XMF Features

Xactium Ltd

September 16, 2004

XMF is a generic executable metamodelling facility for defining and deploying semantically rich (executable) languages. These languages can be tailored to specific domains, or can be widely applicable general purpose languages, for example mapping languages. XMF builds on and extends MDA standards such as MOF, UML, OCL and QVT, thus providing an intuitive and powerful foundation for realising model-driven development.

XMF has been populated with a powerful collection of technologies for language definition (metamodelling) and modelling in general. Some of the core technologies are described below.

Aspects XMF allows existing models to be extended with *aspects*. An aspect is a collection of definitions that are added to an existing model. The aspect is modelled in its own right and contains the new definitions. For example, adding a *toXML* aspect to a collection of classes.

Assembler XMF contains its own machine code definition and assembler. Retaining control in this way allows the machine code to be extended with new instructions and to introduce debugging information in the running object code. Compilers for new languages can compile directly to the XMF assembler. In this way XMF provides a flexible language development system.

Associations Binary associations are a convenient abstraction over attributes. Defining associations involves adding a pair of attributes to classes and an invariant between them.

Clients XMF can connect to external processes using its *clients* interface. XMF can act either as a server or a client and can connect to multiple external processes. The external processes can be managed by creating appropriate internal XMF processes.

Constructors XMF provides flexible mechanisms for creating instances of classes. A class can include the definition of BOA-constructors that define how a class can be used as an operator to create an instance. Keyword constructors can be used to create an instance of a class based on explicit slot initialization.

Compiler XMF implements its own compiler. The compiler is bootstrapped. The compiler can easily be extended or even replaced to support a range of languages.

Constraints XMF provides constraints that are attached to classifiers. A constraint is an invariant that applies to all instances of the classifier. Execution of constraints against a candidate instance is performed under program control and produces a *constraint report* that describes whether the constraints were successful. In the case that a constraint fails, the report provides information that describes why.

Daemons XMF supports *daemons* that are attached to objects in order to monitor slot values. Daemons can be programmed to perform actions when slot values change. XMF provides a number of abstractions based on daemon technology. For example, deamons are the basis of synchronizing multiple objects in a data driven environment.

Data Types XMF provides the following basic data types:

Boolean True or false.

Enumeration Enumerated values.

Float Floating point numbers.

Integer Integer numbers.

Null The undefined value.

Object A value with a classifier and named slots. Most things are objects in XMF.

Operation A value that can be applied to arguments in order to perform an action and return a result. Operations are first class values in XMF and observe lexical scoping.

Sequence Either the empty sequence or a sequence eith a head (arbitrary value) and a tail (a sequence). A sequence has identity so that the head and tail of a sequence may be updated.

Set A set of values; a set has no identity.

String A sequence of characters; a string has no identity.

Symbol A string with identity. Two symbols with the same sequence of characters has the same identity. Typcially, symbols are used for XMF names to make lookup efficient.

Table A hash table.

Vector A fixed length sequence of values.

All other XMF data values are implemented in terms of these basic types. Object is particularly heavily used and is the basis of XMF meta-circularity.

Delivery Edition XMF supports a delivery mode whereby the engine runs executable models developed by a user (possibly using the Developer's Edition) and saved using one of the XMF save formats. Typically the delivery edition will be much smaller than the developer's edition and contain application specific executable models. This mode is suitable for deploying XMF as an embedded system.

Developer's Edition XMF supports a development mode whereby the engine runs Xactium models for constructing and developing languages, tools and models. The developer's edition contains tools for constructing diagrams and for inspecting and editing model elements.

Dialogs The XMF Dialog language provides a declarative way to construct interactive text based dialogs with a user. The language abstracts away from the mechanisms by which the input and output occurs, allowing the developer to focus on the information that must be conveyed by the dialog.

Documentation XMF allows elements to be declared as being self documenting. Documentation can be then entered when elements (such as packages, classes and operations) are defined. The basic XMF system is self documenting. Aspects are defined to translate the documentation to a variety of formats including HTML.

Engine The XMF engine is a virtual machine that executes XMF machine code produced by the XMF compiler and assembler. The engine can be initialised with, and can produce, saved images. A saved image is saved on backing storage and can be reloaded into any XMF engine.

Execution XMF modelling languages are *executable*. Instances of classes can be created and their state can be updated. Execution is achieved by compiling an executable model to XMF assembler or by using the XMF interpreter. Since XMF is a *meta-programming environment* it is easy to construct new executable modelling languages. Execution in a new language is achieved by: translating from the source of the new language to the source of an existing XMF language; or by compiling the new language directly to XMF assembler; or by extending existing XMF executable language constructs; or by writing a new interpreter for the language. The choice of these strategies will depend on how close the new language is to an existign XMF language and whether the new language needs to execute in a highly efficient embedded mode (compiled code being more efficient than interpreted code).

Exceptions XMF provides a Java-like exception handling mechanism. Exception can be thrown in the XMF machine, in system code or in user code. Handlers can be defined to catch any of these exceptions and to take appropriate action.

Flexible Input and Output XMF provides *channels* that are used for input and output. Channels can be attched to strings, files or external

processes. XMF provides a programmable *format* that is used with any output channel; this uses format characters to provide very flexible control over output. Token input channels are provided that recognize sequences of input characters and transform them to data structures such as integers and strings.

Garbage Collection The XMF engine manages its own heap and performs its own garbage collection. XMF applications do not need to manage memory and the XMF garbage collector can be tailored to suit the specific needs of different applications.

Java XMF is written in Java and is therefore highly portable. The basic XMF engine makes very little use of the Java libraries which, depending on the number of XMF packages loaded, provides XMF with a very small footprint.

Language Definition XMF provides packages for language definition. These include the XBNF grammar definition language and Quasi-Quotes that support pattern-based source language mappings. XMF is provided with an extensive executable modelling language that has been defined using these features: XOCL. XOCL (and any other language engineered using XMF) can be seamlessly extended by defining new syntax classes. A syntax class defines a grammar for parsing the concrete syntax of the new construct and provides mechanisms for executing the new construct. Library classes (Sugar etc) make this process very easy for most cases.

Mappings XMF provides mapping technologies. A mapping is a structured definition that is applied to a value to produce a new value. XMF mappings use pattern matching to determine whether and how they should match and transform supplied values. Since XMF is meta-circular mappings may be supplied with values, models or meta-models. XMF mappings are themselves values and are therefore identifiable components that can be constructed, manipulated and saved in their own right.

Meta-Circularity XMF is *meta-circular*. This means that XMF contains a complete definition of itself (or equivalently XMF is written in XMF). There are a variety of benefits from this: XMF is its own test suite; XMF is intended to be a language engineering environment – this claim is validated by its meta-circularity; since XMF is based on an OO model, it is extensible. Everything in XMF is either a simple value, a collection or an object. This uniformity of representation makes writing meta-tools in XMF very easy. For example, XMF provides packages for translating data values to XML; since everything is ultimately one of a handful of data value types, this package can produce XML from any XMF value (even the XML package itself).

Modelling XMF provides a range of basic modelling abstractions. All of the abstractions are defined using the language definition technologies provided by XMF. The abstractions include:

Association A pair of attributes and a constraint between them.

Attribute Defined in a class: defines the name and type of slots in the instances of the class.

Class Defines the structure (attributes), behaviour (operations) and constraints of objects that are created as instances of the class. Classes have parents from which they inherit structure, behaviour and constraints.

Constraint A constraint is a predicate that applies to instances of a given class.

NameSpace A name space is a container of named elements. A package is a name space for packages and classes. A class is a name space for attributes, operations and constraints. A name space can import other name spaces in order to access their contents.

Operation An operation has parameters and an action. An operation is invoked by supplying it with parameter values. Typically an operation is invoked by sending a message to an object.

Package A package is a container for sub-packages and classes.

OCL and XOCL XMF provides a version of the OCL language and extends it with new features to provide XOCL. OCL is the Object Constraint Language of the UML. Within UML it is used to define relational features such as constraints and pre/post-conditions. OCL provides a number of useful abstractions for language engineering such as collections and iteration expressions. OCL is not, however, a programming language. XMF extends OCL with imperative features and integrates it within the XMF meta-circular kernel to provide a powerful programming language (XOCL) that can be used for modelling or for language engineering. XOCL is defined in pure XMF and is therefore an example of a language that can be engineered using XMF technologies. Developers can take all or part of XOCL and use it as the basis of new languages that are tailored to specific application domains. The basic extensions to OCL in XOCL include the following:

Case Case statements are used to dispatch on a value. The clauses of the case statement use pattern matching.

Exception Handling Try/catch-clauses and throw statements are used to manage exceptions.

Find Collections are searched using find statements.

For Collections are traversed using for statements.

Operations Operations are first class data values in XOCL.

Pattern Matching Operation arguments and various language features employ pattern matching.

Side Effects Local variables, object slots, vectors, tables and sequences may be changed by side effect.

TypeCase The type of a value can be interrogated using a type case statement.

Operations XMF provides operations as a first class data structure. Together with other language engineering technologies, first class operations are particularly effective when defining new language abstractions.

Pattern Matching XMF supports pattern matching. Patterns can occur in a variety of supplied language constructs such as *case* statements and operation parameters. Pattern matching allows the developer to focus on information rather than how to navigate to the required information.

Processes The XMF VM XMF supports *multi-threading*. Threads do not block on input and output and are therefore useful for controlling connections to multiple external processes.

Save Formats XMF can save and exchange models in XML, code and binary formats.

Sugar XMF provides technologies that support domain specific languages through the use of *syntactic sugar*. A new language feature is syntactic sugar when its definition can be completely explained in terms of a translation to existing constructs. Sugar is very important when engineering language features that abstract away from the mechanics of *how* the feature is achieved and provides a focus on *what* the feature is providing for the user. Typical examples of sugar provide bespoke definition constructs that meet the users need for expressing orthogonal aspects of a definition.

Synchronisation XMF provides a synchronisation language for capturing rules that determine when two or more element values have changed. When the conditions on these rules are true, arbitrary actions can be called. These can be used to synchronise values across multiple model elements, and are generally useful whenever the state of a particular model element is to be montiored.

Top Level XMF (developer's edition) includes a powerful command interpreter that is used to drive the XMF engine. The top level command interpreter can be used with or without the XMF diagram clients provided by XMF-Mosaic.

Walkers A *walker* is an object that is programmed to pass over a data structure and perform a given task. Walking technology relies on a predefined data format. Unlike other sophistcated modelling environments, XMF provides a simple uniform meta-circular data format. This form makes walking technology very easy to construct and very powerful. For example, a walker can be defined that passes over the entire XMF object repository

and searches for a value matching a given condition (expressed as an XMF operation).

XBNF XBNF is a grammar language provided by XMF. XBNF is like EBNF except that an XBNF grammar is attached to a XMF classifier in order to provide a new language construct that is seamlessly integrated with the XOCL language. The actions of an XBNF grammar are used to synthesize XMF values during the parse. The actions are written in XOCL. All language features in XMF are written in XBNF.

XML XMF provides a sophisticated collection of XML technologies. Declarative rules can be attached tomodel elements that express how the elements are serialized to XML. The serialization technology takes care of sharing by including XML references at specific points in the XML tree. XMF provides XML parsing technology (like XBNF) that allows XML input channels to be parsed and synthesized. Together these technologies make it simple to integrate XMF with other software devlopment tools.