tasks. This section describes the recommended tasks for producing the deliverables for an XML message.

Project considerations. A section discussing some of the factors to be considered in managing a project to produce MB-XML based messages. This section includes discussion about the maintenance and ongoing support of the XML message definitions, and on estimating the time required to perform the message design tasks.

Introduction to the MB-XML architecture and the Message Creation Process

Sources: MB-XML

IBM has worked with standards groups, user groups, and individual customers over a number of years on Model-Based Architectures in many industries. IAA-XML is the direct ancestor of MB-XML. It is a messaging architecture for inter-application communications applicable to any industry. It is also a method that explains how to use this architecture together with a UML model called Interface Design Model (IDM). Different Interface Design Models can be used depending on the subject domain. IAA-XML was developed from the Insurance Industry. IAA-XML together with the first Interface Design Model was published as part of IBM's Insurance Application Architecture in 2001.

MB-XML and Middleware architectures

MB-XML can be used as the standard for designing messages in a middleware architecture.

The participating applications could be, for example, legacy applications, web front-ends, call-center applications, and so on. For each application, the corresponding adapter[1] converts the outgoing messages from the specific format used by the application into commonly understood MB-XML messages. It also converts incoming MB-XML messages into a format understandable by the application

The figure below shows an example of a message hub built around MB-XML messages.

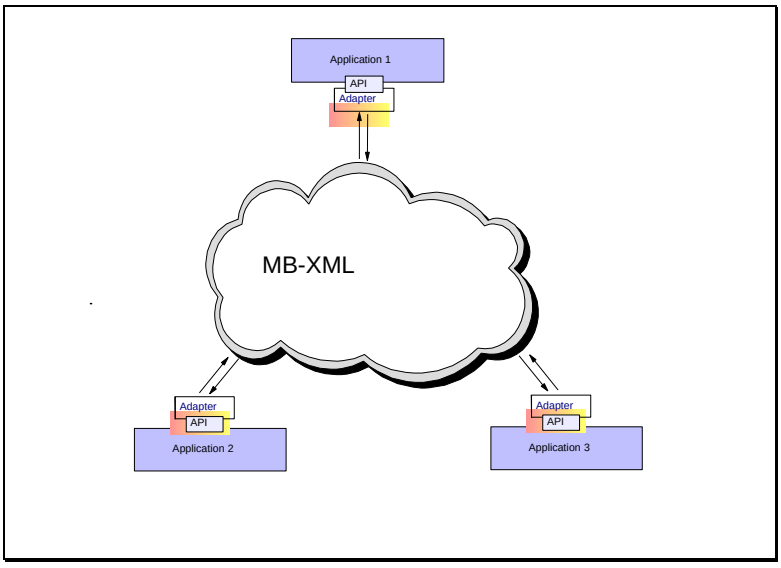

Figure 1. MB-XML Message Hub

[1] An adapter is defined as application code that transforms one message format to another.

Document Type Definition (DTD) vs. XML

The main deliverable of the message design process is a DTD (Document Type Definition). This provides a template that can be used to support the definition of an XML message based on the DTD. The adapters use the DTD to understand the rules for constructing outgoing messages and for understanding the format of incoming messages. The DTD contains the logical structure of the elements within the message, and contains information that can be used for field validation. The adapters contain further rules and logic for performing the full message validation required, i.e., the DTD is limited in terms of how much validation information can be expressed.

The XML message, which is built based on the DTD, depends on the requirements for the actual message being sent. For example, optional data elements defined in the DTD may not necessarily be present in the message.

As an example, below is the DTD of a message element and a sample XML message conforming to this DTD. In the DTD, *type* is defined as an optional property in the *Place* aggregate, while *name* is mandatory: the sample XML that specifies a name but no type conforms to this DTD.

DTD[2]:

```
<!ELEMENT Place  (name , type? )>
<!ELEMENT name  (#PCDATA )>
<!ELEMENT type  (#PCDATA )>
```

Example XML:

```
<Place>
   <name>London</name>
</Place>
```

Recommended Tools

The following tools are recommended to create MB-XML messages:

Rational Rose 2000e is a visual modeling tool that allows you to design object-oriented models in the Unified Modelling Language (UML) formalism. This tool is used as a repository for message definitions (logical view and component view) and additional documentation (sequence diagrams and use-case diagrams) in the use-case view. A Rational Rose script generates the DTDs based on the model content.

XMLSpy 4.0 is a tool that allows you to validate and check the form of DTDs and XML messages. Although it doesn't provide the hierarchical view of the entire DTD, it can be used for message generation. The IDE incorporates a tabular representation to allow XML editor *message generation and DTD generation and validation. You can contact XMLSpy at http://www.xmlspy.com* to download the latest trial version.

VisualDTD is part of the Visual XML Tools suite provided by IBM. This product is not intended for production use but is a graphical tool used to display a top-down hierarchical structure. It also provides a tree structure to view all aggregates and properties within a structure. This tool allows you to generate messages based on the

[2] For details about the DTD and XML notations, you should look at the Annotated XML Spec: http://www.xml.com/axml/axml.html

DTD structure. You can download the latest suite of tools at the IBM Alphaworks web-site: http://www.alphaworks.ibm.com.

Adapter Development

The output of the XML design work provides one of the key inputs to the job of creating the adapters. The deliverables required by the adapter development team are:

- A DTD that defines the data structures that might be part of the XML message.
- A Data Element Definition (DED) list containing the actual data elements that might be found in the message, with a mapping from these elements into the DTD (showing which DTD element supports which data element in the DED).
- A sample XML message showing an example of the actual XML tags which might be sent in a message.

It may be beneficial to provide the adapter developers with a subset of the full DTD(s) for use in their initial adapter development and to test out the basic adapter framework before the full DTD is implemented.

Ideally the adapters should be written in a flexible way so that the amount of hard coding is limited. The majority of the adapter development work is to support legacy / backend system integration.

Overview of the XML Design Process

There are two paths in designing the XML for a new message. The first is the ideal case that can be considered the “fast path”. The other, probably the more typical route, requires more analysis and quite often includes extending the IAA model.

The steps are:

1. Analyze requirements for data and functions
2. Map the requirements to existing DTD(s) (from the Interface Design Model (IDM))
3. Create message sequence diagrams (optional)
4. Review mapping and proposed model extensions
5. Extend and customise the IDM (not required for “fast path”)
6. Execute Rational Rose script to create a new DTD (not required for “fast path”)
7. Create sample XML and other supporting documentation for the message.

The “fast path” can skip steps 5 and 6 when an existing DTD fulfils the needs of the business requirements.

Architecture of an XML Message

MB-XML messages structure information into four layers: message, command, aggregate, and property. Refer to the *Message Architecture* section in the *MB-XML Architecture* document for a more detailed description of these layers.

XML Message Design Tasks

This section contains a description of each task in the XML message design process, with a description of what deliverable is produced by the task, and what roles and skills are required to execute the task.

The seven steps are described in the section Overview of the XML Design Process.

The figure below shows the relationship between the models and main deliverables relevant to the XML message design process. It also depicts the flow of the message design tasks.

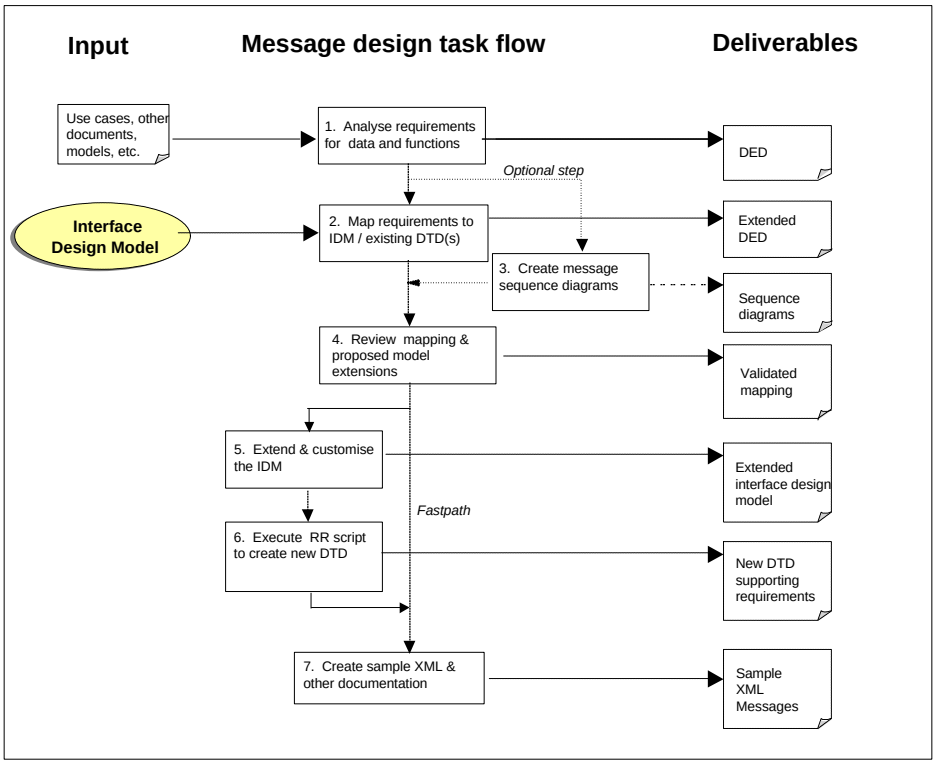

Figure 2. Message Design Task Flow

1. Analyze Requirements for Data and Functions

The first step in developing XML is the generation of a list of data elements. This list contains each piece of data required in the message for that specific set of business requirements. Generally, these requirements are organized in the form of Use Cases (UCs).

Examine Use Cases

In the Use Cases, there are often specific references to specific business constructs. List these references as they are found and add granularity as needed. For instance, if an agent is mentioned as part of the process early in the UC, add agent to the list. As the UC becomes more detailed, e.g., it is mentioned that the agent's phone number and address are required, add these contact points to the agent in the list. Alternatively, there may be no further mention of the agent in the UC; in this instance the reader should question if

the agent belongs in the UC or if there is any information required about the agent that was not mentioned.

Review Other Documentation

Although the UC is the preferred method of communicating business requirements, they have often been recorded by other means. This medium may be in the form of an object model, a relational model, spreadsheets, and so on. The generation of a list of data elements may be easier to create when the requirements are in one of these forms; however, care must be given to ensuring that the business' requirements are still being met. Without the process detail that a UC contains, it is difficult to be sure if the requirements are a fit for the specific situation the XML is being created for.

Deliverable – Data Element Definitions (DED)

The deliverable from this task is a spreadsheet containing a list of all the data elements required in the message. The minimum requirement is to include a data element name with a description of the data element. This can also be a useful vehicle to store other information about the characteristics of the data element, e.g., data-type, valid values, usage (mandatory, optional) and so on.

Role[3]

The person who performs this task should be a business analyst.

2. Map the Requirements to Existing DTD(s)

Once the list is complete, the data elements must be mapped to the model being used for message generation (i.e., IDM, or a customised version of this).

This process may be as simple as finding matching names:

Data Element	DTD Aggregate	DTD Property
Name prefix	PersonName	prefixTitles
Last name	PersonName	lastName
First name	PersonName	firstName

However, there are instances that require combining or breaking up data elements:

Data Element	DTD Aggregate	DTD Property
Business phone number	TelephoneNumber	countryPhoneCode
	TelephoneNumber	areaCode
	TelephoneNumber	localNumber
	TelephoneNumber	extension
	ContactPreference	type = "business"[4]

[3] Role refers to the skill required to perform the described task and produce the deliverable described in this section.

[4] The fact that a particular phone number is used as business phone number as opposed to a home phone number is expressed using the *type* property.

There may be data elements in the list that do not exist in the Interface Design Model. In this instance, the model must be extended. The key word *NEW* is used in the mapping document to reflect this. In the example below, *eyeColor* was needed, but did not exist, so it was added to mapping document as a likely model extension.

Data Element	**DTD Aggregate**	**DTD Property**
Eye color	Person	NEW ->eyeColor

In addition to mapping the data requirements, the functional requirements must be mapped to commands defined in the DTD.

Functional Requirement	DTD Command (request / response pair)	Parameters Request	Parameters Response
Add name address information	AddParty	Party	None
Modify party	ModifyParty	Party	None
Inquiry into address record	InquiryParty	Party or PartySearchCriteria	Party

Deliverable – Extended DED to Include Mapping

The deliverable from this task is an extended version of the DED spreadsheet containing additional columns that explain the mapping of each data element in the DED to the corresponding property in the DTD. Typically, the extension to the DED contains a column for the DTD property name, and a column for the parent aggregate. A fourth column is recommended to define the path through the DTD data structures required to access the aggregate, if this is not obvious.

Template for Mapping Spreadsheet

Data Element	**DTD Aggregate**	**DTD Property**	**DTD Access Path**	**Comment**
Source data element	*aggregate*	*property*	*access path where applicable*	*comment where applicable*

Example of a Partial Mapping Spreadsheet

Data Element Name	***DTD Aggregate***	***DTD Property***	***DTD Access path***
CURRENCY: CURRENCY IDENTIFIER	*PriceList	*currency : Enumeration	*PriceList
DATETIME (effective)	*PriceList	*effectiveFromDate	*PriceList
DATETIME (expiration)	*PriceList	*effectiveToDate	*PriceList
DIVISION: DIVISION IDENTIFIER	*PriceList	*<<attnav>>getDivisionIdentifier() : String	*PriceList

The command mapping can be done in a separate spreadsheet, or as part of the DED spreadsheet.

Role

The person who performs this task should be a business analyst, ideally with some business modeling skills and experience.

---

3. Create Message Sequence Diagrams

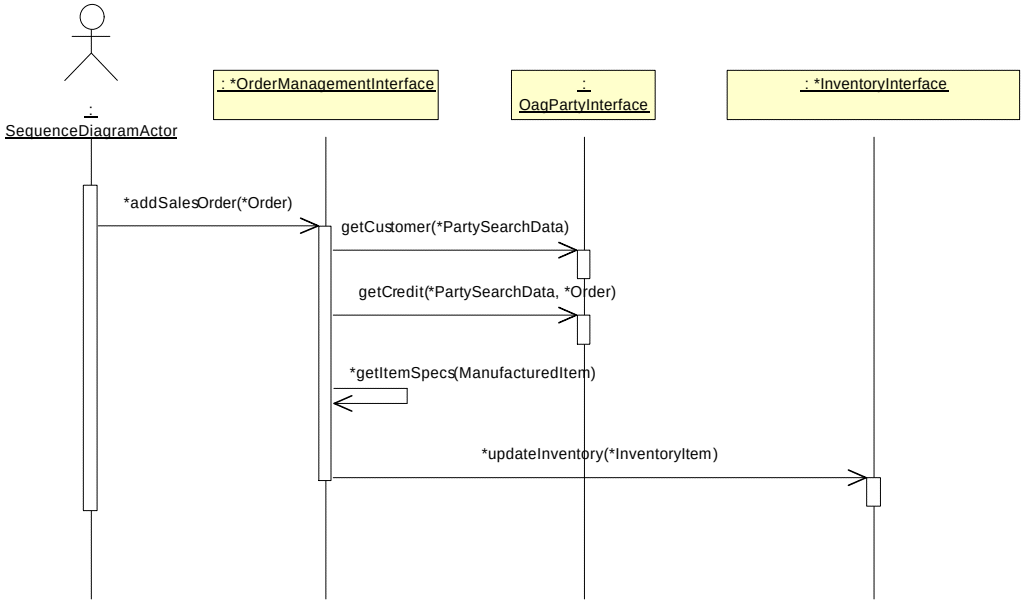

Figure 3. Sequence diagram

It can be beneficial to create sequence diagrams to graphically represent the messages and their use. The diagram shows the commands that an invoking system sends to the defined interfaces. Every arrow represents one of these commands, including the name of the aggregates passed and received as parameters.

Sequence diagrams are created in *Rational Rose*, in the *Use Case View* of the Interface Design Model. The IDM defines operations available on the interfaces of several components. When building sequence diagrams, operations will be used following the command mapping. Where the requirement for a new command has been identified as part of the mapping, a new one is created by customising the model.

The sequence diagram serves as a useful illustration of the interaction between systems, and can be a good basis for understanding and documenting the work-flows supporting the message flows.

Deliverable – Sequence Diagram

The deliverable from this task is a sequence diagram. This is an optional step.

Role

The person who performs this task should be a business analyst, ideally with knowledge of the IDM.

4. Review Mapping and Proposed Model Extensions

Once the mapping document is complete, an expert in the IDM should be consulted to ensure it is correct. If sequence diagrams have been created, they should also be validated at this stage. It is vital not make uncoordinated changes to the model to maintain its usefulness and value.

Deliverable – Validated Mapping

The deliverable from this task are the extended DED and the sequence diagrams from the previous tasks, amended, if required, to take account of the specialist's review input.

Role

The person who performs this task should be a modeling specialist and know the IDM in detail.

5. Extend and Customise the Interface Design Model

One of the results of the previous steps is a list of extensions to apply to the model. Applying these changes requires the knowledge of all the relevant modeling conventions.

Making Additions Directly to the DTD vs. to the Model

There may be times when a previously generated DTD is missing only one or two elements to complete the mapping to the business requirements. At first it appears to be easier to just add these elements manually to the DTD and not the model. There are some important issues that must be considered before using this approach.

The model is the repository for all business requirements and to insure integrity and consistency across all generated messages it must remain current. There is potentially additional maintenance resulting from adding anything to the requirements outside of the model. There may be instances when an element is considered useful to the enterprise and it may be reasonable to leave it out of the model. However, another project may work on the same subject area and need the same element, and if it is not in the model it is quite likely that the project will add it to the model with a different name, or the project may use the same name but the element may have a different meaning.

Creating Additional Attribute (Data Elements) and Classes in the Model

Classes are used to define the aggregate layer; their attributes are used to define the property layer of the messages. For example, a Person class results in a Person aggregate in the DTD, where all attributes of the Person class result in property definitions related to that aggregate. Any relationships the Person class (or any of its super-classes) has to other classes in the model will end up as *Relationship*[5] definition in the DTD.

Properties, aggregates and relationships identified in the mapping spreadsheet as NEW have to be added to the Interface Design Model so they will appear in the DTD.

[5] As explained later, this is only true if the relationship is stereotyped as <<generate>>. If it is not, then the relationship will result in a direct inclusion of the target aggregate in the definition of the source aggregate.

To add an attribute on an existing class in Rational Rose, first locate the class within the *Logical View*, then add the attribute, including its data type. Refer to *Appendix C* in the *MB-XML Architecture* document for a description of valid data types to be used.

When using the data type Enumeration, define the list of permitted values as part of the attribute description, using the following syntax:

eg: *value 1*
eg: value 2

To add a new class, identify the subject area (e.g. Party) where the new class belongs, then allocate the class to the appropriate component. Where applicable, make the new class a sub-class of an existing class in the model. If no existing class would apply as a super-class, make it a subclass of *BusinessModelObject* to ensure it has a *type* attribute and can hold key information.

Check the document *MB-XML Architecture* document, section <cross ref> Use of UML: Modeling conventions for more details on how attributes and relationships are defined in the model.

Avoid deleting attributes, classes or associations in the model. If you want to remove a particular class from the generated XML, exclude it from the scope of the component. If you want to remove a relationship or an attribute from the generated XML, stereotype it <<componentsonly>>.

Creating Additional Operations on Interfaces

The operations defined on the interfaces[6] are used to define the command layer of the MB-XML messages. For example, an addParty() operation on a Party interface results in a twin set of commands: <addPartyRequest> and <addPartyResponse>. In order to create a new command element in the DTD, add a corresponding operation in the interface model.

Scoping a DTD in the Component View of the Model

The Interface Design Model is grouped around subject areas such as *Party*. When working in a component-based environment, it is recommended to retain that structure in the generated DTDs.

To limit the scope of a DTD to a subset of the model, create a component in the component view of Rational Rose and allocate to it all the classes desired in the DTD. The generation is always done for one component; the generated DTD includes aggregates for all the classes assigned to that component, as well as the classes of the *Common* component. The *Common* component includes all the classes that need to be included in every DTD based on MB-XML.

The classes in the selected component and the common classes are generated with their full definition (their attributes and associations). Other classes that are referenced (such as the other end of an association or parameter of an operation) are generated as empty stubs.

For the interfaces selected, it is usual to also include in the component scope all the parameters passed in the interface operations. Otherwise, it means that only references can be passed as parameters and not the aggregate themselves.

[6] An interface is defined as a class with the <<interface>> stereotype.

The commands are only generated for the operations on the interfaces assigned to the component defining the scope of the generation.

Use of UML: Modeling Conventions

In order to be able to generate DTDs from a UML model, some generation directives need to be specified. The UML notion of stereotype is used for this purpose. By using the stereotypes, extensions can be made to the model that allow an attribute, operation, association, or class to be made available or unavailable to the scripts. These stereotypes also allow a modeler to quickly interpret why elements have been added to the model that would not normally be there.

[ ] – Square Brackets

Empty square brackets are used to signify multiplicity. The brackets are placed after the return type on an operation (signifies multiplicity on parameter in response command), or after a parameter passed in an operation (signifies multiplicity on parameter in request command). An example is the operation *findParty (PartySearchCriteria, Party) : Party[]* on the *SpecificPartyInterface*.

When the XML generation script executes, it reads the brackets and adds an asterisk (*) in the appropriate areas of the DTD.

Another situation where empty square brackets are used is to define multiplicity on an attribute in the model. For example, *addressLines : String[]*, an attribute on the *PostalAddress* class in the model, gets translated to *addressLines** in the DTD.

```
<!ELEMENT PostalAddress (…addressLines*…)>
```

All other attributes get translated as single optional properties:

```
<!ELEMENT PostalAddress
    (KeyGroup?,effectiveFromDate?,effectiveToDate?,city?
    ,country?,region?,street?,houseNumber?,addressLines*
    ,postalCode?,boxNumber?,unitNumber?,floorNumber?,bui
    ldingName?,type?,contactInformation?,Place?)>
```

<<attnav>>

This stereotype is used when an attribute already exists in a class but the navigation from where it is defined to where it is required is too complex to express in XML. The goal is to add an additional property element in XML without adding an unnecessary attribute to the model. A new operation stereotyped as <<attnav>> is added to the model and the generation script transforms it into a property element definition in the DTD.

The figure below shows an example of how the ContactPreference of type *Home* is modeled.

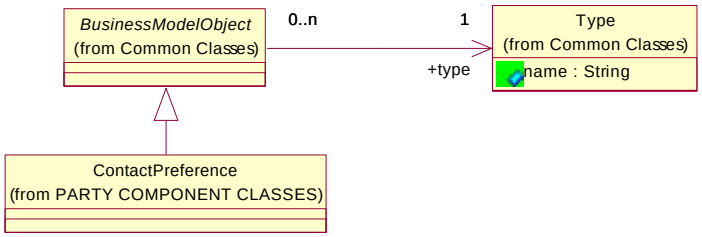

Figure 4. UML Model for ContactPreference

If this model is directly transferred into XML the result is a large structure that adds unnecessary complexity:

```
<ContactPreference>
   <Type>
      <name>Home</name>
   </Type>
</ContactPreference>
```

The figure below shows how the ContactPreference of type *Home* is modeled when <<attnav>> is used on the new type() operation.

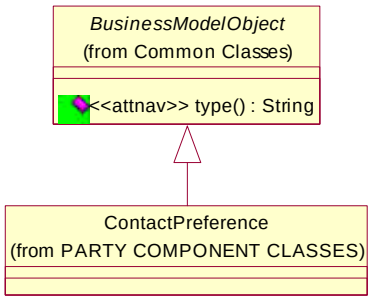

Figure 5. Example of <<attnav>>

The resulting XML is then much simpler:

```
<ContactPreference>
   <type>Home</type>
</ContactPreference>
```

<<typenav>>

This stereotype is used to represent an association that already exists in the model, but with a navigation too complex for XML. A good example of this is the operation getRegisteringAuthority() defined on the Registration class as shown in below.

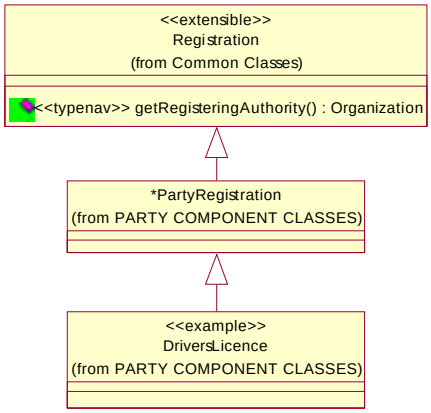

Figure 6. Example of a <<typenav>>

In the model, this is a navigation from *Registration* to *Organization* representing the registering authority of a registration. By stereotyping this operation with <<typenav>>, the model is still able to use it as an operation and the XML generation script generates it as a relationship on all the concrete subtypes of *Registration*.

The definition for DriversLicence for example is the following:

```
<!ELEMENT DriversLicence
    (KeyGroup?,description?,expirationDate?,externalRefe
    rence?,requestDate?,issueDate?,statusDate?,status?,s
    tatusReason?,dateOfDisqualification?,renewalDate?,ty
    pe?,placeOfIssue?,countryOfIssue?,registeredPartyNam
    es*,registeredContactPoints*,registeringAuthority?)>
<!ELEMENT registeringAuthority (Organization?)>
<!ATTLIST registeringAuthority %Relationship;>
```

<<derived>>

This stereotype can be used on attributes and relationships. When using it on an attribute, it designates the attribute as XML only. The scripts used for the IDM ignore it, as should the modelers working with the model. Examples include the derived attribute *smoker* on the *Person* class as shown in the figure below.

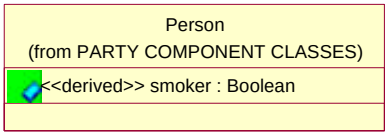

Figure 7. Example of <<derived>> attribute

However, if the attribute is viewed as something that could prove to be valuable to both the XML and the model, add the new XML attribute as an operation on the class with the <<attnav>> stereotype. This way, both can use the element without unnecessarily adding a class attribute to the model.

When using the stereotype <<derived>> on a relationship, it designates the relationship as XML only. This is different from adding an operation with stereotype <<typenav>>

where the relationship itself does not become part of the model, but the operation can still be used by the IDM scripts. In the resulting DTD, the relationship is still generated.

<<generate>>

This stereotype is used on an association when you want to generate an additional level for the association-end.

The next figure shows a class model that would result in an additional <defaultName> element in XML:

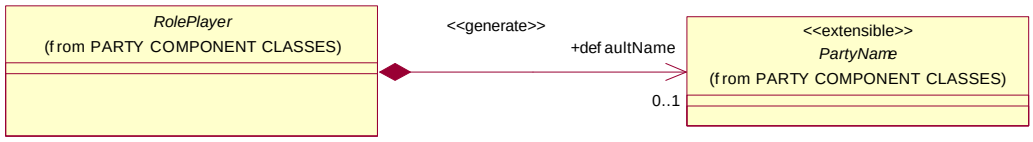

Figure 8. Example of <<generate>> association

The resulting XML with the <<generate>> stereotype then includes defaultName as an element.

```
<Person>
<defaultName>
    <PersonName>
    </PersonName>
</defaultName>
</Person>
```

The XML generated without the <<generate>> stereotype would be more compact.

```
<Person>
    <PersonName>
    </PersonName>
</Person>
```

The <<generate>> stereotype must be used whenever multiple associations exist between the same two classes to ensure a well-defined DTD.

The next figure shows the different relationships that can be created using the stereotype <<generate>>.

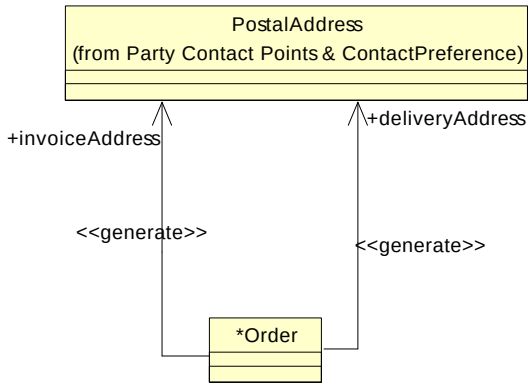

Figure 9. Distinguishing different relationships using stereotype <<generate>>

<<componentsonly>>

This stereotype is used for those modeling elements that have already been expressed in the DTD by one of the various methods listed above. When the Rose script comes across an element in the model with the <<componentsonly>> stereotype, it does not include it in the generated DTD.

Referring back to the <<attnav>> example, the stereotype <<componentsonly>> is added to the class Type because that concept was already expressed with the addition of the new operation in BusinessModelObject.

The stereotype can also be placed on associations, such as the one between BusinessModelObject and Type, and to attributes such as objectId.

The next figure shows an example of these different <<componentsonly>> model elements.

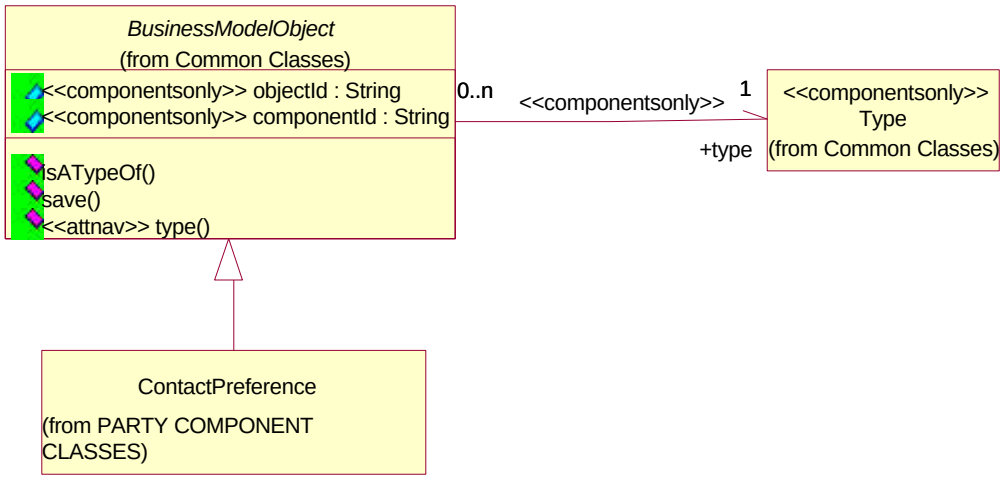

Figure 10. Example of <<componentsonly>>

In general, the strategy for the DTD generation is to make the resulting XML as concrete and specific as possible, whereas the guiding principle in the IDM is to be as generic and flexible as possible. Therefore, most of the generic types and their associations in

the model (for example, Role, ActivityRole, Type) have been marked <<componentsonly>>, and more specific associations (using stereotype <<typenav>> as well as more concrete subtypes have been introduced.

Deliverable – Extended IAA Interface Design Model

The deliverable from this task is an extended and customized version of the Interface Design Model. The IDM is available in the Rational Rose tool, but being UML-conformant, it can be moved into other preferred environments if required. However, note that the supplied script for DTD generation (see next step) is a Rational Rose script, and is therefore dependant on the use of Rational Rose.

Role

The person who performs this task should be a modeling specialist with a good understanding of the MB-XML modelling conventions.

---

6. Execute Rational Rose Script to Create a New DTD

A script has been developed to generate a DTD from a specific component in the Rational Rose model (i.e., the Interface Design Model with any customization). This script executes against definitions in the Component View and adds items necessary for the XML message, e.g., each operation is translated in a Request and Response command, and an operation's return type in the model is the response's parameter in XML. The script also automatically includes the common elements[7] necessary in all components, e.g., Business Model Object.

For more detail, it is recommended to read *Appendix B* in the *MB-XML Architecture* document.

How the Model is transformed into a DTD

When looking at the model (as opposed to the generated DTD), keep the following transformations in mind:

The following translation of CRUD[8]-related operation names to command names occurs:

Interface Design Model	DTD
get	Inquiry
find	Inquiry
create	Add
update	Modify
delete	Delete
remove	Delete

Thus an operation updateParty() in the model results in two commands: ModifyPartyRequest and ModifyPartyResponse.

---

[7] The common elements are defined as all the classes assigned to the *Common* component.

[8] CRUD stands for Create, Read, Update, Delete.

For response commands, the aggregates passed to the corresponding request command will be generated first to support echoing back the request content. Then the actual parameter of the response message is shown:

```
<!ELEMENT InquiryPartyRequest
    ((%Command;),PartySearchCriteria?,%InheritanceParty;
    )>
<!ATTLIST InquiryPartyRequest %CommandRequest;>
<!ELEMENT InquiryPartyResponse ((%Command;),
    PartySearchCriteria?,%InheritanceParty;,%Inheritance
    Party;)>
<!ATTLIST InquiryPartyResponse %CommandResponse;>
```

Only associations stereotyped <<generate>> will be generated as additional layers in the XML structure.

Only operations on interfaces will be generated as commands; operations on classes will be ignored unless they are stereotyped as follows:

- <<attnav>>, in which case an attribute will be generated
- <<typenav>>, in which case the return type will be embedded in the aggregate (treated similar to association in the model)

The concepts of keys, represented as objectId and componentId on BusinessModelObject in the IDM, are represented differently in MB-XML, using the KeyGroup structure. The link between BusinessModelObject and KeyGroup is created by the script.

The concept of an application context is represented with the ApplicationContext class in the IDM. As this information is represented differently in MB-XML, as part of the message header and as attributes on an MB-XML command, any reference to ApplicationContext is removed by the script.

Classes outside the scope of a component, but referenced in an association on a class inside the scope of the component will be generated in a section titled “Empty stubs” in the DTD. No properties or relationships will be generated for these classes.

Deliverable – A New DTD Supporting the Requirements

The deliverable from this task is the DTD that contains the data elements and commands required for the message. The DTD can be reviewed (and modified) with a tool such as XML Spy or Visual DTD.

Role

The person who performs this task should be an IT operations administrator.

7. Create Sample XML and Other Supporting Documentation

Several pieces of documentation support the users of the main message DTD deliverable.

The adapter developers need a sample XML message (or messages) to supplement a DTD as early as possible so that proper testing and understanding can occur. One

message could contain just the mandatory fields and another message could also include additional data.

Other documents that have proved useful for integration projects:

- An explanatory document that gives a brief summary of the nature of the message, and which covers any issues and questions resulting from the message design work. Change control details for the message should also be recorded in this document.
- A data dictionary that lists all the XML properties used by the message set being developed. Each property should have a single generic definition. Additionally, it is also possible, if needed, to produce for each property a cross-reference to any message using the property. The generic definition should ideally come from the IDM element from which the property is generated.
- An XML Message Design Guidelines document. The MB-XML Architecture document contains guidelines for the use of MB-XML. These need to be extended or supplemented to accommodate the usage standards for the actual project. Examples of areas to be dealt with include:
    - Approach to handling currencies and currency exchange
    - Approach for dates and times
    - Standards for aggregate and property naming
    - How to handle permitted values for properties with data type Enumeration
    - When to use referencing vs. embedding

Deliverable – A Sample XML Message and Other Supporting Documentation

The deliverable from this task is:

- An example XML message or set of messages. There are some tools that can aid in the development of these sample messages, but they still require some manual effort.
- Message documentation, as described above:
    - Message explanatory document
    - Message data dictionary
    - Message design guidelines

Role

The person who creates the sample XML message should be a business analyst with an understanding of the business context of the message.

The people responsible for producing the documentation should be the team who were involved in the previous message design steps. Some aspects of the message design guide may require input from the project technical architects and end-to-end designers.

Project Considerations

Maintenance and Management of Messages

Once messages have been created, their management requires some attention. A change control process should be implemented to ensure that message changes are effected in a controlled way. Typically, the following steps might be followed:

1. The business users or designers identify a required change or set of changes to a current message.
2. The impact and timing of the change should be reviewed and agreed with all the relevant parties (business, design, message developers, and so on).
3. The XML message development team should assess which properties and aggregates are affected, and identify which messages will require updating to support the change. The data dictionary will be useful in helping in this task.
4. The change should be made to the various deliverables (e.g., DED, mapping document, customised model, DTDs, explanatory document, sample XML), and new versions of these distributed under version control.

A project may need to accommodate different versions of a message being supported at a given time. For example, it is likely that a restructured message will require adapter modifications, and it may be that different adapters get updated at different times. For this reason, it may be necessary to retain back-level versions of the models that support these older messages. Good repository management techniques for all the message deliverable elements are essential.

Handling Multiple Projects

An additional consideration relates to handling multiple, and essentially independent, projects that are to share the same XML message definitions, for example, in the case of a multinational company whose local companies are each planning to use the same base message definitions across the message middleware.

One approach is to assign totally separate repositories/models to each project. However, greater reuse can be achieved by generating messages from a shared model, which represents an enterprise view of the company.

In doing this, the following factors should be considered:

- Careful model management procedures need to be implemented, so that the customised model represents the combined business inputs from each project. Additional questions may need to be asked to confirm the business meaning of data elements across the participating projects.
- The data dictionary can be used to identify which properties are used by which projects. This is important in the impact analysis of any proposed changes.
- Different levels of sharing across the projects can be achieved depending on the project requirements and overall design:
    - It may be possible for the DTDs to be shared across the projects (i.e., a common modelling view), but to design the messages that use the DTDs as specific to each project (i.e., the DEDs/sample XML messages and so on will be different). The adapters could be customised to support these different usages of the DTD.

- Some messages may be totally shared across projects (i.e., having the same set of data elements and common adapter code to process them), for example: a common message to broadcast new client details.

Project Estimating

The following factors should be taken into consideration when estimating the time required to complete XML message design:

Project Characteristics:

- Is this message design for a new company (new business terminology, standards, etc., will need to be accommodated)?
- Does the new messages design take the analysis into new business areas that have not been dealt with in the project before (e.g., Telecommunications, Government…)
- Are the business requirements well defined and the data elements clearly explained? Poorly defined data elements lead to additional work in clarifying these requirements.

These factors affect the amount of time needed to assimilate the business information (i.e., understand the data elements, do the mappings, extend the model).

Project Team Skills

There will be a learning curve as team members pick up skills and experience with using XML, MB-XML, the tools, and the method. The figures below assume a team skilled with the process and associated tools.

Metrics

- Number of message definitions
- Number of data elements per message definition
- Extent to which data elements across the message definitions are repeated, enabling reuse of an existing property to achieve the mapping. For example, postal address data elements may appear in several messages. Level of reuse is clearly difficult to predict, but recent projects have experienced a reuse factor[9] of between 3 and 7 times for sets or approx 15 messages containing about 400 discrete data elements.
- Number of versions expected before the message definition is approved. This depends on the quality of the review of each version, and the length of time allocated to it. There is an overhead of producing new versions. Given a good review process, approximately 3 versions (including the initial version) should be allowed for.

As a general guideline, the following figures based on previous projects can be used to produce an initial estimate:

Time to create an initial version of a message definition with all of its associated deliverables (excluding sample XML):

- 1.5 days per message definition (simple message – approx 20 data elements)
- 2 days per message definition (medium message – approx 100 data elements)
- 4 days per message definition (complex message – approx 400 data elements)

[9] The reuse factor is the average number of times that a data element is used in a project.

This assumes:

- This is one of about 15 message definitions for the same project
- The “reuse factor” for these message definitions is 5
- No learning curve (skilled with method, tools, and so on.)
- These are the first message definitions for the project, and that there are some new business areas and lines of business to deal with.

To this estimate the following needs to be added:

- Overhead of creating new versions (second and third version assumed): Total 1 day per message definition, assuming 2 new version updates required (i.e., approximately 4 hours per version per message definition, but this depends on the extent of change required in the new version).
- Fixed overheads for the project:
    - Project planning/control and so on. (about 2 hours/week)
    - Create data dictionary (2–3 days)
    - Create message design guide (2–3 days).