GXL and MOF: a Comparison of XML Applications for Information Interchange

Marcus Alanen, Torbjörn Lundkvist and Ivan Porres

TUCS Turku Centre for Computer Science
Department of Computer Science,
Åbo Akademi University
Lemminkäisenkatu 14, FIN-20520 Turku, Finland
e-mail:{marcus.alanen,torbjorn.lundkvist,ivan.porres}@abo.fi

Abstract. In this paper, we compare the Graph eXchange Language (GXL) and the Meta Object Facility (MOF). GXL and MOF are approaches for information interchange, specifically for the interchange of artifacts created during software development. Although there are several traits in common, some differences can also be found, in particular the more static structure of MOF as compared with the more dynamic nature of GXL. We discuss the benefits and drawbacks of these differences. Additionally we discuss common issues and possible future extensions.
Keywords: Object graphs, Information interchange, MOF, GXL, XMI, XML

1 Introduction

Todays fast-paced technological advancements require a more streamlined way to represent and manipulate information. Our current way of managing e.g. software projects includes several kinds of different information: requirements, specification, timetables, personnel resources, actual source code, test reports, et cetera. All these artifacts relate to each other, but are usually described and manipulated using different data formats and tools.

In this article, we compare two XML [31] applications which have been created to model and interchange data about software and software development artifacts. The Graph eXchange Language (GXL) [36, 35] is a standard exchange format for graphs by Richard C. Holt, Andy Schürr, Susan Elliott Sim, Andreas Winter et al. [11] with the backing of several research communities. GXL is used to describe arbitrary graphs, but additionally it can be used to define GXL schemas which constrain the graphs so that only specific kinds of graphs can be built.

The Meta Object Facility (MOF) [17] from the Object Management Group [15] is a framework for describing metamodels. These metamodel can be used to create models. Metamodels can also be seen as models, with MOF as their metamodel. Serialization is done using the XML Metadata Interchange (XMI) [19, 24] format, which is an XML application. In this paper, we concentrate on the older and significantly simpler MOF version 1.4 instead of the relatively new and complex version 2.0 [20]. The arguments regarding MOF remain mostly the same in any case, although we are aware that version 2.0 includes some interesting enhancements.

As can be seen, both standards employ a way to describe languages (GXL schemas and MOF metamodels) as well as instantiations of these languages (GXL graphs described using the GXL Document Type Defi nition (DTD) and MOF models described using XMI). We have extensive practical experience in using MOF-based modeling technologies and XMI [1, 2, 5], but lack practical experience of GXL. However, as both standards aim to provide a way to describe interconnected parts of information and thus are quite closely related, we believe we are in a position to compare them.

As we are interested in modeling information, we claim that any standard with suffi cient expressiveness for representing information is meta-circular [3, 4], i.e. it should be possible to use the standard to represent itself. In the case of MOF 2.0 [20], MOF 2.0 is used to describe UML 2.0 [21], which in turn is used to describe MOF 2.0, which amounts to the same thing: MOF is used to describe itself. Furthermore, the modeled information conforms to some kind of meta-information model that describes more strictly how pieces of information may be connected. This conformance is represented using meta layers in the modeling community, but similar structure can be found in GXL. Thus MOF is the meta-metamodel, models described using MOF are metamodels (e.g. UML) and fi nally the creations of the user are models. In GXL, the terminology is different: the GXL metaschema is similar to a meta-metamodel, GXL schemas are similar to metamodels and GXL graphs are similar to models. So, in a way, this article is a comparison of GXL the DTD together with GXL the metaschema and MOF together with XMI.

We proceed as follows. In Section 2 we give an overview of GXL and MOF. We also look at some practical aspects and the current usage of these standards, such as diagram support, transformation technologies, extensibility, et cetera. As both GXL graphs and MOF models metamodels are XML applications, we also discuss their respective serializations. In Section 3 we summarize the presentation by discussing common issues and differences between the two standards. In Section 4 we present some related work and ideas for future work. We fi nally conclude in Section 5.

2 An Overview of GXL and MOF

For the purposes of this article, we claim that the structure of information can be expressed as graphs. However, these graphs may have constraints on how their nodes and edges can be interconnected.

The benefi t is that graph theory has a very strong and well-understood mathematical foundation. In general, a graph consists of two kinds of *elements*: *edges* and *nodes*. These elements are *typed*, *attributed* and *hierarchical*. The type of an element determines its classifi cation. All elements of the same type can be seen as having some structural or semantic commonality. Attributes are key-value pairs of primitive type such as strings which describe the element further. Allowing an element to include other elements (or other graphs) into itself supports hierarchical graphs.

An edge connects n elements together. If $n > 2$ the edge is said to be a *hyperedge*. An edge can also be *directed* which splits the n connections into two nonempty sets, one considered the *source* collection of elements, and the other the *target* collection. Very often edges may only connect to nodes, not to other edges. Edges have additional properties which describe the *ownership* between the source node and the target node. Edges are often grouped into specific categories depending on which kinds of nodes they interconnect and what the semantics of the edge is. These groups can then be considered *ordered* or *unordered* and the *multiplicity* of the edges is of importance. If several edges between the same source and target node is allowed, the group can be considered a *bag* instead of a *set*.

2.1 GXL

Structure A GXL Node supports directly the properties defined previously for nodes in a graph. GXL supports ownership hierarchies by inclusion of other subgraphs, which contain other nodes and edges. GXL supports binary edges as a special Edge element, and hyperedges with a Relation element. All elements can be attributed via the Attribute element, and all elements support an optional type via the hasType connection to the Type element. The type is defined in a GXL *schema*, which will be discussed later. The GXL graph model arrangement can be seen in Figure 1.

The GXL graph model establishes few restrictions on what graphs can be created and is thereby a very general solution. This can also be seen in its history, where GXL was created by merging properties from several graph formats such as the GRAph eXchange format (GraX) [8], Tuple Attribute Language (TA), and the graph format of the PROGRES graph rewriting system. The only small drawback of such a general solution is that in order to establish more constrained graphs it must be possible to define these constraints in some language. These languages are called *schemas* in GXL. If we also want tools to support generic manipulation of information all of these languages must adhere to some common (meta)language, which is called the GXL *metaschema*. The beauty of GXL is that it describes the schemas and the metaschema as GXL documents. This means that a GXL information processing tool only needs the GXL DTD to load and save GXL graphs, schemas and the metaschema. This arrangement can be seen in Figure 2. Elements in the schemas can be used as types. However, this also means that there is an extra layer of indirection/understanding that tools must perceive, not just the XML document itself. So even though the tool can load arbitrary GXL documents, it must understand the relationship between metaschemas, schemas and vanilla GXL graphs. Failure to accomplish this means that graph modification or querying might not be feasible.

The graph part of the metaschema is depicted in Figure 3. An inheritance hierarchy of the GraphElementClass metaelement with *multiple inheritance* can be created with the GraphElementClass.isA relation. Also some metaelements can be declared *abstract* with the GraphElementClass.isAbstract property. Subgraphs can be created with the hasAsComponentGraph property. These are identified by a name, and have a lower and upper *multiplicity constraint* which tells how many subgraphs of the given name must exist for instances of the metaelement. The order of the subgraphs can also be specified as important with the relatesTo.isOrdered property.

Edges can be of three types: compositions, aggregations and "plain associations". Also edges have lower and upper multiplicity constraints and can be directed or undirected; both the source and target collection can be ordered or unordered, meaning that order is considered important and must be preserved by any input/output routines and

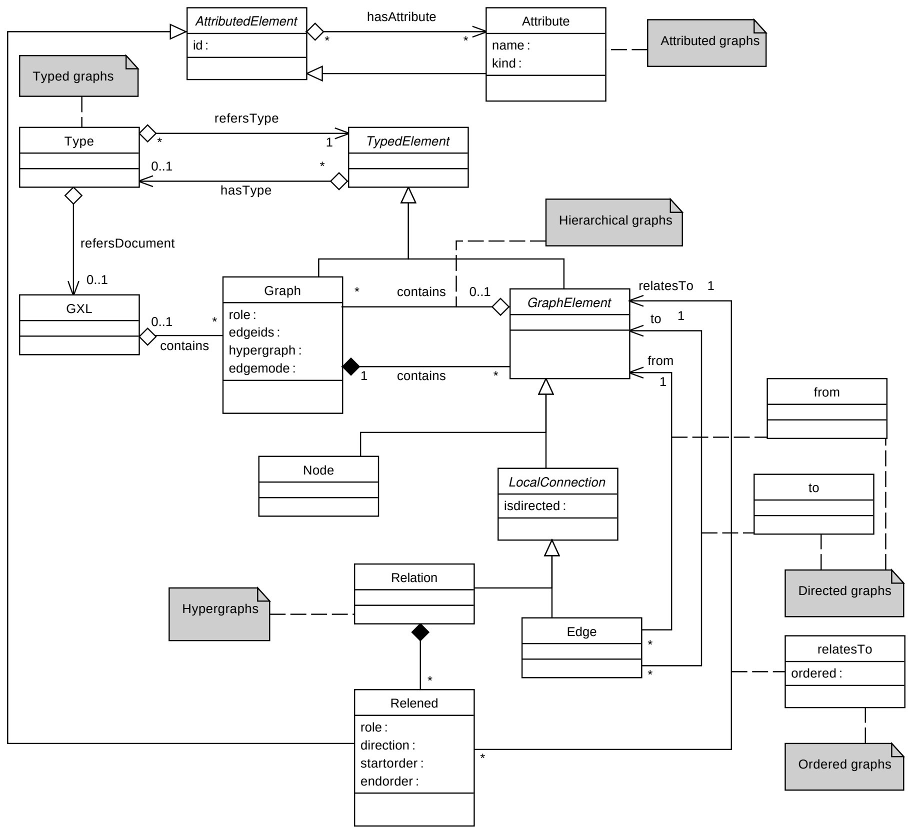

Fig. 1. The GXL Graph Model that defines the GXL DTD. All GXL artifacts correspond to this.

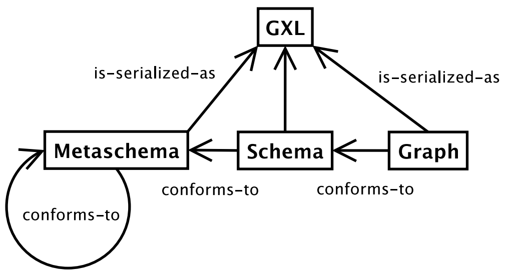

Fig. 2. Overview of GXL and its artifacts. Note how there is only one static serialization format.

must be taken into account by query or transformation algorithms. The edges represent an ownership hierarchy at the graph level, whereas (sub)graph containment represents an ownership hierarchy at the metaschema level and can be used to split schemas into separate “packages” (the subgraphs).

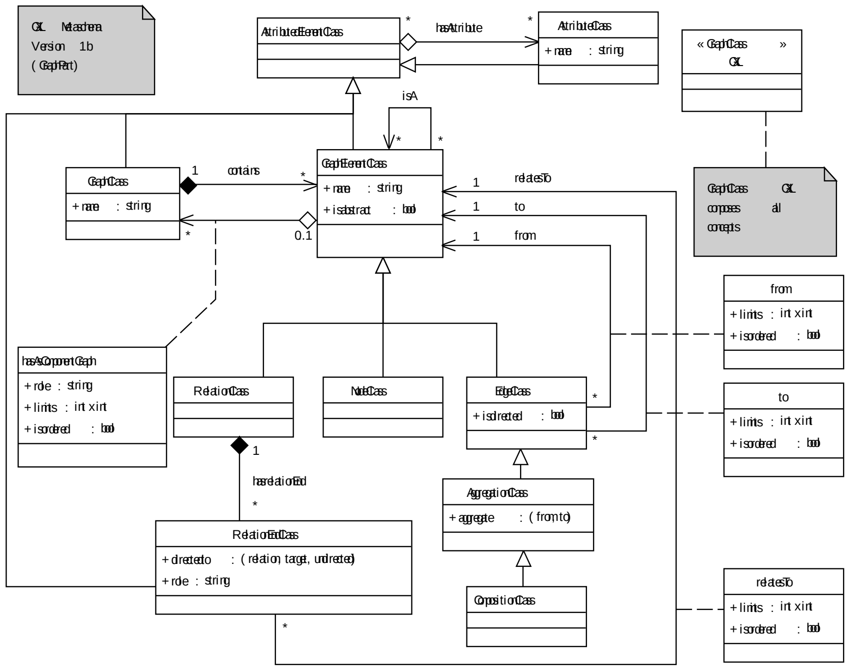

Fig. 3. Part of the GXL Metaschema. Instances of the metaschema define additional restrictions on GXL graphs.

However, the metaschema cannot describe more complicated constraints. This has the benefi t that the theory for representing and validating graphs remains fairly simple, although practical considerations might dictate a need for arbitrary constraints. For example, in the defi nition of the UML metamodel, additional constraints have been heavily used, and thus it does not sound realistic to ignore such a constraint facility.

Element Identification For practical purposes of serialization, elements in a graph may possess an *identity*. In GXL this identity is described with the AttributedElement.id property and is a string unique to the XML document. This is correctly marked as an XML identifi er in the GXL DTD, although in the long run the xml:id candidate recommendation [33] might be adopted when or if it is standardized by the World Wide Web Consortium. However, a globally unique name is important because we want to reference elements from other GXL (or XML) files and a simple local identifier or name is not suitable for that. A more opaque globally unique identifier is necessary.

Schema Identification In order for tools to understand a GXL graph more thoroughly, it is important to be able to identify what schema is being used, i.e. what types are available to GXL elements. The schemas are usually defined in a separate GXL file and shared among all the GXL graphs of that type. Linking to a schema is done using the native facilities of XML, i.e., XLinks [32]. XLink allows the use of Uniform Resource Identifiers (URIs) which can be used to uniquely identify a document e.g. on the WWW. This allows a GXL graph to explicitly reference a specific schema, and additionally it allows tools to download the schema from the location specified by the URI. This means that generic tools can be extended on-the-fly with new schemas.

Visual Representation GXL itself does not define a mechanism for presenting a graph visually on-screen, although this can be remedied in two ways. The simple solution is to define attributes that describe the position, size, form et cetera of a GXL Graph-Element. The more complicated solution is to define a whole new schema for describing the visual representation, thereby decoupling the abstract syntax (the graph) from the concrete syntax (the presentation). This idea is similar to what is already being done by the OMG in the form of the Diagram Interchange (DI) [23] standard and has the benefit that the representation can be split into several possibly different diagrams, each showing a subset of the abstract graph.

Transformation GXL graphs can be transformed with the Graph Transformation eXchange Language (GTXL) [27], although at this moment a revision of GTXL seems to be under way by Leen Lambers [13]. Unfortunately, we do not have experience with GTXL yet and cannot comment on its viability. On the other hand, graph transformations have been extensively researched and we believe it should be possible to adapt any transformation technology using graphs from one underlying schema to another with few problems.

Extensibility GXL allows arbitrary embedding of extra non-GXL information into any GXL node. This has the disadvantage that tools must be ready to process the non-GXL information somehow, either by simply ignoring (and remembering) it or removing it.

Current Support and Licensing Current support of GXL seems to be very good. There are several researchers and companies listed as supporters or contributors on the GXL website [11]. Several tools include export or import capabilities of GXL, such as the round-trip UML software engineering tool Fujaba [14] or the graph transformation toolset GROOVE [26].

Overall, there is activity in the GXL community. GXL is licensed without any fees or restrictions.

2.2 MOF

The Meta Object Facility takes a slightly different approach to modeling than GXL. In MOF, the developer must first define a language (a metamodel) that can be used in creating the actual model (i.e. the actual information). One of the possible metamodels that can be defined in MOF is MOF itself, thereby closing the meta-circularity.

Structure A part of the MOF meta-metamodel can be seen in Figure 4. We have restricted ourselves to the parts that mainly describe the structure of metamodels. As a simple starting point for comparing MOF models to a graph, we may say that the nodes in a graph are mainly Class metaelements, and that edges are represented by Association metaelements.

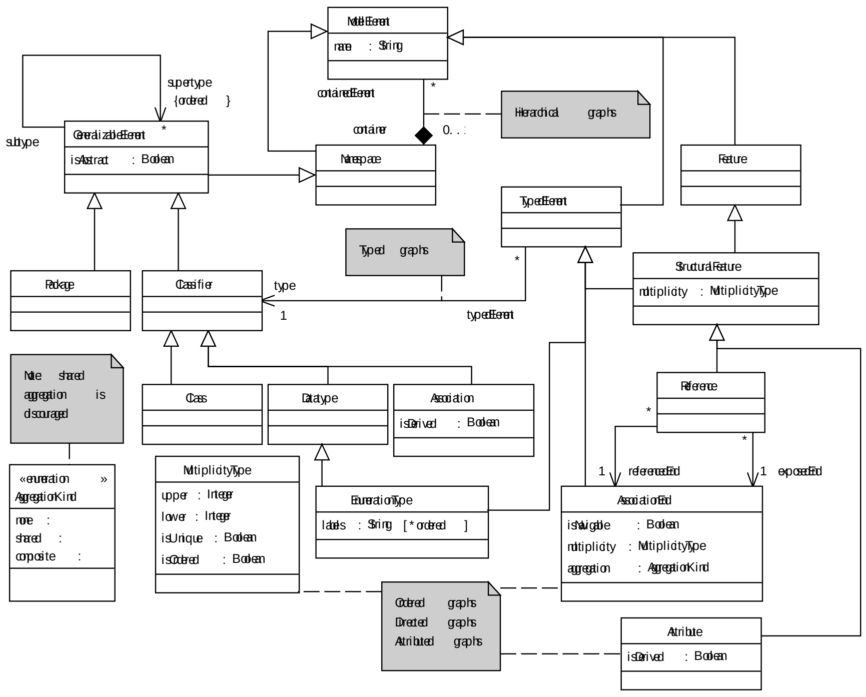

Fig. 4. Part of the MOF meta-metamodel.

It can be understood that a metaelement can establish ownership by two means. One, a metaelement can have Attribute metaelements via the Namespace.containedElement connection. These parts have an obligatory type via TypedElement.type and a MultiplicityType that states the minimum and maximum amount of, as well as possible ordering and uniqueness of, elements. Two, a metamodel can have Association metaelements which each contain exactly two AssociationEnds. These, almost similar to the Attribute, establish a link between two metaelements, but each AssociationEnd can be explicitly set navigable (which supports directed graphs) and three different kinds of aggregation, along with the usual support from MultiplicityType. However, an Association can be and usually is bidirectional, meaning that if a source element is connected to some target element via their slots, that target element is also connected to the source element.

The three different kinds of aggregation are the same as in GXL: plain, aggregate and composite. However, using aggregation (shared composition) is discouraged and it has been removed in MOF 2.0. The reason might be that if one ignores the plain associations, the resulting ownership structure in the form of composite connections form a tree, which has been found to be a very useful structure and which directly maps to XMI. Aggregation, resulting in an ownership structure of directed acyclic graphs, is not as common, although it can certainly be useful.

So to summarize, an ownership hierarchy of metaelements is established via the Namespace.containedElement property, and as in GXL, it can be used to split metamodels into "packages". An ownership hierarchy of elements is established by Associations with one AssociationEnd marked as composite.

Reference metaelements are owned by Classes and are used to track which AssociationEnds are connected to them. This seems a bit redundant, as the Classes could reference some of the AssociationEnds directly. Thus other more light-weight metametamodel approaches have been created, e.g. the Eclipse EMF [9] or our own Simple Metamodel Description (SMD) language in Coral [1].

Since MOF employs a two-step process whereby the user fi rst creates a metamodel, which then allows them to create models, the resulting usage and serialization of those models (in XMI) is very different from GXL. This is depicted in Figure 5 and shows that tools require metamodel-specifi c XMI importers/exporters. In other words, to be able to load a UML 1.4 model from an XMI document, the tools must know how to acquire the UML 1.4 metamodel defi nition fi rst, otherwise it is unable to load it correctly. This is a big contrast with GXL-compliant tools. The GXL tools may be able to load the graph with an unknown schema, although they cannot process it much further.

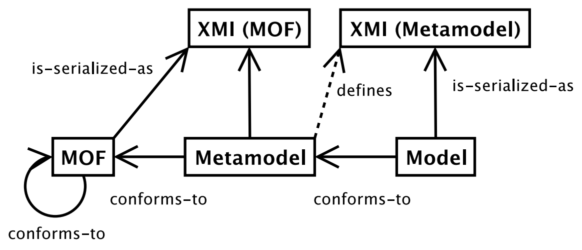

Fig. 5. Overview of MOF and its artifacts. Note how there are several serialization formats on top of XMI. One is the static serialization format, XMI[MOF], and every metamodel defines its own serialization format.

Constraint support in MOF can be assessed as excellent due to the Object Constraint Language (OCL) [16, 22], an addition to MOF. OCL enables a metamodel developer to add arbitrary constraints to the users' models, thus enforcing very sophisticated constraints between elements. A tool can then check these constraints and report nonwell-formedness.

Element Identification Element identification in MOF is handled by XMI with its **xmi.id** and **xmi.uuid** XML attributes. They have been properly defined in XMI and the UUID specification [6] and we have extensively discussed this in [2]. To summarize, elements can be locally identified in an XML document as well as globally with a UUID string, enabling rigid inter-file element identification. On the other hand, current support by XMI exporters/importers is brittle.

Language Identification Similarly to GXL, it is important to detect which metamodel is being used in a model. XMI allows using several metamodels in the same document, and in the new XMI 2.0 standard the XML namespace [30] declaration string describes which language is being used where. This usage is nicely aligned with advances in XML by the World Wide Web Consortium. The only issue is that "there is no requirement or expectation by the XML Namespace specification that the logical URI be resolved or dereferenced during processing of XML documents. [24]" This implies that a tool cannot in general be able to even load a model without knowing the metamodel in advance, because it cannot acquire the metamodel.

Visual Representation MOF does not define a visual representation for models. The basic premise is that there is a strong separation of abstract models containing the semantic data and the diagram which merely display the artifacts on-screen. Thus, the Diagram Interchange (DI) standard [23] has been developed. DI has been successfully used in the Poseidon tool [10] and our Coral tool. This has also been discussed in [2] and the conclusion is that DI is a viable standard that can be used to represent diagram models.

Transformation Even though it is conceivable that several different transformation technologies are used for model transformations, the Query-View-Transform (QVT) [18] is a standard pushed by the OMG to enable the transformation of MOF-based models. As the standard itself is relatively new, we feel it is too early to discuss its benefits or drawbacks.

Other transformation technologies have been described by several authors, for example UMLX [34], YATL [25] and VIATRA [29].

Extensibility MOF metamodels can not as such be extended, but both metaelements and elements can be tagged with arbitrary information using the XMI.Extension XML node. A whole XMI file can be tagged with the XMI.Extensions XML node.

Current Support and Licensing Current support for MOF is low. The meta-metamodel itself has some nonintuitive quirks and is quite big and complex, which presumably has lead e.g the Eclipse team to create EMF. We have also avoided using MOF due to these reasons and opted to explore what fundamental parts are really required in a meta-metamodel. Additionally, MOF 2.0 has become even more complex than it predecessor.

Ironically, the low support for MOF will perhaps not matter, as the serialization is not dependent on MOF per se, but on the metamodels created in MOF. For example, even though our Coral tool is not based on MOF it still is compatible with e.g. the UML 1.4 XMI serialization format.

MOF is released under a royalty-free license.

3 Common Issues and Differences

Comparing Sections 2.1 and 2.2, we can discern several common issues and differences between GXL and MOF. It must be stated that MOF has the backing of an industry consortium which has enabled MOF and related technologies to evolve at a high pace. Examples of these technologies are OCL, DI and QVT, not to mention the flagship metamodel UML, although there is perhaps an ever-increasing fear of a "design-by-committee" syndrome, where a standard reflects only few needs of its users. GXL is more of a community-driven effort where individuals create what they need.

On the metamodel/schema level, both standards have their positive and negative points. MOF has quite a complicated way to describe metaelement interconnections. It even has a second way to establish them, in the form of Attributes, even though an Attribute is basically equivalent to a unidirectional, composite Association. GXL on the other hand does not have inherent support for bidirectional edges (MOF Associations).

GXL contains a crude tagging mechanism in the form of (GXL) Attributes with key-value string pairs (although these do nest). We assume that this concept is included due to the roots of GXL being in describing graphs, which often use attributes for tagging nodes with arbitrary data. Its benefi ts are not clear for information modeling, especially since a composite edge would mostly serve the same purpose. This is somewhat similar to the Attribute/Association issue in MOF. We feel that perhaps XML itself should employ a standard way to tag elements with extra data.

For modeling information, the choice of having a separate Graph metaelement for nesting is unusual and the benefi t is not very clear. A GraphElement could transitively own other GraphElements, without apparent loss of expressivity. This simplifi es the GXL graph model and metaschema since it reduces the amount of concepts it must defi ne.

Support for aggregation in MOF has been dropped, which means that there are some information systems that are awkward to describe in MOF. Indeed we have ourselves developed metamodels where aggregation would have been benefi cial. Support for aggregation in GXL among multiple fi les is not without problems, though. For example, it is not clear which fi le contains the shared subtree of nodes.

GXL has support for hypergraphs whereas the Associations of MOF are restricted to binary edges. We have not found this to be much of an issue when creating metamodels, but it is worth researching further. N-ary relations are used in the database community for entity-relationship diagrams.

Perhaps the largest differences between MOF and GXL are found in serialization, constraint handling and node interconnections. GXL has only one serialization format, the GXL DTD, which serializes graphs, schemas and metaschemas. This has its advantages, but does require yet one GXL-specific validator for validating the schema-graph relationship. MOF on the other hand defines a serialization format for each metamodel. On one hand there is no extra level of indirection involved, but on the other hand there are multiple serialization formats. So, we do agree with Winter et al that the "XMI/MOF approach requires different types of documents for representing schema and instance graphs" (p. 8 of [36]) and that this indeed is a serious drawback, but only because finding the definition of a previously unknown metamodel is impossible, as has been described in Section 2.2. If the metamodel is known, generic XMI reader and writer routines can be created: e.g. Coral supports reading XMI 1.x and 2.0 as well as writing XMI 1.2 and 2.0 in around 5000 lines of C++ code.

Furthermore, Winter et al claim that "XMI/MOF offers a general, but very verbose format for exchanging UML class diagrams as XML streams (p. 8 of [36])." Our opinion is that the format is verbose for two reasons: bidirectionality, which GXL lacks as such, and named slots, which GXL also lacks. If both of these were added to GXL, it would be just as (or even more) verbose as XMI/MOF. And although XMI requires to use names for all interconnections, we find this to be a great advantage. For example, a class can own a set of attributes and a set of operations in two different slots. The serialization of these elements are cleanly separated into their own XML nodes. Also, navigation via named slots simplifies the manipulation and query of models.

Where MOF (or, perhaps, OMG technologies) really outperforms GXL is in its handling of constraints using OCL. OCL has become well-established in the modeling community and allows additional arbitrary wellformedness constraints to be added to metamodels and models. Naturally, this does not prevent a constraint language to be added to GXL, but the point here is pragmatic: OCL exists currently and is in wide use, whereas we do not know of a similar effort based on GXL.

4 Related and Future Work

Modeling and metamodeling platforms are becoming more of a commodity all the time. A high-level view of the current situation is presented by Harald Kühn and Marion Murzek in [12]. Interoperability between metamodeling platforms is becoming more important all the time. We would thus want to find all the necessary concepts for modeling information. Failure to support a concept directly or by means of a lossless transformation to supported concepts means that transformation of data from (or perhaps even to) that platform is not possible.

Similar views on general-purpose meta-metamodels can be found in e.g. [3] and [28]. In contrast with the meta-circular definition, the work of Thomas Baar avoids the meta-circularity with a set-theoretical framework [4] to describe abstract the syntax of languages.

We plan to research further on this topic, trying to cover other meta-metamodels and other information systems. Examples of such are the Eclipse EMF, the XMF/XCore system from Xactium [7] and our SMD. Our aim is to extract the fundamentals in modeling information from these frameworks. Even though it is not necessary to create a meta-meta-metamodeling language, one emerges as a side-effect from a generic modeling platform, as can be seen in Figure 6. A suffi ciently expressive meta-metamodel can be used to create metamodels from the other meta-metamodels, and transformation technology means that all of this ought to be transparent to the end user. The generic meta-metamodel might even be one of the existing meta-metamodels.

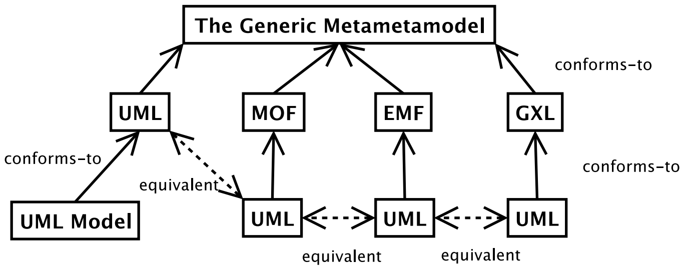

Fig. 6. Overview of how a very expressive meta-metamodel can model all the different meta-metamodels. These can be used to model metamodels, which can be used to create models. The different UML metamodels are equivalent in model expressivity, although the models might be manipulated in different ways since the metamodels are defined by different meta-metamodels.

5 Conclusions

We have presented an overview of two different solutions that can be used to describe information as graphs with nodes interconnected by edges. The Graph eXchange Language has its roots in graph theory and describes every metalevel using the same kind of XML document conforming to the GXL DTD. Additionally graphs must conform to their respective schema which conform to the GXL metaschema, establishing the three meta-layers that is so prevalent in such systems.

The Meta Object Facility has a slightly different approach, by being mainly oriented towards creating metamodels. Using these metamodels, models can be created. Serialization is handled by the XML Metadata Interchange standard. This has the drawback that every metamodel has a different serialization format.

At this point we hesitate to give any judgment on which standard would be more suitable for information interchange. Rather, as can be seen in advances in MOF 2.0, new ways to establish relationships between elements (such as subsets and unions) can and are invented. Therefore there is no perfect solution, but tools and technologies must be able to adapt between different standards. We believe that this is possible by creating and maintaining a meta-metamodel which encompasses all concepts from the different meta-metamodels and metaschemas that exist today.

References

1. Marcus Alanen and Ivan Porres. Coral: A Metamodel Kernel for Transformation Engines. In D. H. Akerhurst, editor, *Proceedings of the Second European Workshop on Model Driven Architecture (EWMDA)*, number 17, pages 165–170, Canterbury, Kent CT2 7NF, UK, Sep 2004. University of Kent.
2. Marcus Alanen and Ivan Porres. Model Interchange Using OMG Standards. In *Proceedings of the 31st Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, September 2005. Special MDE session. To appear.
3. José Álvarez, Andy Evans, and Paul Sammut. MML and the Metamodel Architecture. In Jon Whittle, editor, *WTUML: Workshop on Transformation in UML 2001*, April 2001.
4. Thomas Baar. Metamodels without metacircularities. *L'Objet*, 9(4):95–114, 2003.
5. Ralph Back, Dag Björklund, Johan Lilius, Luka Milovanov, and Ivan Porres. A Workbench to Experiment on New Model Engineering Applications. In Perdita Stevens, Jon Whittle, and Grady Booch, editors, *UML 2003 - The Unified Modeling Language*, volume 2863 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, 2003.
6. CAE Specification. DCE 1.1: Remote Procedure Call, 1997. Available at `http://www.opengroup.org/onlinepubs/9629399/toc.htm`.
7. Tony Clark, Andy Evans, Paul Sammut, and James Willans. *Applied Metamodelling: A Foundation for Language-Driven Development*. 2005. Available at http://www.xactium.com/.
8. J. Ebert, B. Kullbach, and A. Winter. Grax: Graph exchange format. In *Workshop on Standard Exchange Formats (WoSEF) at (ICSE'00)*, 2000.
9. EMF Development team. Eclipse Modeling Framework. www.eclipse.org/emf.
10. Gentleware. The Poseidon for UML product. `http://www.gentleware.com/`.
11. Graph Exchange Language website. `http://www.gupro.de/GXL/`.
12. Harald Kühn and Marion Murzek. Interoperability Issues in Metamodelling Platforms. In *Proceedings of the First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA 2005)*, February 2005.
13. Leen Lambers. A new version of GTXL: An Exchange Format for Graph Transformation Systems. In *Workshop on Graph-Based Tools (GraBaTs) 2004 at Second International Conference on Graph Transformation (ICGT 2004)*, October 2004.
14. Ulrich A. Nickel, Jörg Niere, and Albert Zündorf. Tool demonstration: The FUJABA environment. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE)*, pages 742–745. ACM Press, 2000.
15. Object Management Group. `http://www.omg.org/`.
16. OMG. Object Constraint Language Specification, version 1.1, September 1997. Available at `http://www.omg.org/`.
17. OMG. Meta Object Facility, version 1.4, April 2002. Document formal/2002-04-03, available at `http://www.omg.org/`.
18. OMG. MOF 2.0 Query / Views / Transformations RFP. OMG Document ad/02-04-10. Available at www.omg.org, 2002.
19. OMG. XML Metadata Interchange (XMI) Specification, version 1.2, January 2002. Available at `http://www.omg.org/`.
20. OMG. MOF 2.0 Core Final Adopted Specification, October 2003. Document ptc/03-10-04, available at `http://www.omg.org/`.

21. OMG. UML 2.0 Infrastructure Specification, September 2003. Document ptc/03-09-15, available at `http://www.omg.org/`.
22. OMG. UML 2.0 OCL Specification, Ocober 2003. OMG document ptc/03-10-14, available at `http://www.omg.org/`.
23. OMG. Unified Modeling Language: Diagram Interchange version 2.0, July 2003. OMG document ptc/03-07-03. Available at `http://www.omg.org`.
24. OMG. XML Metadata Interchange (XMI) Specification, version 2.0, May 2003. Available at `http://www.omg.org/`.
25. Octavian Patrascoiu. YATL:Yet Another Transformation Language. In *Proceedings of the 1st European MDA Workshop, MDA-IA*, pages 83–90. University of Twente, the Nederlands, January 2004.
26. Arend Rensink. The GROOVE Simulator: A Tool for State Space Generation. In J. Pfalz, M. Nagl, and B. Böhlen, editors, *Applications of Graph Transformations with Industrial Relevance (AGTIVE)*, volume 3062 of *Lecture Notes in Computer Science*, pages 479–485. Springer-Verlag, 2004.
27. G. Taentzer. Towards Common Exchange Formats for Graphs and Graph Transformation Systems. *Electronic Notes in Theoretical Computer Science*, 44(4), 2001.
28. Dániel Varró and András Pataricza. Metamodeling Mathematics: A Precise and Visual Framework for Describing Semantics Domains of UML Models. In *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*, pages 18–33, London, UK, 2002. Springer-Verlag.
29. Dániel Varró, Gergely Varró, and András Pataricza. Designing the automatic transformation of visual languages. *Science of Computer Programming*, 44(2):205–227, August 2002.
30. W3C. Namespaces in XML, January 1999. Available at `http://www.w3.org/`.
31. W3C. Extensible Markup Language (XML) 1.0 (Second Edition), October 2000. Available at `http://www.w3.org/`.
32. W3C. XML Linking Language (XLink) Version 1.0, June 2001. Available at `http://www.w3.org/TR/xlink/`.
33. W3C. xml:id Version 1.0 W3C Candidate Recommendation 8 February 2005, February 2005. Available at `http://www.w3.org/`.
34. Edward D. Willink. Umlx: A graphical transformation language for mda. In Arend Rensink, editor, *CTIT Technical Report TR-CTIT-03-27*, pages 13–24, Enschede, The Netherlands, June 2003. University of Twente.
35. Andreas Winter. Exchanging Graphs with GXL. Technical Report 9–2001, Universität Koblenz-Landau, Institut für Informatik, Rheinau 1, D-56075 Koblenz, 2001.
36. Andreas Winter, Bernt Kullbach, and Volker Riediger. An Overview of the GXL Graph Exchange Language. In *Revised Lectures on Software Visualization, International Seminar*, pages 324–336, London, UK, 2002. Springer-Verlag.