Håvard D. Jørgensen, Frank Lillehagen, Dag Karlsen

Collaborative Modelling and Metamodelling with the Enterprise Knowledge Architecture

This paper presents the Modelling Platform for Collaborative Enterprises (MPCE) currently being developed in the ATHENA project. The platform enables interoperability between enterprises by providing an environment where most aspects of the collaboration can be negotiated and described as enterprise models and metamodels. It also facilitates business interaction by executing the models. We here introduce the metamodelling framework of the MPCE, known as the Enterprise Knowledge Architecture (EKA). The EKA can represent models on any meta-level in a uniform way. It departs from the conventional ordered meta-levels of software engineering. Instead it treats models as a constellation of mutually reflective, partial views, e.g. different views from different companies. Currently five tools are exchanging models through this framework, and we plan to submit it for standardization.

1 Introduction

The ATHENA project [Athe04] is developing a modelling platform for cross-enterprise collaboration (MPCE) [SLJ+05]. This paper describes the modelling framework of this platform. The primary objective of this work is to establish an Enterprise Knowledge Architecture (EKA) for full enterprise model exchange between

- Different companies
- Different disciplines, functions and roles
- Different modelling tools, languages, paradigms and metamodelling architectures

The core of the MPCE is a model repository with its content stored as EKA structures in XML format. This paper describes the underlying rationale and design of the EKA. Other interoperability research topics in ATHENA include cross-organizational business processes, ontologies, service-oriented architectures, and model-driven architectures. While these areas mainly deal with technical interoperability between software tools, our work on enterprise modelling emphasizes communication between people. Our approach reflects the need to see metamodelling as an integral part of modelling, and to support concurrent modelling, metamodelling, and model execution in order to facilitate the processes of negotiating and maintaining an unfolding, socially constructed shared understanding between companies.

The second main objective of our research is to extend the metamodelling capabilities of our Metis tool. Metis is a generic enterprise modelling tool with metamodelling capabilities so that customers can adapt it to their local needs, preferences, and terminologies. The tool was originally developed around 1990 to support product design and engineering, but in recent years it has been leading in the Enterprise Architecture and IT Governance markets.

Section 2 describes the background of this paper, outlining the current metamodelling capabilities of Metis. Then a number of requirements for metamodelling and model interoperability are outlined in section 3. Section 4 describes our solution, while section 5 outlines the implementation and usage experience. The second part of the validation briefly compares our approach to related standards such as MOF, OWL and RDF, before section 7 points out directions for further work.

2 Background: Metis

Enterprise Architecture (EA) models cover many aspects of the enterprise, connecting business strategy to the operational level business processes,

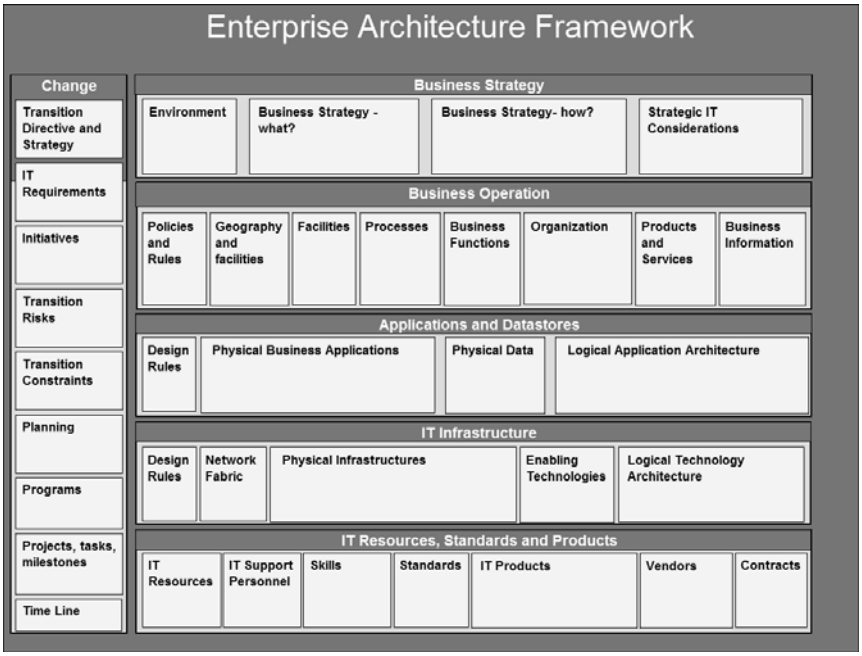

Figure 1. Example enterprise architecture framework

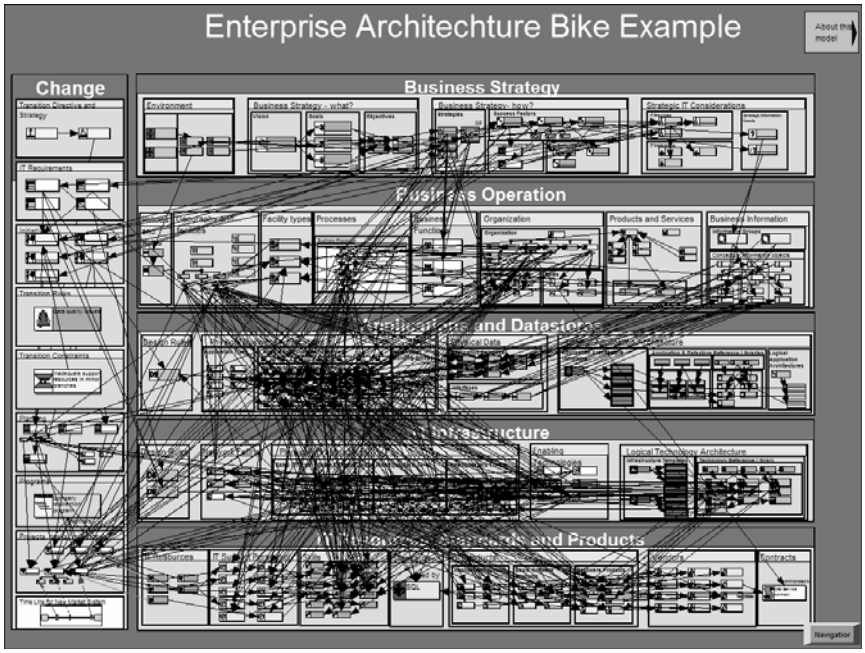

Figure 2. Enterprise architecture example, with relationships connecting business strategies and operations to the IT architecture

organizational structures, and IT infrastructures (cf. Figure 1 and Figure 2). For large enterprises, complete EA models can consist of hundreds of thousands of objects and relationships stored in the Metis Enterprise repository. In order to work with such large models, powerful dynamic queries, multiple views, and sub-model management are critical capabilities.

Most Metis customers utilize the metamodelling capabilities of the tool to define their own modelling *types*. Models consist of *objects* and *relationships*, and objects may have *interfaces* (roles).

Relationships are binary, and go from one object or relationship to another. Relationship type definitions control which type of elements may be connected, with cardinality constraints. *Part-of* decompositions are supported natively, and may be visualized as tree structures or nested (components inside their parent). In the definition of an object type, you also define which types of parts it can have. Since Metis handles deep nesting, allowing you to zoom in to the next level repeatedly within the same view, the modelling area can be thought of as an infinite 2½ dimensional space.

Objects and relationships have *properties* and behavioral features such as *methods* and *criteria* (queries) that may be invoked during analysis. Methods may also be used for implementing rules ensuring that one property's value is derived from other properties on related objects or relationships. Property *value types* such as enumerations may also be defined.

Model elements are instances of one and only one type, but they may represent both concrete individuals as well as generic classes. Users define new types in metamodelling forms. An example metamodelling form is shown in Figure 3.

2.1 Enterprise Architecture Methodology

As an indication of how important metamodelling is to Metis customers, we here briefly discuss its place in the EA development methodology.

1. EA models are typically constructed to answer some business questions, such as
   - How can we optimize our application and service portfolios?
   - What are the effects if this application becomes unavailable?
2. After a business question has been proposed and prioritized for inclusion in the EA, we identify the sources that can bring this data into the EA
   - Existing databases, spreadsheets or XML data can be automatically imported through the Metis Collection Framework.
   - Some information may have to be collected manually from people in the company.
3. Once you know where to find the information, you need to decide how it should be represented in the EA model
   - In what framework (Zachman, DoDAF, etc.)?
   - Using which modelling types?

Sometimes, the 3rd step will apply some of the hundreds of modelling types already available. More often, however, existing types have to be extended with e.g. new properties or behavior in order to be able to fully answer the business question.

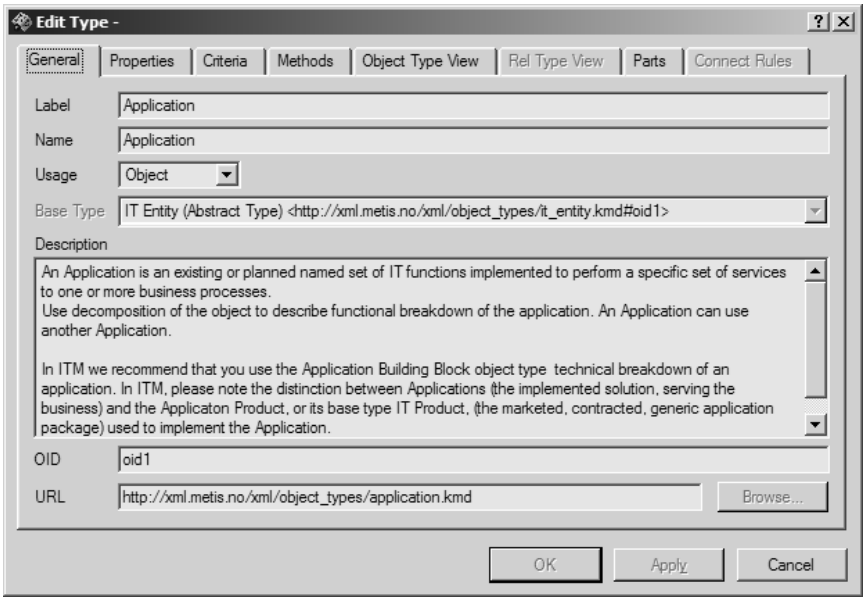

Figure 3. Metis form for type definition.

2.2 Visualization

In order to manage complexity, each model can consist of multiple *model views* (diagrams). New views can be constructed dynamically, e.g. from queries performed on the underlying model repository. This viewing capability is crucial for answering business questions, which are formulated as model queries. Their result sets can be shown in newly generated model views.

Each object or relationship may thus have multiple visual representations, (even within one model view). *Symbols* may be selected individually for each view, but are typically inherited from the type's *view-style*. *Macros* are used for controlling the visual elements (such as colors and texts) by property values. This is widely applied, e.g. for indicating the state of an object. A typical encoding would use grey or green for objects that are in an ok state, and red for objects that need attention. Macros may also be defined locally for each view instance, e.g. to use a picture to represent an object. In the symbol editor view of Metis, users can draw their own object and relationship symbols.

2.3 Templates and Metamodels

Related object, relationship, property, method, criteria and visualization types are commonly grouped together in *metamodels*. Often metamodels reflect a particular domain, such as "Organization structure" or "High-level business processes". Metamodels can be nested (include other metamodels) in a hierarchy. A *template* is a starting point for creating new models. In addition to a set of metamodels, it may include some model elements, e.g. to establish a framework for modelling such as that shown in Figure 1. With Metis we offer a range of standard templates, such as UML, Metis Enterprise Architecture Framework, IT Management, Business Process Modelling, Capital Asset Planning with Business Cases, DoDAF, XML and Database Import Configuration etc. Many customers and consulting partners have also built their own metamodels and templates.

In addition to basic metamodelling, Metis supports a number of metamodel and model evolution management services, such as:

- Changing the type of an existing object or relationship (drag-and-drop)
- Changing the symbol of an object or relationship view (drag-and-drop)
- Extend, restrict or replace the set of metamodels available for a particular model
- Locally override a metamodel by replacing some of its types.

Changes such as these may create inconsistencies in the models. Sometimes this will cause data loss, e.g. if you change the type of an object to one that does not allow all its current properties. Other inconsistencies, such as violation of relationship connection rules or part rules, are tolerated. The *validation* function in Metis checks your model and flags all such inconsistencies. It also proposes default resolution operations, e.g. relocating wrongly connected parts.

3 Requirements

This section outlines requirements for a platform that enables interoperability between modelling tools and facilitates shared understanding to be negotiated between people from difference companies and backgrounds. Requirements will be numbered (R1, R2 etc.) as they are introduced.

3.1 Multiple Modelling Dimensions

Enterprises have several dimensions. In order to illustrate this, let us look at a product component, such as an electronic circuit board. It has a process lifecycle (design, manufacturing, maintenance, recycling phases); it is handled by organizational roles and responsibilities, which require knowledge and skills, and use systems, data, and software services. The product also has a product decomposition structure and variant hierarchy, a timeline (expected lifespan etc.), physical and spatial properties (size, weight etc.), money parameters (cost, pricing etc.), and there will be decisions that control it (e.g. select among alternative designs). All of these dimensions, and potentially many more, may be represented in an enterprise model.

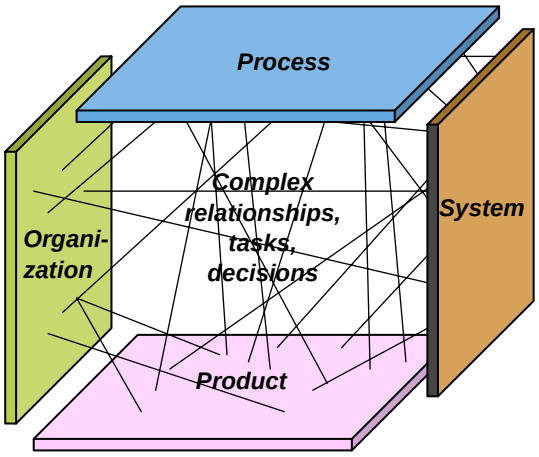

Figure 4. Multiple interrelated model dimensions.

Figure 4 above shows four such views, for the design of business processes, organizational structures, product portfolios and software systems, respectively. The ATHENA enterprise modelling core is called POP* because it is designed to capture processes (P), organizations (O), products (P) and any other (*) relevant enterprise dimension. The EKA should be able to represent these dimensions and utilize them to organize the modelling structures in a manageable way (R1).

3.2 Language Definition

Language definition and extension, and subsequent metamodel organization and management, is an important model management challenge. New language constructs can be defined by [Styh02]

- *Disjunction* (R2), where new constructs are defined as specializations of existing constructs in a top-down tree structure. This is the common object-oriented software approach.
- *Conjunction* (R3), where new constructs are defined by composition of existing constructs, involving multiple inheritance, combining aspects.

In one view, we may for instance define "Project" as a specialization of "Process", e.g. stating that all projects are unique, composite processes with a clear objective. In another view, we may define "Project" as a temporary "Organizational Unit" or as a kind of "Budget item". Within each view, we thus need the conventional approach of language extension by disjunction, while across views, conjunction is needed. Studies have argued that most enterprise concepts should have a conjunctive definition [Styh02].

3.3 Model Interoperability through Mutually Reflective Views

Multiple views are at the core of interoperability problems. Different individuals, groups and companies apply modelling tools with different capabilities, languages and meta-languages, in order to describe partially overlapping aspects of their joint and separate enterprises. These differences create interoperability problems on many levels (cf. Figure 5).

Interoperability resolution will often benefit from investigating multiple meta-levels together. For instance, if we are trying to establish whether object a in view A and object b in view B refer to equivalent or overlapping concepts, we will benefit from assessing which constructs in the languages of A and B that they are instances of.

The existence of a standardized or common framework that the companies agree to use will make it easier to achieve interoperability. The standard format can be defined on the data encoding layer (e.g. XML Schema), as a common language (e.g. UML or POP*), and/or as a meta-language (MOF, RDF, OWL, or EKA). However, when a common language has been selected, both companies face the task of relating their views of data and models to the common framework. It is also known that a common language does not guarantee interoperability, e.g. because there are detailed semantics (one meta-level up) that are left implicit.

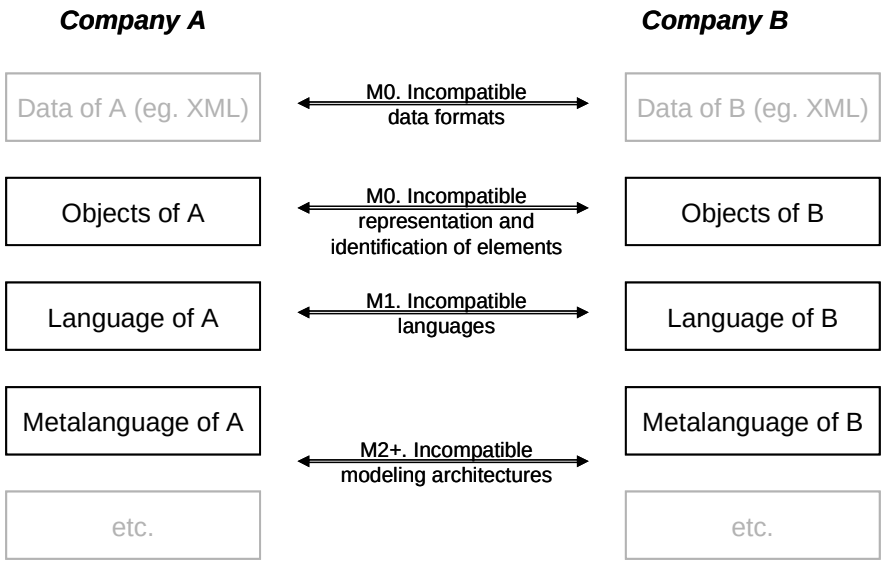

Figure 5. Interoperability problems on different levels.

The identity, precision and granularity with which objects are represented in a model view built with a common language, also depends on interpretation and pragmatic decisions by each party, and may thus cause interoperability problems even if a common language has been defined. Consequently, interoperability requires assessment of multiple meta-levels (R4) also in the presence of standards.

In general we cannot assume that different companies, groups or individuals use compatible concepts on any level [Kang99]. A modelling approach to resolving interoperability problems should thus not assume a single integrated objective model or metamodel. Instead it should facilitate the bringing together of any set of views. The process of establishing relationships between views, in order to achieve interoperability, consists of a set of translation and resolution tasks and decisions (cf. Figure 4). When integrated and related in this way, each view will deepen the interpretation of the other views; they will thus be *mutually reflective* (R5).

In order to grasp these processes, we have examined literature on social negotiation of meaning between people and groups from different backgrounds [BeLu66; Weng98]. There we found that ambiguity is a prerequisite for establishing shared meaning, because even before sufficiently shared understanding is established, we need some common terms in order to communicate. Ambiguous terms allow us to speak the same language even though we don't interpret the terms exactly the same way. Formal languages, seeking to remove interoperability problems by providing precise semantics, prohibit the articulation of ambiguous concepts, and thus provide no support for negotiation of meaning. They are thus suitable for reflecting a closed world where no interoperability problems exist. They can document the outcome of interoperability through negotiation, but not support the negotiation process. For interoperability support, we need an approach that tolerates ambiguity and uncertainty (R6), but which can also express shared understanding precisely when it is achieved (R7).

3.4 Supporting Modelling by Business Users

Another key challenge is to enable business users to perform modelling, to assist view resolution, interpretation and execution. This requires frameworks that are simple and intuitive (R8). In order to address multi-level interoperability problems and to be able to construct customized and role-specific views (R9), metamodelling should be an integrated part of modelling, not just a specialist activity based on other tools and concepts then modelling. Type definition or selection should be at the control of the user, but constrained according to authorization and modelling phase (R10).

3.5 Expressiveness

A metamodelling framework must be able to represent several kinds of language elements. A framework that is to facilitate interoperability between several tools with different underlying metametamodels should ideally include every kind of element found in any tool. This expressiveness requirements must however be balanced with simplicity and usability (R8). For the tools we have studied, these features are necessary:

- Object, property and relationship as first class citizens that may possess properties and have relationships between them (R11),
- Instance evolution, that an element may change type during its lifecycle (R12),
- Metamodelling on instance, class, and meta-class levels (extending R4).

Among the features that are not well supported in current metamodelling frameworks, we find refining, specializing, decomposing and relating property and relationship structures, other than as a side effect of defining object class structures. As briefly discussed above, weaving together aspect specifications from different dimensions is another key challenge. Aspects are also important for interoperability because they enable a more clear specification of what (aspects) two elements have in common, and in which aspects they differ.

3.6 Summary of Requirements

R1	Multiple modelling dimensions/aspects
R2	Language extension by specialization
R3	Language extension by conjunction
R4	Metamodelling on multiple meta-layers
R5	Mutually reflective views
R6	Ambiguity and uncertainty
R7	Precision and formality
R8	Simple and intuitive languages
R9	Multiple views for roles, tasks etc.
R10	User-defined languages
R11	Expressiveness
R12	Instance evolution

4 The ATHENA MPCE Platform

The core of the MPCE architecture is an enterprise modelling *repository* with its content stored as enterprise knowledge architecture (EKA) views (discussed below). The platform also offers generic modelling support services (e.g. transformation and validation), administration services (e.g. access control), model management services (e.g. versioning and configuration management), and knowledge management services, which include meta-modelling. It also provides standardized interfaces to different modelling tools, execution services (e.g. messaging, process, and rule engines), and integration services for plugging web services and applications into model execution.

Figure 6 below shows the architecture of the first implementation of the MPCE. At the top, we find five modelling tools all accessing the common repository through its web services. The repository is implemented on top of the Metis Team server. In addition to this server we have implemented EKA services to assist with model merging. Metis Team also provides a web browser interface that allows users to upload and download files. Enterprise models and metamodels are stored as files in the EKA XML format. This XML format and the web service interface are open and defined independently of the underlying technology.

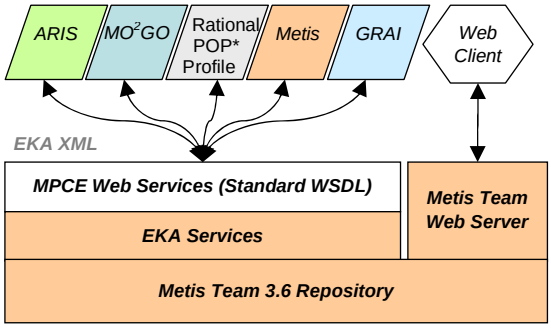

Figure 6. MPCE version 1.0 architecture.

4.1 Enterprise Knowledge Architecture

In order to meet the requirements outlined in section 3, we have defined a metamodelling framework, shown in Figure 7. The core element of this architecture is ***Object***. An Object may have ***Properties***, and ***Relationships*** link two Objects through **Origin** and **Target Roles**. Relationships, roles and properties are also objects, so they may possess properties and have relationships to other objects. Objects, properties, relationships, and roles, together called *elements*, are contained within ***Views*** that express (partial) models (cf. R9). ***States*** make up the lifecycle of an object (R12).

The EKA does not separate between meta-classes, classes and instances because this would make it difficult to handle and integrate any meta-layer structure. Instead, a special relationship called ***Is*** between two objects (or relationships or properties), denote that the origin is defined by the target, and can thus express both specialization and instantiation (R2). The instantiation relationship ***Is-a*** shares most of the semantics of Is, but it is used to separate meta-levels (for the modelling frameworks where this is required). Other relationship types include general links and associations, and decomposition with (***Part***) and without (***Member***) ownership. Relationships and properties have ***cardinality***. Note that this approach enables classification, decomposition and states of properties, relationships and views just like objects (R11).

The EKA is inherently *reflective* (R5). This makes it *coherent*, so users apply the same modelling constructs (object, property, relationship) and operations on any meta-level. They may perform "metamodelling" operations such as adding a property in the same way on instances and classes, or for that matter relationship and property instances and classes (R4). This facilitates *instance level exceptions and evolution* (R12). Similarly, users may perform modelling operations on classes, e.g. adding default parts and property values.

Multi-dimensional views (R1) are captured as multiple "Is" or "Is-a" relationships from an element. This approach can also be applied to mix in new *aspects* locally. For instance, if a group wants to add a cost dimension to a process model, they simply add an "Is" relationship from "Object" to "Cost Component" in their model. All objects within the model will then inherit the properties and behavior of cost components. Such extensions can be local to each view.

Multiple inheritance (R3) is controlled by "Is" and "Is-a" links between the properties of objects. These links articulate which properties are inherited from which super. This also opens up for reuse along other structures than classification and specialization, e.g. to have property hierarchies cross-cutting the class hierarchy. Through reflection, we may define e.g. that a "Part", "Member" or ordinary relationship is an "Is" relationship as well, enabling reuse along these dimensions. In previous work, we have discovered several scenarios where such inheritance is valuable for and intuitive to business users [Jorg04].

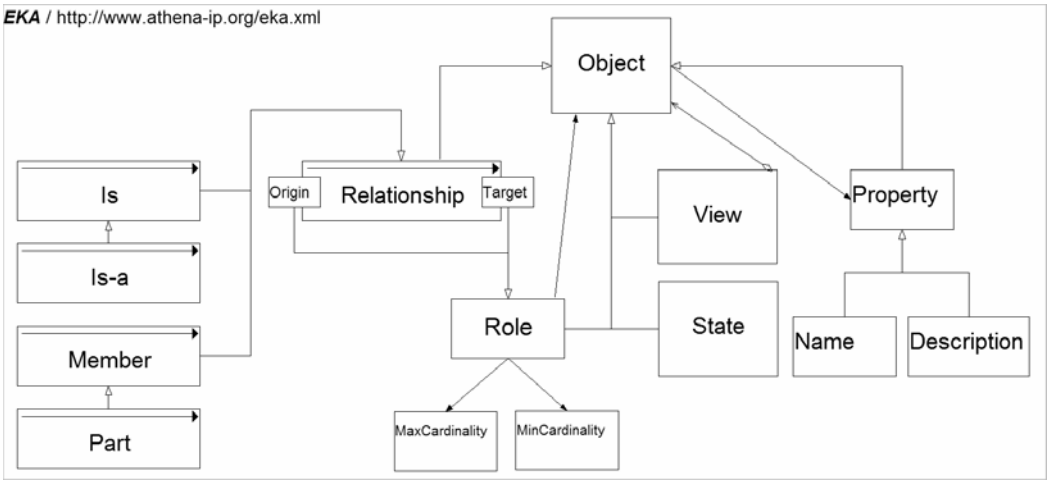

Figure 7. The core concepts of the EKA.

The *expressiveness* of the EKA satisfies the requirements listed in section 3.5. Aspects can be represented as objects, and reused through local "Is" links. Multiple inheritance allows not only aspects, but also interoperability between overlapping classification schemes, e.g. for different companies, tools or disciplines. Because *properties and relationships* are first class elements, they may be defined and managed just like objects, specialized and decomposed. Properties can have an extensible set of properties themselves (meta-properties) that define e.g. how they are to be managed (e.g. "read-only", "derived"). The tasks and decisions involved in creating, selecting and maintaining a relationship may e.g. be defined as a process meta-view on the relationship. For instance, most people would agree that behind the relationship "marriage", there is a complex process that creates and sustains it.

Even though the EKA is this expressive, its core is still quite *simple* compared to most other frameworks (R8). This, together with coherent modelling constructs and techniques across meta-levels, implies that it should be useable for suitably trained business people (R10). More work is however needed in order to verify this. Our previous experiences indicate that users are capable of both modelling and metamodelling, provided customized views are available for such tasks [Jorg04]. The design and derivation of views for different contexts will be a key challenge in our further work. We need to design general frameworks for typical view types, integrated in suitable methodology processes, in a manner which can be adapted to each customer's organizational maturity and individual skill levels.

5 Related Work

Prior to defining this EKA framework we analyzed existing standards such as MOF XMI (Meta Object Facility XML Metadata Interchange) [Omg02], RDF (Resource Description Framework) [W3C04a], and OWL (Web Ontology Language) [W3C04b] Most of these frameworks define a language as a set of class concepts that objects in a model instantiate. Metis today also follows this model, although it supports instance modelling to a larger extent. Classes define the expected properties and behavior of the objects, and the relationships it can have with other objects. During several years of research and industrial experience with meta-modelling, we have discovered numerous shortcomings of this model [Lill03; Jorg04]:

- It handles unforeseen exceptions at the instance level (such as the addition of a property or a relationship) poorly (not meeting R4).
- It does not support instance evolution, where different classes may reflect different states in the lifecycle of an object [AuFB93] (not meeting R12).
- Multiple inheritance and instantiation is often prohibited, making it difficult to capture multiple dimensions of an element (R1).
- Aspects or facets [OpSi97] cannot be used to extend the local meaning of a concept through mix-in inheritance (R3).
- Strict inheritance rules are too rigid for evolving systems, where cancellation inheritance is needed (allowing removal of

inherited features) [Taiv95] (supporting R2 poorly).

- Properties are treated as second class elements, existing only as part of an object. There is poor support for decomposing and specializing properties into attributes, parameter trees and multiple value sets, and you cannot classify or describe properties with properties, making it more or less impossible to manage property structures (R11).
- Relationships are seen as simple one- or two-way links. The tasks and decisions that create and maintain a relationship cannot be captured, making it difficult to grasp the precise meaning and status of each link (R11).

In MOF, we also found the meta-levels to be rigidly separated, and the support for property and relationship structures limited. MOF is also rather large and complex (violating R8). OWL and RDF schema better support property modelling, but treats reflection as an exception rather than an inherent feature, and has poor support for object creation. Their lack of a first class relationship construct is also problematic, because relationships are the most important part of an enterprise architecture model. On the other hand, most of these frameworks, with the exception of basic RDF and OWL Full, enable automatic reasoning (R7) beyond what our flexible and user-oriented proposal cater for. This is the result of a conscious design decision, to start with a simple and flexible core, but at the same time to facilitate extensions that may introduce more rigor and control where it is needed (combining R6 and R7).

6 Implementation and Usage Experience

An XML Schema is defined for storing and exchanging EKA model views. Since RDF is reflective and does not define a type system, it is well suited for our approach. We therefore chose to base the EKA XML format on RDF. We also reuse RDF mechanisms for e.g. identification, data types and collections rather than to define our own. The support of RDF for distributed modelling, e.g. its use of namespaces, is also suitable for the EKA. We do however not utilize XMI, RDF schema or OWL in our definitions, because of the limitations discussed above.

The EKA XML format has been applied to define a set of common concepts for process modelling. These constructs constitute the process dimension of the ATHENA POP* multi-dimensional enterprise modelling core. The five modelling tools are able to exchange such process models. We have also defined tool specific metamodel views, capturing the translation between the POP* core and each tool's particular language. These metamodels have been used to control view merging services provided by the MPCE. In particular, the EKA has enabled us to handle specific mapping problems:

- Between bipartite graphs such Event-driven Process Chains (EPC) and single type graphs, by representing EPC events as objects that are members of the flow relationships between the functions.
- Between tools that support roles and tools that do not, by representing roles as part of the relationship (cf. Figure 7).
- Enabling tool specific mappings, e.g. for transferring execution properties from a high-level process language (GRAI) to an executable specification (MO^2GO).

We have thus demonstrated that the EKA is suitable for achieving interoperability at the model and language level.

7 Conclusions and Further Work

This paper has presented interoperability challenges associated with heterogeneous modelling architectures. A perspective was advocated where every model is regarded as an incomplete and partial view. Interoperability problems were conceptualized as the lack of relationships between heterogeneous views on different meta-levels. Requirements for a modelling architecture that facilitated such mutually reflective views were articulated. It was found that existing architectures could not meet these requirements directly, and thus a new solution, called the Enterprise Knowledge Architecture (EKA) was presented. The underlying perspective of the EKA is aligned with human knowledge, sense-making and communication, rather than software programming languages. Experiences show that the EKA can be implemented and applied as a model and metamodel exchange format.

The ATHENA project aims to promote its results as international interoperability standards where that is appropriate. Both the services of the modelling platform for collaborative enterprises (MPCE) and the EKA formats will be considered as candidates for such standardization. In our further research, we see the need for addressing several open issues:

- Whether the EKA also can utilize metametamodels to achieve interoperability, and not just languages and models as today?
- What relationship types are needed for connecting heterogeneous views (other than "Is"), and what task and decisions patterns create them?
- Which kinds of views are needed for different purposes and what services should they include?
- Replication, reuse and other basic modelling services need to be detailed and implemented in a flexible manner. Inheritance rules should be defined to relieve the users of having to define change scopes and reuse patterns manually every time they change a reused view [Jorg04].
- What negotiation, mapping and resolution services are useful for model driven interoperability establishment on top of the EKA?

7.1 Acknowledgements

This work is supported by the European Union, 6th framework program for research, through the ATHENA integrated project (2004-2007). Other partners in the project have contributed to this work by challenging our ideas and proposing improvements.

References

[Athe04] ATHENA 2004- 2007, Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their Applications, http://www.athena-ip.org/, 22-09-2005.

[AuFB93] Augeraud, M. & Freeman-Benson, B. N. 1993. Dynamic Objects, ACM Conference on Organizational Computing Systems (COOCS), Milpitas, California, USA.

[BeLu66] Berger, P. L. & Luckmann, T. 1966. The Social Construction of Reality. A Treatise in the Sociology of Knowledge. Penguin Books, USA,

[Jorg04] Jørgensen, H. D. 2004. Interactive Process Models. PhD thesis, Norwegian University of Science and Technology, Trondheim, Norway, http://www.diva-portal.org/ntnu/theses/abstract.xsql?dbid=4, 22-09-2005.

[Kang99] Kangassalo, H. 1999. Are Global Understanding, Communication and Information Management in Information Systems Possible?, in Conceptual Modeling. Current Issues and Future Directions, Springer, LNCS 1565.

[Kicz96] Kiczales, G. 1996. Beyond the Black Box: Open Implementation, IEEE Software, vol. 13, no. 1.

[Lill03] Lillehagen, F. 2003. The Foundation of the AKM Technology, in Jardim-Gonçalves, Cha & Steiger-Garção (eds), Concurrent Engineering, Enhanced Interoperable Systems, Rotterdam: Balkema.

[Omg02] OMG 2002. Meta-Object Facility Specification Version 1.4, http://www.omg.org/mda/specs.htm#MOF, 22-09-2005.

[OpSi97] Opdahl, A. L. & Sindre, G. 1997. Facet Modelling: An Approach to Flexible and Integrated Conceptual Modelling, Information Systems, vol. 22, no. 5.

[SLJ+05] Solheim, H.G., Lillehagen, F., Jørgensen, H.D., Karlsen, D. & Smith-Meyer, H. 2005, Obliterating the border - Concurrent modeling and execution platform, Proceedings of CE 2005.

[Styh02] Styhre, A. 2002. Thinking with AND: Management Concepts and Multiplicities, Organization, vol. 9, no. 3.

[Taiv96] Taivalsaari, A. 1996. On the Notion of Inheritance, ACM Computing Surveys, vol. 28, no. 3.

[Trou05] Metis 5.0 product information, http://www.toux.com, 22-09-2005.

[Weng98]Wenger, E. 1998. Communities of Practice. Learning Meaning and Identity. Cambridge University Press, UK.

[W3C04a] RDF Primer, W3C Recommendation 10 Feb 2004, Manola, Miller, eds. http://www.w3c.org/RDF/, 22-09-2005.

[W3C04b] OWL Web Ontology Language Reference, W3C Recommendation 10 Feb 2004. Dean, Schreiber (ed), http://www.w3c.org/2004/OWL/, 22-09-2005.

Håvard D. Jørgensen, Frank Lillehagen, Dag Karlsen

Troux Technologies AS
PO Box 482, N-1327 Lysaker, Norway
{hjorgensen, flillehagen, dkarlsen}@troux.com
http://www.troux.com/services/research/