Enterprise Information Systems, 2015
Vol. 9, No. 7, 681–708, http://dx.doi.org/10.1080/17517575.2013.792395

Taylor & Francis
Taylor & Francis Group

Incremental checking of Master Data Management model based on contextual graphs

Myriam Lamolle[a]*, Ludovic Menet[b] and Chan Le Duc[a]

[a]*LIASD, IUT de Montreuil — University Paris 8, F-93100 Montreuil, France;* [b]*Orchestra Networks, 75, Boulevard Haussmann F-75008 Paris*

(*Received 28 September 2011; final version received 31 March 2013*)

The validation of models is a crucial step in distributed heterogeneous systems. In this paper, an incremental validation method is proposed in the scope of a Model Driven Engineering (MDE) approach, which is used to develop a Master Data Management (MDM) field represented by XML Schema models. The MDE approach presented in this paper is based on the definition of an abstraction layer using UML class diagrams. The validation method aims to minimise the model errors and to optimisethe process of model checking. Therefore, the notion of *validation contexts* is introduced allowing the verification of data model views. Description logics specify constraints that the models have to check. An experimentation of the approach is presented through an application developed in ArgoUML IDE.

Keywords: models; metamodels; incremental validation; UML; XML Schema; master data management

1. Introduction

There is no need to prove that Model Driven Engineering (MDE) is a solution to reduce the difficulties encountered during complex structure modelling. Indeed, the main goal of MDE is to move the complexity of realisation of a software application to the specification of this application (Frankel 2003; Mellor and Balcer 2002). Therefore, the programming language is ignored and the specification is based on an abstract modelling process using different standards such as UML (OMG 1997), Object Constraint Language (OCL) (OMG 2002), Meta Object Facility (MOF) (OMG 2006) or XMI (OMG 2007) from OMG.[1]

In the context of Master Data Management (MDM), which is a way to unify, manage and integrate master data across companys' Information Systems, a MDE approach is recommended to be followed at the modelling level (Menet and Lamolle 2008a) and at the mapping models level (Menet and Lamolle 2008b). MDM models are XML Schema (SX) documents (Thompson et al. 2004) and XML model instances, which contain Master Data Values.

On the other hand, UML provides a powerful graphical modelling formalism widely used in software engineering. The richness of the UML formalism is used to make the modelling and the exchange of XS models easier. Moreover, MDM and XML Schema Definition (XSD) profiles are specified at the UML metamodel level. Therefore, an

*Corresponding author. Email: myriam.lamolle@iut.univ-paris8.fr

end-user, using a UML Integrated Development Environment (IDE) including MDM and XSD profiles, defines his/her own model by a MDM/XSD view. The research work is focused on UML class diagram metamodel corresponding to XS semantics (i.e. static representation of data model).

At each level, the changes made to the models must be progressively validated to obtain 'live' stable versions. Structural consistency and validation processes are the main aspects during the steps of model definition. The classical validation approaches check the consistency of the whole model. These approaches are appropriate when the size of models is not very large. But, they are no longer suitable in industrial contexts because they become too expensive and time-consuming. These issues arise from unused information, or from information that is not related to updates of models but handled unnecessarily during the validation processes of the whole model.

In this paper, an incremental model validation approach is proposed for a MDM model that is designed by using a UML class diagram with a MDM profile. Such a profile relies on an XS profile. The term 'model validation' is used herein in the sense of 'consistency checking', which is used for evaluating models with respect to semantic and syntactic criteria. Consistency checking evaluates a model against its metamodels. In addition to metamodels, constraints can be used (mostly defined in the UML metamodel with OCL) to validate a model. MDM and XSs satisfy specific semantic and syntactic criteria, which are defined from profiles at the metamodel level. The incremental model validation has been defined in four steps:

(1) formalising the metamodel in terms of UML class diagrams to express validation rules;
(2) defining validation contexts based on graph representation. A validation context is a set of elements (i.e. sub-graphs) to be validated. These elements may be changed when an action (such as creation, deletion or modification) is performed. Contexts make it possible for parts of the model concerned by the element modification to be located;
(3) defining the incremental validation algorithm;
(4) implementing and integrating the proposed approach in the UML modelling IDE.

The goal is to validate structural constraints that are defined at the metamodel level. Models corresponding to the metamodel are created from the MDM and XS profiles. Therefore, they are compliant UML class diagram models. Then, UML class diagrams are mapped from MDM point of view to formal models that are expressed straightforwardly in description logic (DL) axioms. In turn, DL reasoners check automatically the consistency of the part of the model that is changed by an action such as creation, modification or deletion.

The paper is organised as follows. Section 2 presents the MDM notion and the proposed MDM solution, the EBX architecture. Section 3 is aimed at introducing the various approaches of model validation in the scope of MDE. Section 4 presents a formal representation of metamodel by mathematical object. Section 5 describes the incremental validation approach based on the notion of validation contexts. Section 6 shows how to implement the proposed approach by exploiting DLs and inference engines. Finally, Section 7 presents the conclusions and potential outlooks of this work.

2. EBX platform for MDM

2.1. Principles of MDM

Within the framework of interoperability of heterogeneous data sources, there are two main data integration approaches: the virtual approach (or mediator) (Lenzerini 2002) and the materialised approach (or data warehouse).

The virtual approach provides a global vision by using a single schema to represent all the heterogeneous data sources. This schema can be automatically defined with tools, or schemas extractors. The well-known TSIMMIS (The Stanford-IBM Manager of Multiple Information Sources) (Garcia-Molina et al. 1997) project, developed by the Stanford University, is based on this approach. One of the purposes of TSIMMIS is to integrate heterogeneous sources which are slightly structured and able to change in time. Apart from TSIMMIS, we can also quote other projects such as Distributed Information Search Component (DISCO) (Tomasic, Raschid, and Valduriez 1998) and YAT (Simeon 2000).

In the virtual approach, requests from users are formulated with the semantics of the global schema. To execute these requests, they need to be translated into sub-requests that are understandable to all the sub-schemas corresponding to several data sources.

More precisely, the mediator in this approach knows the global schema and abstract views of data sources. By using these views, the mediator decomposes a user request into sub-requests and submits them to wrappers. These sub-requests are finally translated into the language of data sources by wrappers.

Unfortunately, the virtual approach is not suitable in the context of MDM since data are embedded in the local sources, and can have a heterogeneous representation.

Regarding the materialised approach, data are copied into the data warehouse (or data repository). Actions on the data warehouse are asynchronous with regard to the sources. Data of the repository are disconnected from those included in the heterogeneous sources. Propagating modifications from the warehouse to the sources (and vice versa) is delegated to an integrator, which maintains mappings between the data warehouse schema and schemas of sources. Unfortunately, some limitations remain in this approach in terms of data management, such as:

- the implementation of roles and access rights to access to data,
- the uniform data representation within an information system through a unique management tool,
- the tools for auditing and versioning data,
- the efficient management of data lifecycle and the concurrent access to data between several users.

The MDM approach (Régnier-Pécastaing, Gabassi, and Finet 2008) has been defined to address these limitations, and in the same time to focus on the problematics of the data sources federation.

Without MDM, the propagation of data updates has to be made without a central data repository or a common model of information (often by peer-to-peer architecture). This way is rather common in basic Extraction-Transformation-Loading (ETL)[2]/Enterprise Application Integration (EAI)[3] architectures.

MDM (Loshin 2008) is a way to unify, manage and integrate master data across the Information Systems of business companies. These data can be of several kinds (products, services, offers, prices, customers, providers, legal data, financial data, structures, persons, etc.).

2.2. EBX architecture

In the context of Service-Oriented Architecture (SOA), Bonnet, Detavernier, and Vauquier (2009) suggested that the use of MDM must meet the following requirements:

- unified modelling of the Master Data (MD) using UML which is then translated into XSs, which is a layer above specific data formats. Moreover, the XSs facilitate data access from different applications. The unified MD model including rules makes possible the validation of data updates with respect to format and the referential integrity constraints, etc.;
- creating a MD warehouse, which physically stores the values of this data. This reference system contains the real information related to the MD;
- setting up a business tool, which will enable the functional and technical teams (depending on authorisation rights) to update and access to the MD. This tool replaces the old data-update solutions for the mainframe as well as for the Java, XML and database tools that are used by developers and production teams;
- modelling and implementing data integration processes between the applications and the warehouse. These processes are integrated by an Enterprise Service Bus (ESB)[4] or ETL type of technical layer.

A MDM solution of the materialised approach is proposed herein by a platform (called EBX) implemented in Java and XSs. EBX is based on two concepts: (i) an **adaptation model,** which is a data model for a set of MD. It is an XS document; and (ii) an **adaptation,** which is an XML instance of the adaptation model which contains Master Data Values. Using XSs allows each node of the data model corresponding to an existing data type according to the W3C standard (Thompson et al. 2004) to be specified.

The solution presented in this paper provides a way to unify MD from several data sources. The existing works on data integration (Baril and Bellahsene 2003; Golfarelli, Rizzi, and Vrdoljak 2001; Nassis et al. 2004; Abiteboul et al. 2002; Delobel et al. 2003) are focused on conceptual models without semantic relations between concepts. Therefore, the metamodel of the adaptation models is enriched by adding some specific object concepts (e.g. generalisation, specialisation, aggregation, composition, dependency and so on) to represent semantic relations.

Therefore, this MDM solution must be completed by a MDE approach to maintain the consistency of the global XSD model and its adaptation models. On the other hand, the integration of a new heterogeneous source of data implies a new version of the global model. It is argued that the MDE approach is a good way to optimise the versioning of schemas by implementing an incremental validation of models.

To take into account the OMG recommendation about MDA, two profiles have been designed in UML: XS and MDM profiles[5] (Menet and Lamolle 2008b). To put it in a nutshell, the XS profile is based on XSD concepts.[6] XS profile is established by the following elements:

- **Metaclasses:** Package, PackageImport, Class, Property, DataType, Association, Constraint, Comment.
- **Stereotypes:** Schema, XSImport (Import, Redefine, Include), XSEntity, XS data-types, XSElement, GlobalElement, SimpleType, ComplexType, Sequence, Choice, All, Key, Ref, KeyRef, Unique, List, Union, Attribute, GlobalAttribute, At-tributeGroup, XSDocumentation, Documentation, AppInfo, Facet,

FacetEnumeration, FacetLength (FacetMinLength, FacetMaxLength), FacetPattern, FacetBoundary, etc.

- **Enumeration:** FormDefault, BlockDefault, FinalDefault.

The MDM profile inherits from the XS profile and is established by the following stereotypes: AdaptationModel, Root, Service, MDMEntity, Domain, MDMFeature, MDMSim-pleElement, Table, PrimaryKey, AutoIncrement, Function, DynamicFacet, DynamicEnu-meration, DynamicLength (DynamicMinLength, DynamicMaxLength, DynamicBound-aryLength), ConstraintEnumeration, Label, Description, Default-ErrorMessage, Manda-toryErrorMessage.

Example. Let us consider a domain of books publication. The *Author, Title, Publisher* concepts must be represented in a UML class diagram from the MDM point of view (i.e. by using the MDM profile). The MDare saved in a relational database. The *Author* concept is defined by the properties *aut_id, firstName, lastName, address, city* and *country.* The *Title* concept is defined by the properties *title_id, title, type, unitPrice, totsales, pub-date.* The *Publisher* concept is defined by the properties *pub_id, name, city* and *country.* A *Title* is written by one author and published by one publisher.

In Figure 1, the domain of book publication (from a relational database) is represented by the stereotype «*root*». It is associated with *Title, Author* and *Publisher* stereotypes «*complexType*» «*Table*». The stereotype «*TableRef*» in *Title* expresses the integrity constraint related to foreign keys. For instance, the attribute *aut_id* in *Title* represents a reference to an instance of *Author.*

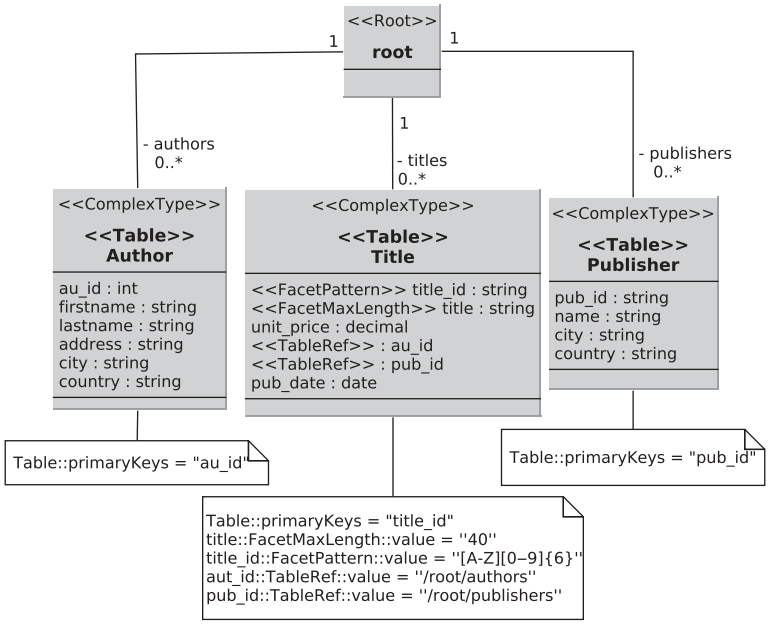

Figure 1. Partial class diagram of book publication example from MDM point of view.

3. State of the art

The classical approaches to validation of models are focused on checking the integrality of a model. In these approaches, when a model is updated, it is necessary to validate the whole model in order to check whether the modification has implied some inconsistence in the structure of the model. These approaches are suitable for small models but not for large ones which are common in industrial applications. The main cause of this issue is that the information provided by previous validation processes is not used by later validations. Furthermore, an incremental validation approach needs to specify what rules have to be verified for each action on the model and which parts of the model have to be verified. The solution to the first question is to consider that, for a given action, a series of rules have to be verified. In that matter, a classification of rules is used to indicate exactly which rules have to be checked when a specific action occurs. Regarding the second question, the notion of contexts that is going to be introduced is used, for a given model, to know whether each part of the model could satisfy a validation rule. Using this context mechanism, the validation process can be optimised by considering that only modifications that affect a kind of contexts can trigger verification. Based on these principles, an incremental validation algorithm will be setup.

In the following subsections, a non-exhaustive list of approaches of model validation in the MDE field is presented.

3.1. Approaches based on OCL

UML class diagrams do not describe all relevant aspects of system specifications. However, it is necessary to have the possibility to describe additional constraints on the model objects. For example, a management rule such as *A nurse cannot access the file of a patient who has left the hospital more than 15 days ago* is not directly expressible. This kind of integrity constraint, which is very important in the business process, cannot be represented in a UML class diagram. The usual representation is done using natural language. But, a non-formal language representation leads to ambiguities. On the other hand, using a formal language removes these ambiguities, thanks to its precise grammar. In this way, the OMG has adopted the OCL (OMG 2002) to describe UML semantic part and to complete the various UML diagrams allowing constraints to be expressed. The advantages of OCL are: (i) it is a language which is easy to understand and to use; (ii) it is an intermediate language between natural language and mathematical language and (iii) it is widely used in metamodel description (including UML metamodels).

OCL is a declarative language without side-effects. So, the evaluation of OCL constraints expressed in the model do not change the model instances. OCL allows the following constraints to be expressed:

- *class or type invariants:* these constraints must always be checked to ensure class instances or concerned type run well;
- *operation constraints:* they must always be checked to ensure the operation concerned runs well;
- *pre and post conditions of operation:* these constraints must be verified before (pre) or after (post) the operation execution;
- *guards:* constraints concerning object state change;
- *browsing expressions:* constraints to represent paths in class structures.

However, OCL raises issues (Olivé 2007) such as:

- lack of readability for complex constraints;
- lack of rigour compared to the language Z (Spivey 1992) or the language B (no possible proof) (Abrial 1995);
- limitation of its power of expression.

In the MDE field, OCL can be used to define constraints on the metamodels, which restrains the structure of models. Several works use OCL to define methods of model validation. Neptune (Millan et al. 2008) is a method inspired by the Unified Process (Jacobson et al. 1999). This method covers the analysis and design phases of software systems. It consists in modelling the system by a set of UML diagrams, which must satisfy validation rules specified at several levels of the model design. In Neptune, the validation rules are defined with OCL within metamodel and model. One of the qualities of Neptune is that rules can be edited, added and deleted at any time during model design. However, the disadvantage of Neptune is the lack of dependencies management between validation rules. Yet, dependency management is essential to a system to detect conflicting rules. Moreover, the validation process is the user's responsibility. It does not work in the background. It is also important to note that despite the choice of validation types (whole model or sub-parts), the Neptune validation mechanism is not incremental. This poses performance problems in terms of validation time for large scale models.

UML Analyser (Egyed 2007b) implements an incremental validation mechanism for UML models. Unlike Neptune, the model checking is automatically done in the background during model edition. UML Analyser supports scalability for industrial-size models. But the drawback of this method is the use of a specific formalism to define validation rules. The incremental validation mechanism of UML Analyser is based on the *scope* of validation rules. The rule's scope allows all model elements and all events affecting the rule validity to be determinated.

The advantage of the model validation approaches based on OCL is that they are properly integrated in the modelling environment insofar as it is not necessary to transform a model into a mathematical formalism to check it. Moreover, approaches such as UML Analyser proved their efficiency on large models. Although OCL is recommended for defining model validation constraints by the OMG, the use of OCL remains limited and is not widespread in IDE. Then, OCL is used to express all rules that cannot be graphically defined. So, the end-user needs to learn the peculiar OCL syntax.

Cabot and Teniente (2009) proposed news incremental techniques to determine, at design-time, when and how each constraint must be checked at runtime to avoid irrelevant verifications. They are *incremental* because '*they minimize the subset of the system state that needs to be checked after each change by assuming that the system was initially in a consistent state and just re-evaluating the elements that may have been affected by that change'*.

3.2. Approaches based on DLs

A formal language allowing formalising the rules to be checked like SBVR[7] (OMG 2008) is an alternative to OCL to express additional constraints. SBVR is typically used in business domains. SBVR allows domain vocabulary, concepts (including fact types) and business rules to be described. In a SBVR metamodel, a noun concept is an object type or an individual concept, the verb concepts represent operations performed by/for a business

entity; the different associations types are represented by associative fact types, categorisation fact types and partitive fact types.

SBVR is a part of OMG's MDA. Therefore, the described vocabulary can be used at Platform Independent Model (PIM) and Platform Specific Model (PSM) levels. The business rules can also be expressed in natural language by employing formal semantics (Baisley, Hall, and Chapin 2005), which are based on typed predicate logic, artihmetic, set and bag comprehension, modal logic, and First-OrderLogic (OMG 2008). Therefore, mapping to different logic-based languages is simple. SBVR supports XS to promote interoperability. Bajwa, Bordbar, and Lee (2011) performed a comparative analysis of SBVR and OCL to identify their principal features such as similarities, differences, key parameters on which these both OMG's standards can work together. And, Cabot, Pau, and Raventos (2010) proposed transformation from UML/OCL to SBVR. VETIS (Nemuraite et al. 2010) is a tool implementing such a transformation.

In the context of MDM, SBVR seems better than OCL to represent semantic business rules, to transform a SBVR model into an ontology, or to integrate a reasoner checking the model consistency.

Indeed, DLs approaches are based on first-order logic languages to represent both validation rules and metamodels, which establish the grounding of models to be validated. Inference mechanisms are used to detect violations of these validation rules. The mechanisms using DLs are performed in two steps. The first step consists in transforming models into a logical representation. The second step checks whether this representation is satisfied with respect to validation rules expressed in DL.

In the literature, there are several researches which have addressed the issue of incremental model validation using logical approaches. Nentwich et al. (2003) used first-order logic to represent the views of models and validation rules between these views. Validation error detection is performed with the help of a theorem prover in a semi-automatic manner.

Sabetzadeh and Easterbrook (2005) used the inference engine Jess (Friedman-Hill 2003) to detect validation errors on UML models. The rules are defined by conditions which trigger solving actions. When these rules are valid, they become usable by users. The checking process is managed in the background in transparent manner. Jess's inference algorithm (RETE) has an algorithmic complexity increasing exponentially in model size (Schmedding, Sawas, and Lausen 2007). The main limitation of this approach lies in the dependency on the metamodel and model modification actions. Each UML class diagram is mapped to the internal formalism of the Jess engine in a specific manner. Moreover, this dependency raises issues about the management of inter-model rules. In the context of complex information systems, data models can be linked with other ones which may result, for instance, from data factorisation and availability. This may imply data dependencies and provide the possibility to specify evolutions causing inconsistencies of linked data models.

Sourrouille and Caplat (2002) used the Sherlock inference engine (Bahl et al. 2007) to manage UML model validation. Sherlock is composed of an inference engine and a set of Sherlock agents. Sherlock works in three steps: (i) identifying dependencies in a model, (ii) building a graph of dependencies and (iii) locating errors. Rules are written by conditional expressions associated with specific actions to be executed in case of failure. Sherlock requires a UML metamodel description. The integration from modelling tool to inference engine is done by means of XML documents.

Logic based approaches enable all model validation activities to be covered: specification of rules with logic languages, detection of inconsistency with inference

mechanisms and inconsistency treatment by means of the specification of correction rules. Moreover, many inference engines such as Racer (Haarslev and Moller 2001), FaCT (Patel-Schneider and Horrocks 1999), FaCT++ (Tsarkov and Horrocks 2006) and Pellet (Sirin et al. 2007) enable reasoning about models. Racer, FaCT and FaCT++ conform to DIG (Bechhofer, Moller, and Crowther 2003), which is a standard protocol to interrogate an inference engine through *http* requests. Racer is a server software, which communicates through network. Racer also offers a functioning form of cluster computing with RacerProxy. This feature gives an advantage. The server is either remote or cluster computing, which greatly facilitates managing strong requests.

However, the first limitation of these approaches is the computational complexity of validation error detection mechanisms. These mechanisms do not enable scaling on large models. An incremental approach is necessary to reduce the impact of algorithmic cost. Another limitation concerns the issue of modelling tool and inference engine integration. This has an impact on automation and the overall transparency of validation error management activities.

3.3. *Approaches based on graphs*

Graphs are a kind of model that have been used in various fields of computer science. On the one hand, they are well suited to formally describe complex structures. On the other hand, the structure of models, visual models in particular, can be best described by graphs because of their multidimensional extension. Graphs can be manipulated by rules-based transformations. Given the current trends in software development such as model-based development, it becomes necessary to describe accurately the model handling, the model evolution, semantic model, etc. Recent research has shown that graphs transformation is a promising formalism to specify model transformations.

However, the use of conceptual graphs for metamodelling has been relatively little studied. Esch (1994) addresses metamodelling by conceptual graphs by proposing the introduction of two relationships *Kind* and *Subt. Kind* binds a concept to the type. *Subt* binds two types such that one is a subtype of the other. The set of *Subt* defines the type hierarchies for each level and *Kind* connects the elements of the hierarchies. However, Esch does not address the issue of relationships and relationship types. Wermelinger defines higher order types in a more formal way and offers a translation into first-order logic (Wermelinger 1995). Gerbé et al. (2003) extend the notions introduced by the former. They define two hierarchies of types, one for concepts types and one for the relations types. Concepts types and relations types are classified according to their semantics. Levinson and Ellis (1992) implemented the first type hierarchy that could support retrieval and classification in logarithmic time by adapting algorithms from chemical graphs. More recent research on chemical graphs has been used in algorithms for computing semantic distance between Conceptual Graphs (CGs). Berners-Lee concluded by comparing the RDF language and the conceptual graphs that the latters are easily integrated into the Semantic Web (Berners-Lee 2001). Sowa (2008, Chapter 5) classified CGs in *core* CGs, *extended* CGs and *research* CG. The latter allows novel techniques for reasoning, knowledge representation and natural language semantics to be explored. The semantics of the core and extended CGs is defined by a formal mapping to and from the ISO standard for Common Logic (CL).

Hayes and McBride (2004) used CL to define the semantics for the languages RDF(S) and OWL. The CL standard specifies three concrete languages that are capable of expressing the full CL semantics: the Common Logic Interchange Format (CLIF), the

Conceptual Graph Interchange Format (CGIF) and the XML-based notation for CL (XCL). RDF and OWL can also express subsets of the CL semantics. However if any statement in RDF or OWL can be translated to CLIF, CGIF or XCL, only a subset can be translated back to RDF or OWL.

The lack of specific representation (language, models, etc.) of some important modelling domains like goals and non-functional requirements led to introduce new standards in the scope of graph. The User Requirements Notation (URN) (Amyot 2003; URN 2003), standardised by ITU-T as Recommendation Z.151, combines two points of view: the Goal-oriented Requirements Language (GRL) (GRL 2006) and the Use Case Map (UCM) notation (UCM 2003). This standard intends to combine goals and scenarios for expressing and reasoning about functional and non-functional requirements. GRL is a graph. XS model validates a GRL instance. Therefore, it is possible to use the same process of graph checking to verify the consistency of GRL instances. Its evaluation shows the impact of qualitative decisions on high-level functions. A satisfaction evaluation is possible by fuzzy logic. To integrate GRL in UML, a GRL profile must be defined at the metamodel level in the same way as the MDM profile presented in the Section 2.2.

Moreover, approaches based on graph grammars are derived from widely studied theory in computer science literature (Rozenberg 1997; Bardohl et al. 1999). The principle of these approaches is to consider the models as graphs and the validation errors as patterns in the graphs. Two approaches are outlined and conclusions are drawn on their contributions and limitations.

Fujaba[8] (Geiger and Ziindorf 2006) is a UML environment tool suite for generating executable prototypes from a class diagram. Fujaba is based on graph transformation techniques to define and to detect validation errors. A preliminary study of dependencies between validation rules has been done, but it only works in the particular case where two rules are in conflict (resolving one validation rule involves the violation of another validation rule). With this solution, a good knowledge of the UML metamodel, the language OCL and the techniques of graph rewriting are needed to define validation rules. No study exists on the performances of this system. On the other hand, pattern detection in a graph is a problem known to be NP-complete (Varro et al. 2005).

The Attributed Graph Grammar (AGG) (Taentzer 2004) approach allows validation errors to be defined, detected and resolved. AGG uses an external graph tool, which implements the typed graph theory (Folli and Mens 2007; Hermann et al. 2008). An important contribution of this approach is to enable the detection and the management of dependencies between validation rules. Nevertheless, the dependency analysis between validation rules makes the flexibility of the management of these rules more difficult. Indeed, the complexity of the impact analysis algorithm during modification is expensive because of the need to perform it again at each modification of validation rules. From the checking automation point of view, detection triggering is manual unlike the implemented method in Fujaba.

Other works are focused on a validation approach based on graph grammar. The work of Goedicke, Meyer, and Taentzer (Goedicke, Meyer, and Taentzer 1999) used AGG but do not deal with the issue of dependency between validation rules. On the other hand, the validation error detection activity requires a manual coordination of the actors involved in the model design process. It is not executed automatically as in Fujaba. It has been shown (Engels et al. 2002) how it is possible to ensure validation properties from an ordered set of graph transformation rules. This approach has the advantage of demonstrating that it is possible to address validation issues through model structure analysis techniques.

Metamodelling platforms have emerged using the advantage of a visual graph representation but also by integrating standardised languages such as OCL in MOFLON (Ameluxen et al. 2006) to specify constraints and facilitate the implementation of graph transformation rules.

From these approaches based on OCL, DL and graphs, DL and graph approaches are combined. The DL power contributes to define the validation rules applied to UML profiles (i.e. XS and MDM profiles). And, the graph is exploited to represent and to handle the models, which have to be validated (Ameluxen et al. 2006; Chein and Mugnier 2008).

4. Formal representation of the metamodel concept

Model validation is a crucial step in the modelling process. Basically, the existing validation ways presented in Section 3.3 are validations of the whole model. Therefore, these kinds of validation are inadequate for large models. To introduce an incremental model validation, it is necessary to formalise the metamodel in a logical formalism to which there are already reasoners or provers available. This way, inference services offered by these reasoners, can be exploited to check this formalised metamodel for rules and constraints, which are expressed in the same formalism. Therefore, the first step of the method proposed consists in formalising the metamodel notion. In this section, the formalisation of the metamodel is explained, and can be applied to the UML class diagram metamodel.

4.1. Definition

A metamodel can be seen as a set of *classes, attributes* and *references.* Therefore, the metamodel (MM) is defined as an 8-uplet $MM = \{MC, P, R, T, MetaClass, PropType, RefType, Super\}$ where:

- MC is the finite set of meta-classes of the metamodel,
- P is the finite set of properties (attributes) of the metamodel,
- R is the finite set of references (associations) of the metamodel,
- T is the finite set of primitive types used in the metamodel,
- $MetaClass$ is the function which returns the metaclass of each property and reference,
- $PropType$ is the function which associates each property to its primitive type,
- $RefType$ is the function which associates each reference to its type of metaclass,
- $PropValue$ is the function which associates a value to each property of a metaclass,
- $RefValue$ is the function which associates a value to each reference of a metaclass,
- $Super \subset MC \times MC$ is the inheritance relation between the metaclasses of the metamodel. This relation is binary, not reflexive and not symmetric.

From these definitions, a metamodel is defined as a set of metaclasses, properties, associations and inheritance relations.

4.2. UML class diagram metamodel

Figure 2 presents the simplified UML class diagram metamodel as defined in Section 4.1.

In the current version, the notions of *interface* and *operation* of UML class diagram metamodel are excluded because of the MDM and XS model semantics (mainly, the static point of view).

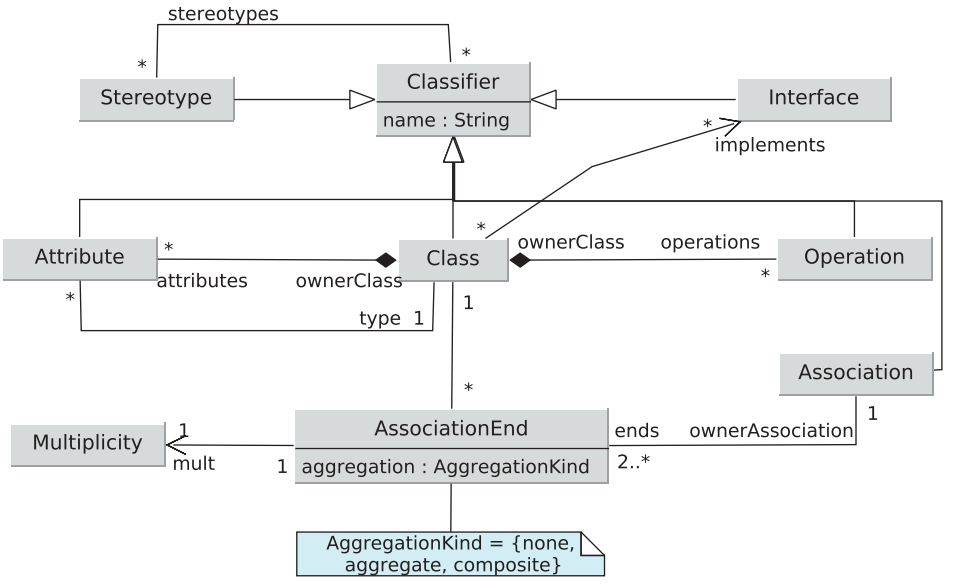

Figure 2. Partial metamodel of UML class diagram.

Indeed, in this research context, the UML class diagrams are used at the end-user model level to represent XSs and/or MDM (through a profile (Menet and Lamolle 2008b) integrated at the metamodel level) in which interface and operation cannot be represented. Yet, OCL should be used to express additional constraints rather than a short-term solution. Moreover, as very few tools partially or fully support OCL at this time, the authors decided to preserve this metamodelling and to focus on rules and constraints expressed as formulae of DLs. For the same reason, the language SBVR[9] was not chosen because DLs are more expressive and it is possible to check consistency of models by using DL reasoners such as Pellet or Racer. In the context of this work, the formalism of UML class diagrams becomes: *MC* = {Classifier, Class, Attribute, AssociationEnd, Association, Stereotype, Multiplicity}

P = {name, lowerBound, upperBound, aggregation}
R = {type, ownerClass, attributs, mult, ends, ownerAssociation, superClass, stereotypes}
T = {String, Integer}
MetaClass = {Classifier}
MetaClass(lowerBound) = {Multiplicity}
MetaClass(upperBound) = {Multiplicity}
MetaClass(aggregation) = {AssociationEnd}
PropType(name) = {String}
PropType(aggregation) = {String}
PropType(lowerBound) = {Integer}
PropType(upperBound) = {Integer}
MetaClass(type) = {Attribute}, RefType(type) = {Class}
MetaClass(ownerClass) = {Attribute}, RefType(ownerClass) = {Class}
MetaClass(attributes) = {Class}, RefType(attributes) = {Attribute}

Table 1. DL TBox of the partial UML class diagram and its hierarchy.

TBox
Metaclass
Classifier $\subseteq$ Metaclass
Class $\subseteq$ Classifier
Stereotype $\subseteq$ Classifier
Attribute $\subseteq$ Classifier
AssociationEnd $\subseteq$ Classifier
Association $\subseteq$ Classifier

MetaClass(mult) = {AssociationEnd}, RefType(mult) = {Multiplicity}
MetaClass(ends) = {Association}, RefType(ends) = {AssociationEnd}
MetaClass(ownerAssociation) = {AssociationEnd},RefType(ownerAssociation) = {Association} MetaClass(superClass) = {Class}, RefType(superClass) = {Class} MetaClass(stereotypes) = {Classifier},

RefType(stereotypes) = {Stereotype}

Super = {(Class, Classifier), (Attribute, Classifier), (Stereotype, Classifier), (Association, Classifier)}

In the scope of DL, the UML metamodel of class diagrams is represented by a *TBox* (cf. Table 1).

5. Incremental validation of models based on graphs

Classical approaches of the model validation consist in checking the overall model consistency. The main disadvantage of these approaches is that the model should be revalidated in its entirety once a change is made to it, even if the change is minimal (e.g. adding an attribute in a concept of model). Regarding incremental model validation, two issues arise:

(1) What rules have to be verified for each action on the model?
(2) Which parts of the model have to be verified?

The answer to the first question is to consider that, for a given action, a set of rules has to be verified. In this case, a classification of the rules to be checked can be used to indicate which rules must be progressively validated when a specific action occurs, and which rules are not verified because they are not concerned by this action.

Concerning the second question, the notion of contexts, which will be introduced in a latter section, defines the parts of the model that could satisfy or not a validation rule for a given model. Using this mechanism of contexts, the validation process is optimised by considering only the modifications affecting a type of context, which can require its verification. Based on these principles, an incremental validation algorithm will be set up.

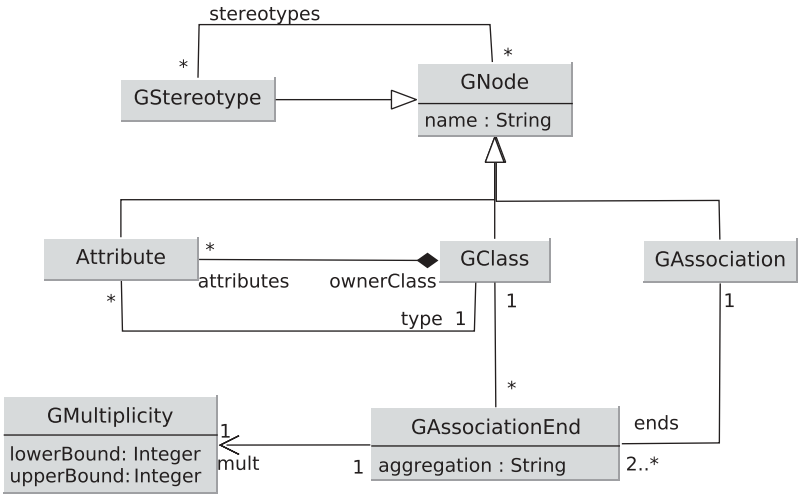

Figure 3. Metamodel of graph structure.

5.1. *Model representation based on graphs*

A model representation has been chosen by means of oriented graphs, as graph search algorithms are well known and easy to update. The graph allows a model to be partitioned and it is possible to do reasoning about the model or its sub-parts. So, UML class diagrams are transformed into graphs. Figure 3 shows the defined graph metamodel.

To simplify the transformation process from UML class diagrams to graphs, the designed graph metamodel is almost similar to a UML class diagram metamodel in the scope of this research work. Figure 4 presents some mapping rules between these two metamodels.

Figure 5 shows a part of the graph representing an academic model about book publication.

Remarks. Note that the elements *GAssociationEnd* and *GAssociation* are merged for representation simplicity in Figure 5.

5.2. *Action typology and validation rules*

Action types and validation rule categories are defined for a data model definition process. The actions *create, delete* and *update* are possible on elements *class, property* and *reference.* Table 2 presents feasible action types on model elements.

Definition. An event *evtX*is the set of feasible pairs of actions/elements combinations where $X \in \{C, D, M\}$, which represents respectively the creation, deletion or modification actions on model elements.

For example, in Table 2, class/update and property/update are events of *evtM* type, and reference/delete is event of *evtD* type. This action typology specifies which triggering events of rule sets have to be checked (cf. Algorithm 1). Four rule sets are defined to classify the consistency rules:

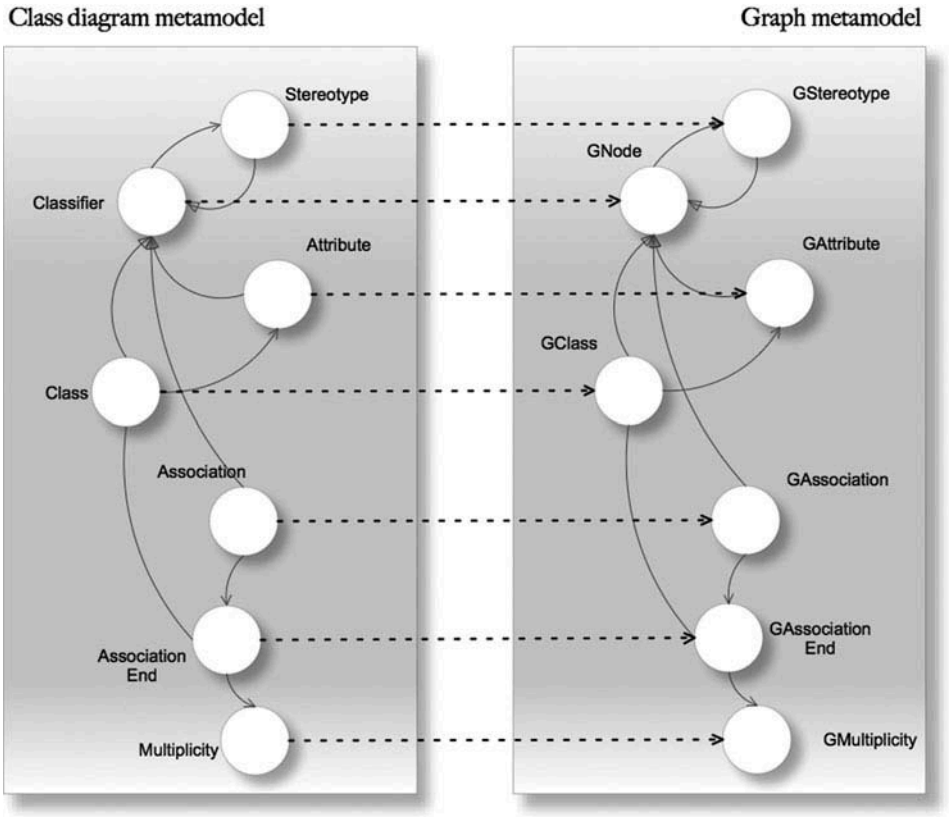

Figure 4. Mapping rules from UML class diagram metamodel to graph metamodel.

- RC is the rule set defined about a model,
- RC_{CL} is the rule set defined and applied to an element of *Class* type,
- RC_{Prop} is the rule set defined and applied to an element of *Property* type,
- RC_{Ref} is the rule set defined and applied to an element of *Reference* type, with

$$RC = RC_{CL} \cup RC_{Prop} \cup RC_{Ref}.$$

The adopted consistency rules classification and the graph representation (Chein and Mugnier 2008) allow validation contexts of a model to be defined.

5.3. *Validation contexts*

The goal is to determine which model sub-parts must be checked when an event is triggered on the model. Therefore, the notion of validation contexts is defined as a set of elements (sub-graphs) to be validated in relation to a type of event.

The contexts are processed with the elements linked to entities resulting from events. The context will be validated when an event occurs if the metamodel updated by the event is satisfied with respect to the rules and constraints.

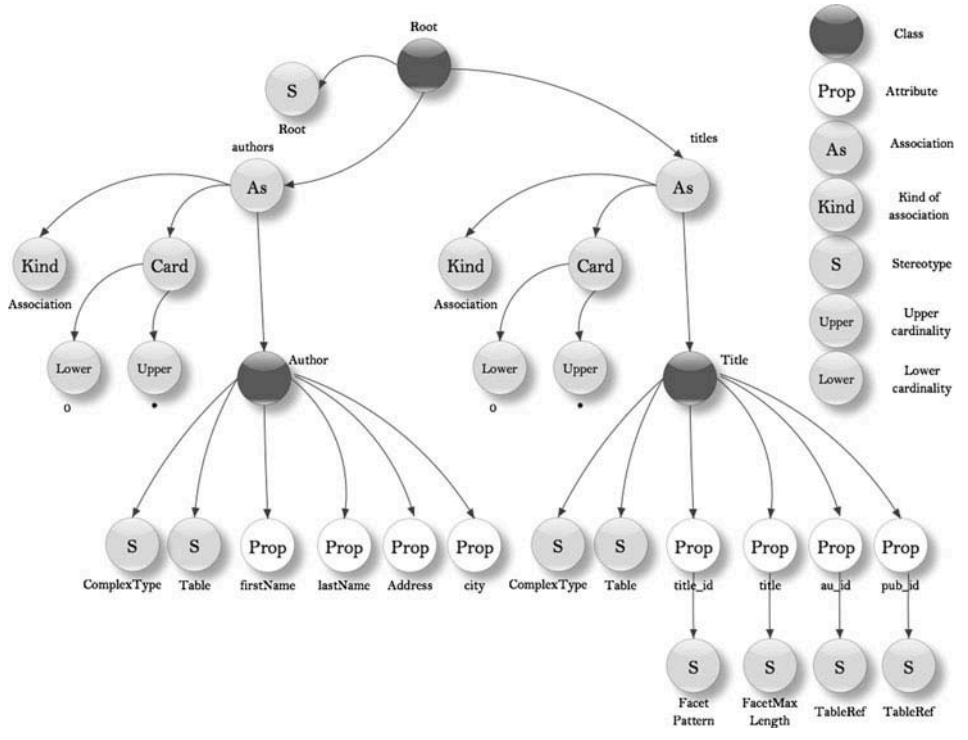

Figure 5. Graph for the book publication example.

Table 2. Events and action types on model elements.

Element/action	Class	Property (attribute)	Reference (association)
Create	*evtC*	*evtC*	*evtC*
Delete	*evtD*	*evtD*	*evtD*
Update	*evtM*	*evtM*	*evtM*

Three kinds of validation contexts are taken into account:

- *Class context,* which represents an element set depending on an event involving an element of type *Class*,
- *Property context,* which represents an element set depending on an event involving an element of type *Property*,
- *Reference context,* which represents an element set depending on an event involving an element of type *Reference.*

To illustrate how to determine the three contexts, let us continue with the *publication book* example of Figure 1, modelled by a UML class diagram using the stereotypes of MDM profile.

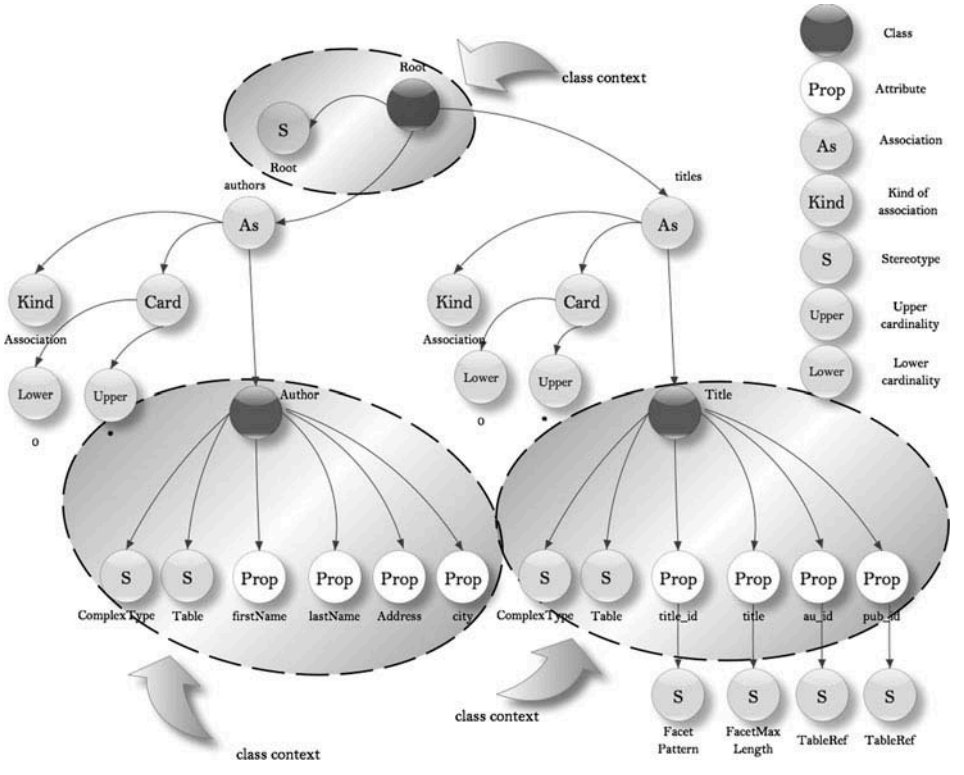

Figure 6. Class contexts validation example – The grey ovals bounded by a dotted line represent three contexts of *Root, Author* and *Title* classes.

These different contexts are shown in Figures 6, 7 and 8, respectively, for the book publication example.

In Figure 6, a dark grey node represents a UML class (i.e. *Root, Author* and *Title*). A node *S* specifies a stereotype. For instance, the *Root* node is defined by the *Root* stereotype and the Author node is defined by *complexType* and *Table* stereotypes. A node A_S is an association, a node *Card* is a cardinality for an association with range defined by nodes *Lower* and *Upper,* a node *Kind* describes a type of association and a node *Prop* represents a class property.

Three class contexts are represented in Figure 6:

- Class context associated with class *Root* is composed of class *Root* and its stereotype (*Root*),
- Class context associated with class *Author* is composed of class *Author,* its stereotypes (*ComplexType* and *Table*) and its attributes *(firstName, lastName, Address, city*),
- Class context associated with class *Title* is composed of class *Title,* its stereotypes *(ComplexType* and *Table*) and its attributes (*title_i d, title, au_i d, pub-id*).

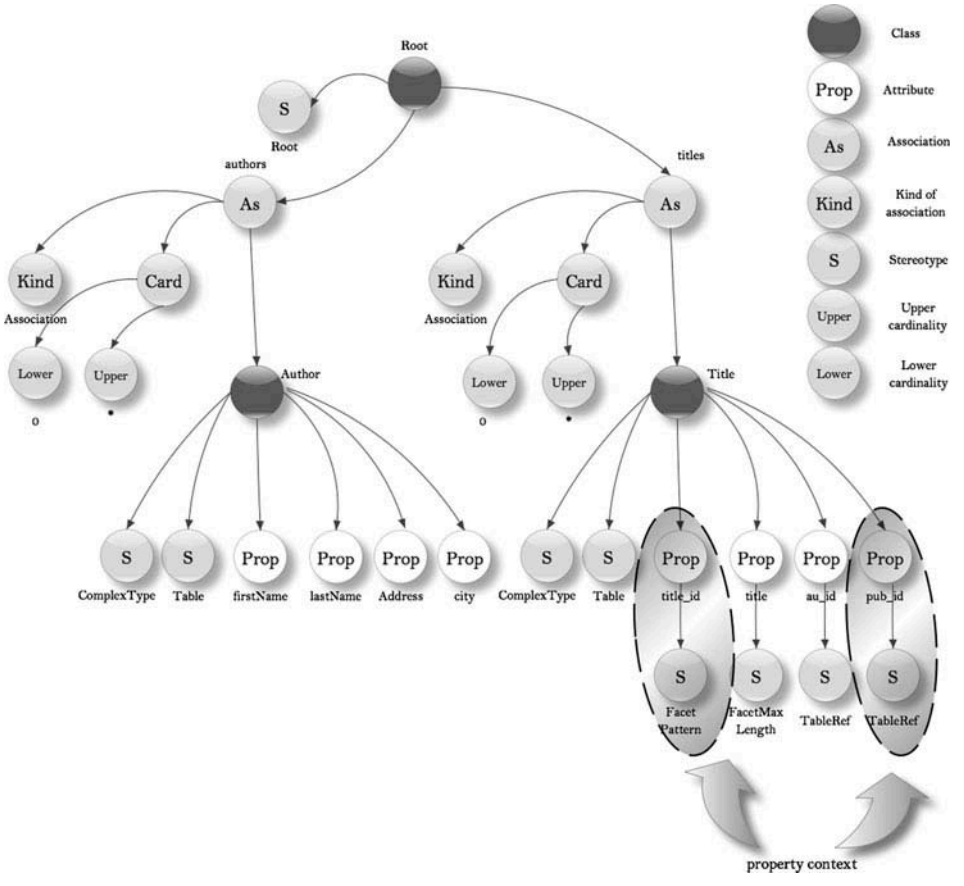

Figure 7. Property context validation example – The grey ovals bounded by a dotted line represent two contexts of *title-id* and *pub-id* attributes of *Title* class.

So, for instance, if a deletion event concerns the *Table* stereotype of *Author* class then the *Author* context class is only updated. In Figure 7, two property contexts appear:

- Property context associated with attribute *Title-id* is composed of attribute *Title-id* and its stereotype (*FacetPattern*),
- Property context associated with attribute *Pub-id* is composed of property *Pub-id* and its stereotype (*TableRef*).
- Finally, two reference contexts are represented in Figure 8:
- Reference context related to association *Authors* is composed of association *Authors,* its type (*Association*) and its cardinalities,
- Reference context related to association *Titles* is composed of association *Titles* and its type (*Association*) and its cardinalities.

Decomposing a model into sub-parts allows the validation process to be optimised by associating each atomic event with a context type corresponding to a sub-graph representing the sub-model to be checked. The incremental validation algorithm is defined from events and validation context notions presented in preceding sections.

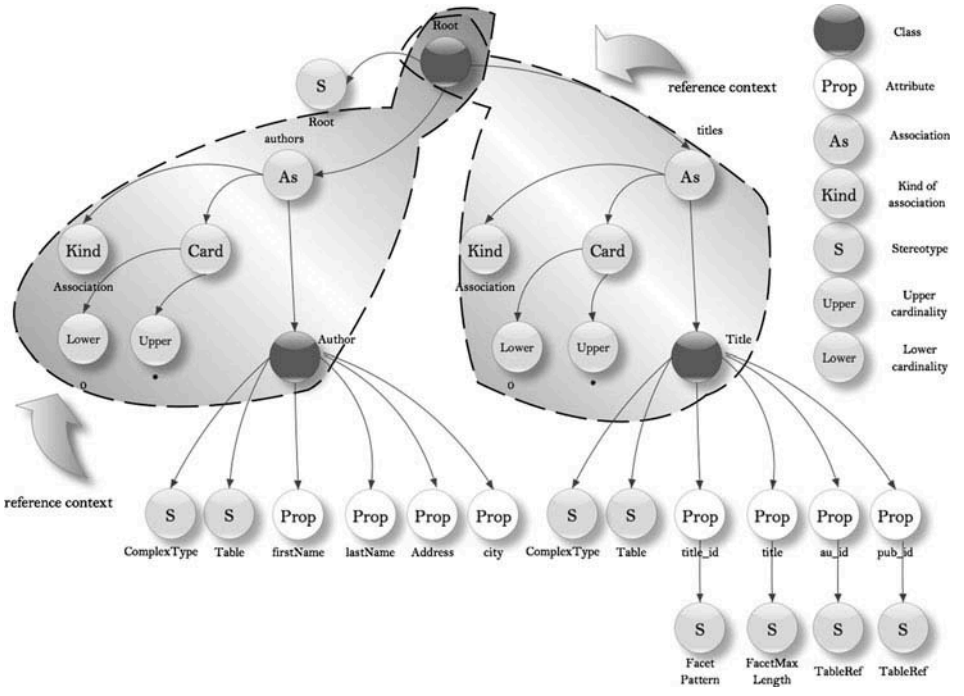

Figure 8. Reference context validation example – The grey ovals bounded by a dotted line represent two reference contexts of authors and titles association of Root class.

5.4. *Incremental validation algorithm*

When an event is triggered, the process of incremental validation of the model runs in three steps:

(1) process the context associated with an event,
(2) process the validation rules associated with this event,
(3) validate the contexts depending on this event.

The following algorithm describes these different steps:

Algorithm 1: Validation of graph update
Input: *newEvent,* a new event and G, a model graph
Output: G, the updated model graph
//Event of creation
1 if *newEvent is a creation event (evtC type)* then
//context creation depending on event
2 create a new context *c* of *newEvent*;
3 add the context *c* to graph *G*;
4 validate the context *c* depending on *newEvent* in the graph *G*;
5 else
//*newEvent* is a modification *(evtM* type) or a deletion *(evtD* type) of G's element
//update element in graph
6 update *G* taking into account *newEvent*;

```
        //evaluation of depending contexts
        //process the contexts in G that are involved by newEvent
7       dependencySet ←getContexts(newEvent, G);
8       for each d in dependencySet do
9       c←d;
        //validation of depending context
10       validate the context c depending on newEvent in the graph G ;
```

Algorithm 2: Validation of context depending on new event on graph
Input: c, a context, and *newEvent,* a new event
Output: a boolean value

```
    //the following algorithm is applied on a context c (1)
    //validate the context c depending on newEvent in the graph G
    //rulesList, a list of rules associated with an event about an
    updated element
1 rulesList — getRules(c, newEvent);
    //Checking rules list rulesList with current context
2 while rulesList no empty do
    //r the current rule to check
3 r ←rulesList.next() ;
4 if r is not valid (taking into account newEvent) then
5 return false;
6 return true;
```

The problems of cycles in the graph are managed by a strategy of graph coloration (Gravier 1997).

6. Implementing incremental validation of model

In this section, an example of implementation of the incremental model validation method using the reasoner *Racer* (Haarslev and Moller 2001) based on DLs is presented. Racer has been chosen for its great performances of reasoning on models of important size. Furthermore, it has a Java API that is able to be exploited by third-party applications. The DL focuses on reasoning services. One goal of DL consists in optimising the reasoning to decrease the computation time. Therefore, a lot of research works about the study of the rate *expressivity/performance* of different DL have been made (Fournier-Viger 2005; Nardi and Brachman 2003; Tsarkov and Horrocks 2003). The main quality of DL engines is their inference algorithms. Their complexity is often lower than that of the first-order logic provers (Tsarkov and Horrocks 2003). DL uses an ontological approach, i.e. general categories of individuals and logical relations between categories or individuals are defined to describe domain's individuals. This ontological approach is natural for reasoning because even if the majority of interactions take place at the individual (i.e. instances) level, the greater part of the reasoning occurs at the category level (Russel and Norvig 2003).

In the following section, the implementation of the incremental model validation is described by exploiting the power of the reasoner Racer in the IDE ArgoUML.

6.1. Integration of Racer in ArgoUML

The first step is to integrate Racer in ArgoUML to use UML and XS models. Therefore, it is necessary to get an evolution of ArgoUML's validation process taking into account (i) the restrictions (cf. interface and operation in Section 4.2) of XS semantic and (ii) the solution of incremental validation. Figure 9 presents the validation mechanism integrated in ArgoUML.

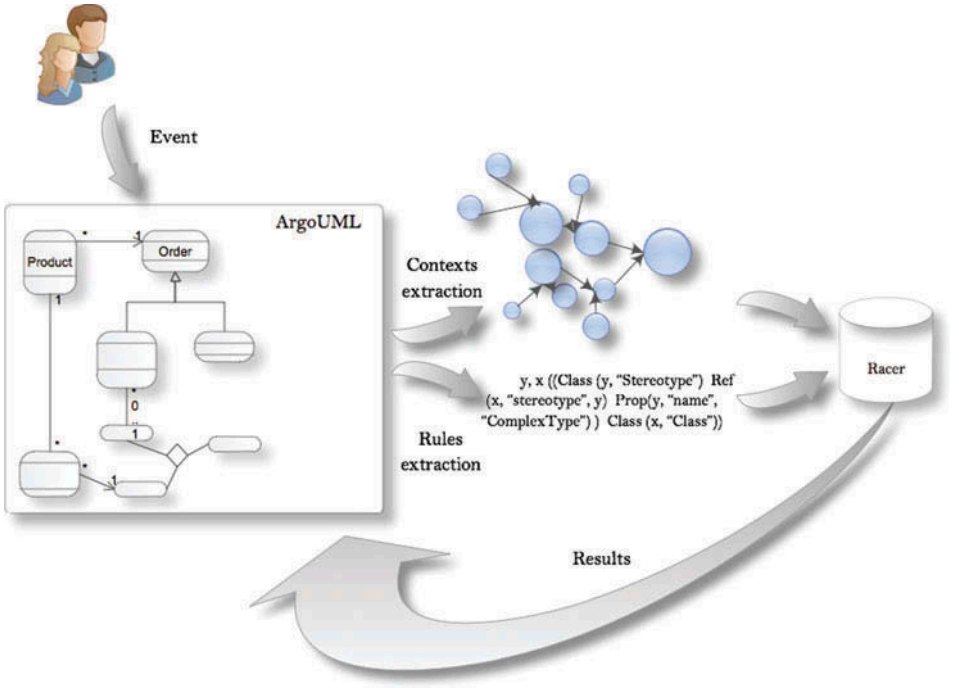

Figure 9. Process of incremental model validation integrated in ArgoUML.

Racer includes a Java API, which serves as an interface between external application and Racer. The algorithm given in Section 5.4 is implemented. When an event occurs on a model, the validation engine is loaded in the background to treat this event automatically and validate it. The validation of the event is done by processing its contexts and the rules applied to context elements. The sets of contexts and rules are sent to Racer through the Java API. Then, Racer reasons about the part of the model to be validated. The inference process result is returned to ArgoUML. Finally, ArgoUML generates a validation report to express errors due to the event occurred from the Racer's messages. Figure 10 shows the end-user report interface.

6.2. Experimental results

The method of incremental validation has been validated on two models having different sizes. The results are compared with those of a classical validation method. The benchmarks used are the time taken for a first model validation, the time taken for a complete revalidation and the time for a third validation after an insertion action, a deletion action and a modification action on a model. The first model validation is tested to compare the validation method presented in this paper, which validates the whole model (and the reasoner must validate all rules) like the other validation processes. The second test (i.e. complete revalidation without any change) is made to compare the validation process, which obtains no rule to be checked because there is no modification event on the managed model, and the classical validation, which behaves like the first validation. The last test is made for the simple actions which are the most performed test during modelling.

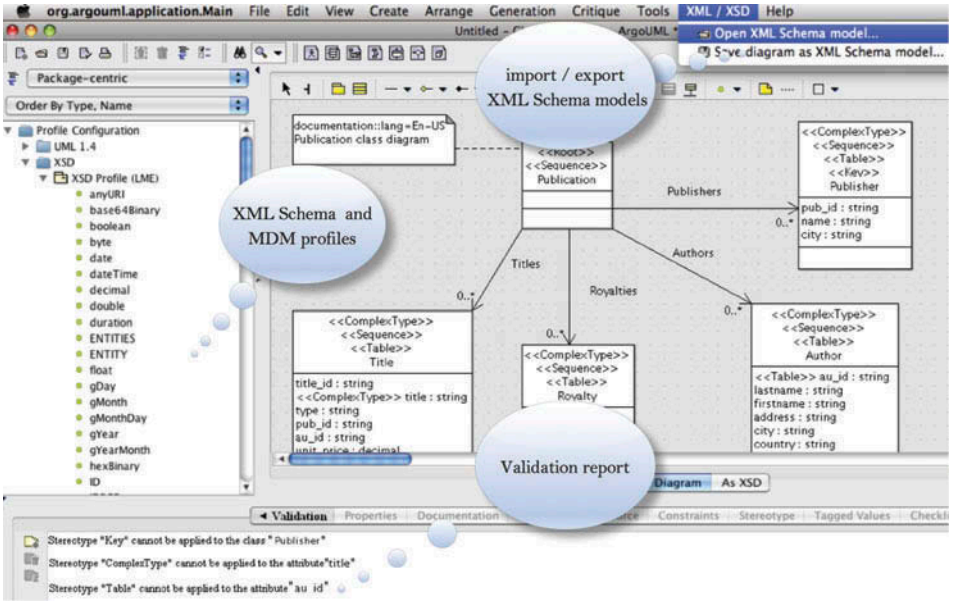

Figure 10. Validation report in ArgoUML.

For the first experimentation, the book publication example (Figure 5) is tested. This example has a small model (56 nodes and 55 edges).

From the results presented in Figure 11(a) for a small model, the approach optimises the validation time whatever the event occurring on the model. If a 'no-incremental' (noted 'classical' in Figure 11) validation method is used, the whole model must be validated again. In this case, the performances remain constant since the model size has been considerably reduced except that a complex element is deleted.

With this approach, the information from the first validation is reused for the second validation without model modification. This feature reduces the validation time by 10 times compared to classical validation. Similarly, when an event occurs on a model, subsequent validations are five times faster on average. Indeed, the validation algorithm detects which model sub-parts have to be validated, and only these sub-parts are (re-) validated. These initial results of the experimentation on a small model are encouraging. Therefore, the algorithm was tried on a bigger model. The graph volume for the second experimentation is composed of 5000 nodes and 4999 edges.

On a large size model, the solution remains very successful compared to a classical validation method. The provided optimisation varies between a hundredth and a tenth of the required time for classical validation (cf. Figure 11(b)). The most obvious gain appears on the model revalidation when no element has been altered. Note that the validation process is not only triggered by an event but also by an explicit call to a model validation. In this case, it is necessary to verify whether one or more parts of a model are checked. This explains why the validation time is never reduced to zero.

One can see from the method that after deleting a complex element, the revalidation time is longer than after inserting or updating an element. This is due to the evaluation process of contexts associated with the deleted element. This process is more or less efficient depending on the complexity of the model at hand.

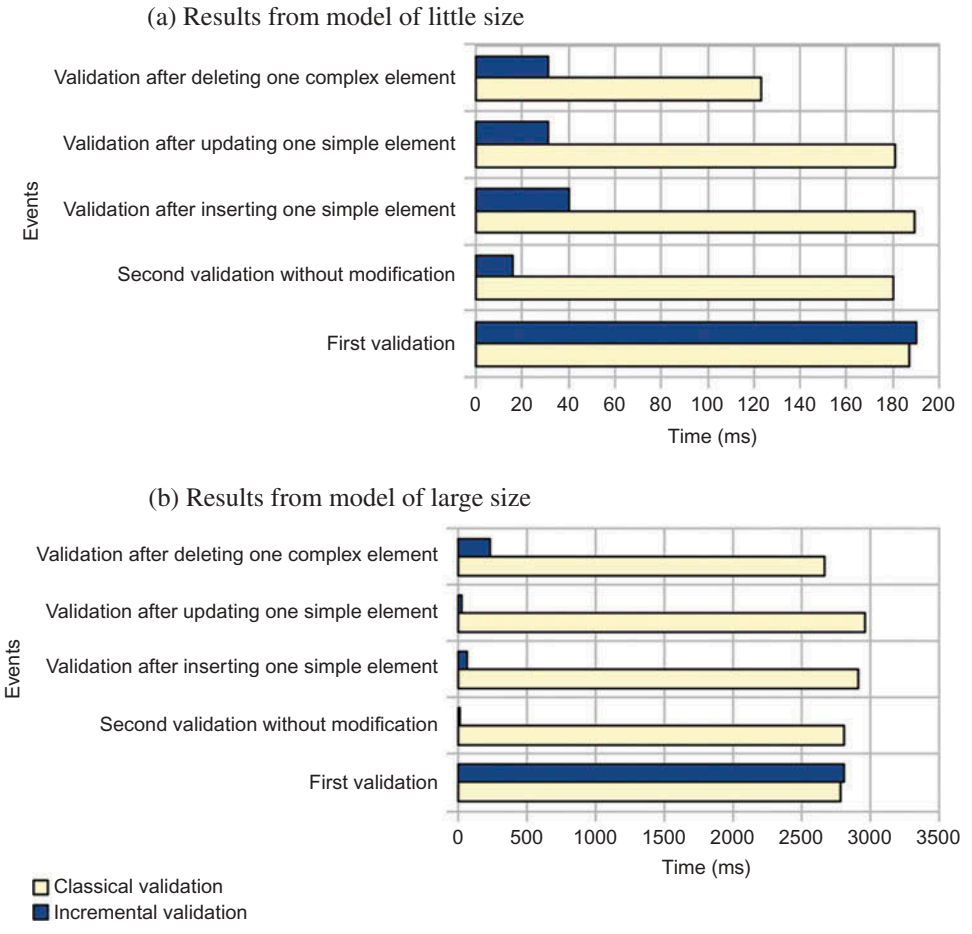

Figure 11. Global validation versus incremental validation results.

7. Conclusion and outlooks

The incremental model checking approach introduced in this paper is based on a formalisation of metamodel by mathematically handable objects. The goal of this formalisation is to express validation rules and structural consistency rules through DL formulae. To represent model sub-parts, which are potentially impacted during atomic modification, a graph representation is used to easily introduce the notion of context validation. Context validation allows the model parts to be checked and processed optimally. The next step determines which validation rules have to be checked on a given context. The solution considers that, for a given action, only one set of rules is checked. Indeed, a classification of rules to be checked makes possible the indication of rules that have to be validated, and which rules have not to be checked because they are not involved when a precise action occurs.

With this context mechanism, a methodology is offered to optimise the validation process taking into account the fact that only the modifications affecting a context type may lead to checking. One of the advantages of this approach is its ability to be applicable

on all types of data model, with the constraint of providing a formalisation of metamodel associated with models for checking.

Nevertheless, the methodology depends partially on an EBX platform. An enterprise business model represented in EBX uses specific EBX's notation (i.e. *osd:* XML namespace). Therefore, the transformation rules from an EBX instance to graph take into account the MDM profile, the XSD profile and the native EBX notation. The second limitation concerns the semantic business requirements. EBX integrates the integrity constraints and the management constraints. But, the incremental model checking processes only consider the structural events. What happens when a semantic business requirement is added, updated or deleted? It is necessary to define the notion of requirement context. Another limitation is the poor semantic of XS itself. It will be better if the EBX platform gives the possibility to directly model in RDF or ontology Web language (OWL).

In future works, this method will be improved by exploiting ontologies (Leenheer and Mens 2008; Pan et al. 2006) with the OWL (W3C OWL Working Group 2009) and the UML OWL profile (Brockmans et al. 2006). OWL has been designed to describe classes and relationships inherent to Web applications. OWL is used (i) to formalise a domain by defining classes, relationships and their properties, (ii) to define individuals and to assert their properties and (iii) to reason on classes and individuals within the scope of OWL semantics.

Graphs are used to answer these three issues. Ontology enables metamodels (domains) and relationships between entities (individuals) to be represented through a standard means. Exploiting modular ontologies is another way to split a model into sub-parts, which can integrate the validation context notion in the scope of ontology.

On the other hand, the method of incremental model validation in the scope of MDE does not integrate automatical resolution of errors raised during the validation process. The end-user has to correct the raised errors for each event manually. It will be interesting to calculate the set of corrective model modifications (Egyed 2007a), which allows the model to be corrected and revalidated in automatical manner. Finally, it is necessary to detect the validation rules being mutually excluded (Mens, Straeten, and D'Hondt 2006).

Notes

1. Object Management Group, http://www.omg.org.
2. Extraction Transformation Loading.
3. Enterprise Application Integration.
4. Enterprise Service Bus.
5. Download at http://www.iut.univ-paris8.fr/ lamolle/research/projects/buildArgoXSD.zip.
6. Cf. http://www.w3.org/TR/xmlschema11-1/ and http://www.w3.org/TR/xmlschema11-2/
7. Semantics of Business Vocabulary and Rules.
8. Acronym for *From Uml to Java And Back Again.*
9. http://www.omg.org/spec/SBVR/1.0/.

References

Abiteboul, S., S. Cluet, G. Ferran, and M. Rousset. 2002. "The xyleme project." *Computer Networks*, 39: 225–238.

Abrial, J. R. 1995. *The B Book: Assigning Programs to Meanings.* Cambridge, NY: Cambridge University Press.

Ameluxen, C., A. Königs, T. Rötschke, and A. Schürr. 2006. "MOFLON: A Standard-compliant metamodeling framework with graph transformations." In *Model Driven Architecture – Foundations and Applications, Second European Conference*, Vol. 4066 *of Lecture Notes in*

Computer Sciences (LNCS), ECMDA-FA, ed. A. Rensink Bilbao and J. Warmer, 361–375. Berlin: Springer.

Amyot, D., 2003. "Introduction to the user requirements notation: learning by example." *Computer Networks* 42 (3): 285–301.

Bahl, P., R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. 2007. "Towards highly reliable enterprise network services via inference of multi-level dependencies." *ACM SIGCOMM Computer Communication Review* 37 (4): 13–24.

Baisley, D. E., J. Hall, and D. Chapin. 2005. "Semantic formulations in SBVR." In *W3C Workshop on Rule Languages for Interoperability'05*, 1–5, W3C, 205, Washington, DC.

Bajwa, I. S., B. Bordbar, and M. Lee. 2011. "SBVR vs OCL: a comparative analysis of standards." In *14th IEEE International Multitopic Conference (INMIC 2011)*, 261–266. Karashi: IEEE Press.

Bardohl, R., M. Minas, A. Schürr, and G. Taentzer. 1999. "Vol. II: applications, languages and tools, application of graph transformation to visual languages." In *Handbook of Graph Grammars and Computing by Graph Transformation*, 105–180. Singapore: World Scientific.

Baril, X., and Z. Bellahsene. 2003. "XML data management: native XML and XML-enabled database systems." In *Designing and Managing an XML Warehouse*, 455–474. Reading: Addison Wesley Professional.

Bechhofer, S., R. Moller, and P. Crowther, 2003. "The DIG description logic interface." In *Proceedings of the 2003 Description Logic Workshop (DL 2003) CEUR Workshop Proceedings*, Rome, September.

Berners-Lee, T. 2001. "Conceptual graphs and the semantic web." Accessed November 14, 2012. http://www.w3.org/DesignIssues/CG.html

Bonnet, P., J.-P. Detavernier, and D. Vauquier. 2009. *Sustainable IT architecture – the progressive way of overhauling information systems with SOA*. Hoboken, NJ: Wiley, ISTE.

Brockmans, S., R. B. Colomb, E. F. Kendall, E. K. Wallace, C. Welty, and G. T. Xie. 2006. "A model driven approach for building OWL DL and OWL full ontologies." In *The Semantic Web – ISWC 2006*. Vol. 4273/2006 of *Lecture Notes in Computer Science*, 187–200. Heidelberg: Springer Berlin.

Cabot, J., R. Pau, and R. Raventos. 2010. "From UML/OCL to SBVR specifications: a challenging transformation information systems." *Information Systems* 35 (4): 417–440.

Cabot, J., and E. Teniente. 2009. "Incremental integrity checking of UML/OCL conceptual schemas." *Journal of Systems and Software* 82 (9): 1459–1478.

Chein, M., and M. L. Mugnier. 2008. *Graph-based representation and reasoning*. Advanced Information and Knowledge of Conceptual Processing Series. London: Springer.

Delobel, C., C. Reynaud, M.-C. Rousset, J.-P. Sirot, and D. Vodislav. 2003. "Semantic integration in xyleme: a uniform tree-based approach." *Data Knowledge Engineering* 44 (3): 267–298.

Egyed, A. 2007a. "Fixing inconsistencies in UML design models." In *Proceedings of the 29th International Conference on Software Engineering (ICSE'07)*, 292–301. Washington, DC: IEEE Computer Society.

Egyed, A. 2007b. "UML/analyzer: a tool for the instant consistency checking of UML models." In *Proceedings of the 29th International Conference on Software Engineering*, 793–796. Washington, DC: IEEE Computer Society.

Engels, G., R. Heckel, J. M. Küster, and L. Groenewegen. 2002. "Consistency-preserving model evolution through transformations." In *UML* 2002 – *The Unified Modeling Language*, 212–227. Heidelberg: Springer Berlin.

Esch, J., 1994. "Contexts, canons and coreferent types." In *Proceedings of the Second International Conference on Conceptual Structures (ICCS'94), LNAI*, ed. J. Dick, J. Sowa, and W. Tepfenhart, 185–195. London: Springer-Verlag.

Folli, A., and T. Mens. 2007. "Refactoring of UML models using AGG." *Journal of Electronic Communication of the European Association of Software Science and Technology (ECEASST)*. Vol. 8 of *ERCIM Symposium on Software Evolution*, Paris, October.

Fournier-Viger, P. 2005. "Unmodele de representation des connaissances a troisniveaux de sémantique pour les systemestutorielsintelligents." Thesis (PhD). Universite de Sherbrooke, Sherbrooke, QC.

Frankel, D. S. 2003. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Hoboken, NJ: Wiley.

Friedman-Hill, E. 2003. “Jess, the expert system shell for the java platform.” First Printed November 1997, SAND98-8206 (revised), Version 6.1RC1 (24 March 24, 2003). Livermore, CA: Distributed Computing Systems, Sandia National Laboratories.

Garcia-Molina, H., Y. Papakonstantinou, D. Quass, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. 1997. “The TSIMMIS approach to mediation: data models and languages.” *Journal of Intelligent Information Systems (JIIS) – Special Issue: Next Generation Information Technologies and Systems* 8 (2): 117–132.

Geiger, L., and Ziindorf, A. 2006. “Tool modeling with Fujaba.” *Electronic Notes in Theoretical Computer Science, Elsevier* 148 (1): 173–186.

Gerbé, O., M. Mineau, and R. Keller. LA metamodelisation et les graphes conceptuels. 2003. Technical report, rapport HEC-GRESI-03-02, Montroeal, ISSN 0832-7203.

Goedicke, M., T. Meyer, and G. Taentzer. 1999. “Viewpoint-oriented software development by distributed graph transformation: towards a basis for living with inconsistencies.” In *Proceedings of Fourth IEEE International Symposium on Requirements Engineering (RE'99)*, 92–99. Limerick: IEEE Computer Society, June.

Golfarelli, M., S. Rizzi, and B. Vrdoljak. 2001. “Data warehouse design from XML sources.” In *Proceedings of the 4th* ACM *International Workshop on Data Warehousing and OLAP*, *DOLAP'01*, 40–47. Atlanta, GA: ACM.

Gravier, S. 1997. “Coloration etProduits de graphes.” Thesis (PhD). University of Joseph Fourier, Grenoble.

GRL, ITU-T. 2006. *URN Focus Group: Draft Rec. Z.151 – Goal-oriented Requirement Language (GRL)*. Geneva

Haarslev, V., and R. Moller. 2001. “Description of the racer system and Its applications.” In *Proceedings of the International Workshop on Description Logics (DL-2001)*, 132–141. Stanford, CA: CEUR Workshop Proceedings, August.

Hayes, P., and B. McBride. 2004. *RDF Semantics*, W3C Technical Report. Accessed February 21, 2012. http://www.w3.org/TR/rdf-mt/.

Hermann, F., H. Ehrig, and G. Taentzer. 2008. “A typed attributed graph grammar with inheritance for the abstract syntax of UML class and sequence diagrams.” *Electronic Notes in Theoretical Computer Science (ENTCS)* 211: 261–269.

Jacobson, I., G. Booch, and J. Rumbaugh. 1999. *The Unified Software Development Process*. Addison-Wesley Object Technology Series. Indianapolis, IN: Addison Wesley Professional.

Leenheer, P. D., and T. Mens. 2008. “Vol. 7 of *Computing for human experience*, ontology management.” In *Ontology Evolution*, 131–176. Boston, MA: Springer US.

Lenzerini, M. 2002. “Data Integration: a theoretical perspective.” In *Proceedings of the Twenty-First* ACM *SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, *PODS'02*, 233–246. New York: ACM.

Levinson, R., and G. Ellis. 1992. “Multilevel hierarchical retrieval.” *Knowledge Based Systems*, 5 (3): 233–244.

Loshin, D. 2008. *Master Data Management*. The Morgan Kaufmann/OMG Press Series. Philadelphia, PA: Elsevier.

Mellor, S. J., and M. J. Balcer. 2002. *Executable UML: a foundation for model driven architecture*. Indianapolis, IN: Addison-Wesley Professional.

Menet, L., and M. Lamolle. 2008a. “Designing XML pivot models for master data integration via UML profile.” In *Proceedings of the Tenth International Conference on Enterprise Information Systems (ICEIS'08)*, ed. J. Cordeiro, and F. Joaquim, Vol. DISI, 461–464. Barcelona: INSTICC, June 12–16.

Menet, L., and M. Lamolle. 2008b. “Towards a bidirectional transformation of UML and XML models.” In *Proceedings of the International Conference on ELearning, E-Business, Enterprise Information System and E-Government (EEE'08)*, 461–464. Las Vegas, NV: CSREA Press, July 14–17.

Mens, T., R. V. D. Straeten, and M. D'Hondt. 2006. “Detecting and resolving model inconsistencies using transformation dependency analysis.” In *Proceedings of the International Conference MoDELS/UML*. Vol. 4199 of *Lecture Notes in Computer Science*. Berlin: Springer-Verlag.

Millan, T., L. T. Sabatier, P. Bazex, and C. Percebois. 2008. “NEPTUNE II une plateformepour la verification et la transformation de modeles.” *GenieLogiciel, GL et IS, numéro spécial,* 85: 30–34.

Nardi, D., and R. Brachman. 2003. "An introduction to description logics." In *The Description Logic Handbook: Theory, Implementation and Applications*, 544. Cambridge: Cambridge University Press.

Nassis, V., R. Rajugan, T. S. Dillon, and W. Rahayu. 2004. "Conceptual design of XML document warehouses." In *Proceedings of the 6th International Conference on Data Warehousing and Knowledge Discovery*. Vol. 3181/2004 of *LNCS* (DAWAK'04), 1–14. New York: ACM.

Nemuraite, L., T. Skersys, A. Sukys, E. Sinkevicius, and L. Ablonskis. 2010. "VETIS tool for editing and transforming SBVR business vocabularies and business rules into UML and OCL models." In *16th International Conference on Information and Software Technologies*, 377–384. Kaunas: Kaunas University of Technology.

Nentwich, C., W. Emmerich, A. Finkelstein, and E. Ellmer. 2003. "Flexible consistency checking." *ACM Transactions on Software Engineering and Methodology (TOSEM)* 12 (1): 28–63.

Olivé, A. 2007. *Conceptual Modeling of Information Systems*. Berlin Heidelberg: Software Engineering, Springer-Verlag.

OMG. 1997. "Unified modeling language." Accessed April 20, 2013. http://www.uml.org

OMG. 2002. "Response to the UML 2.0 OCL." Version 1.5, June 2002, 2.1–2.25. Accessed April 20, 2010. http://www.omg.org/docs/ad/02-05-09.pdf

OMG. 2006. "Meta object facility (MOF) specification." Accessed April 20, 2013. http://www.omg.org/mof/

OMG. 2007. "OMG modeling and metadata specifications, MOF 2.0, XMI mapping, v2.1.1." Accessed December 10, 2012. http://www.omg.org/spec/XMI/2.1.1/

OMG. 2008. "Semantics of business vocabulary and business rules (SBVR), Version 1.0." Accessed February 15, 2012. http://www.omg.org/spec/SBVR/1.0/

Pan, Y., G. Xie, L. Ma, Y. Yang, Z. M. Qiu, and J. Lee. 2006. "Model-Driven ontology engineering." *Journal of Data Semantics VII* 4243: 57–78.

Patel-Schneider, P., and I. Horrocks. 1999. "DLP and FaCT." In *TABLEAUX'99: Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, 19–23. London: Springer-Verlag.

Régnier-Pécastaing, F., M. Gabassi, and J. Finet. 2008. *MDM: Enjeux et méthodes de la gestion des donnees*. CollectionInfoPro – Management des Systemesd'InformationéditionsDunod.

Rozenberg, G. 1997. *Handbook of Grammars and Computing by Graph Transformation: Volume I. Foundations*. River Edge, NJ: World Scientific Publishing.

Russel, S., and P. Norvig. 2003. *Artificial Intelligence: A Modern Approach*. 2nd ed. Upper Saddle River, NJ: Prentice Hall.

Sabetzadeh, M., and S. Easterbrook. 2005. "An algebraic framework for merging incomplete and inconsistent views." *In Proceedings of the 13th IEEE International Requirements Engineering Conference (RE'05)*, 306–318. Washington, DC: IEEE Computer Society.

Schmedding, F., N. Sawas, and G. Lausen. 2007. "Adapting the rete-algorithm to evaluate F-logic rules." In *Advances in Rule Interchange and Applications*. Vol. 4824/2007 of *Lecture Notes in Computer Science*, 166–173. Heidelberg: Springer Berlin.

Simeon, J. 2000. "Data integration with XML: a solution for modern web applications." *Lecture at Temple University.* Accessed October 10, 2009.

Sirin, E., B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. 2007. "Pellet: a practical OWL-DL reasoner." *Web Semantics: Science, Services and Agents on the World Wide Web* 5 (2): 51–53.

Sourrouille, J. L., and G. Caplat. 2002. "Constraint checking in UML modeling." In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02)*, 217–224. New York: ACM.

Sowa, J. F. 2008. *Handbook of Knowledge Representation*, ed. F. van Harmelen, V. Lifschitz, and B. Porter. San Diego, CA: Elsevier B.V.

Spivey, J. M. 1992. *The Z Notation: A Reference Manual*. Upper Saddle River, NJ: Computer Sciences Prentice Hall.

Taentzer, G. 2004. "AGG: a graph transformation environment for modeling and validation of software." In *Applications of Graph Transformations with Industrial Relevance*, Vol. 3062/2004, 446–453. Heidelberg: Springer Berlin.

Thompson, H., D. Beech, M. Maloney, and N. Mendelsohn. 2004. *W3C Recommendation, XML-Schema Part 1: Structures*. 2nd ed. Accessed February 15, 2012. http://www.w3.org/TR/xmlschema-1

Tomasic, A., L. Raschid, and P. Valduriez. 1998. “Scaling access to heterogeneous data sources with DISCO.” *IEEE Transactions on Knowledge and Data Engineering* 10 (5): 808–823.

Tsarkov, D., and I. Horrocks. 2003. “DL reasoner vs. First-Order prover.” In *Proceedings of the 2003 Description Logic Workshop (DL'03) CEUR Workshop Proceedings*, Vol. 81, Rome, September, 152–159. Accessed February 15, 2012. http://CEUR-WS.org

Tsarkov, D., and I. Horrocks. 2006. “FaCT++ description logic reasoner: system description.” *In Automated Reasoning*. Vol. 4130/2006 of *Lecture Notes in Computer Science*, 292–297. Heidelberg: Springer Berlin, October.

UCM, ITU-T. 2003. *URN Focus Group: Draft Rec. Z.152 – UCM: Use Case Map Notation (UCM)*. Geneva. Accessed February 15, 2012. http://www.UseCaseMaps.org/urn/

URN, ITU-T. 2003. *Recommendation Z.150, User Requirements Notation (URN) – Language Requirements and Framework.* Geneva.

Varro, G., A. Schuorr, and D. Varro. 2005. “Benchmarking for graph transformation.” In *Proceedings IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, ed. Amber and Zhang, 79–88. Dallas, TX: IEEE Computer Society Press, September.

W3C OWL Working Group. 2009. “Web ontology language.” Accessed March 13, 2010. http://www.w3.org/TR/owl2-overview/

Wermelinger, M. 1995. “Conceptual graphs and First-Order logic.” In *Proceedings of the Third International Conference on Conceptual Structures (ICCS'1995), LNAI*, Vol. 954, ed. G. Ellis, R. Levinson, W. Rich, and J. Sowa, 323–337. Santa Cruz, CA: Springer-Verlag