# 8. UEML: a Further Step

Giuseppe Berio
*Dipartimento di Informatica Università di Torino berio@di.unito.it*

*This paper presents a further step towards a UEML (Unified Enterprise Modelling Language) starting from the result of the UEML project, funded by the European Commission under the IST-V$^{th}$ Framework Programme of Research. Specifically, the paper provides the basic theories and thinking underlying the project work as well as current improvements based on a data-integration perspective.*

## 1. INTRODUCTION

Many problems raising in *Enterprise Integration* (EI) and *Enterprise Engineering* (EE) are certainly due to the fact that there are many *Enterprise Modelling Languages* (EMLs) and *Enterprise Modelling Tools* (EMTs), spanning from industrial analysis, management analysis, strategic planning, human management, budget and so on. Probably, the reasons are: because there is no one language covering all the aspects required for modelling and on the other hand, each EMT with the specific language is able to perform specific analysis or have more or less direct link to *enterprise software tools (i.e. enterprise software applications).* The major need facing to this situation is probably an environment in which both *enterprise models* and *enterprise tools* can be integrated. In this way, integrated models have more chance to be used and maintained in a consistent ways. However, while *integration of tools* is usually perceived as really useful in practice and feasible, *integration of models*[45] is the major challenge because focusing on the integration of the *content of models.*

The idea of a UEML (*Unified Enterprise Modelling Language*) for improving the situation described above was born in the context of *ICEIMT initiatives* (Petit *et al.*, 1997), further advocated and explored in recent papers, e.g. (Vernadat, 2002). However, the *first project* on such a UEML started in 2002, funded by the *European Commission* under the *IST-V$^{th}$ Framework Programme of Research.*

This paper reformulates the approach undertaken in the UEML project in term of *data-integration* (Calvanese, *et al.*, 2002, 2003) In fact, results of the UEML project are really close to some data-integration approaches. Then, the paper presents a possible further step towards the introduction of *Enterprise Reference Architectures* (ERAs) (ISO 1998, IFAC-IFIP Task Force, 1999). The interest of ERAs is to represent a coherent set of distinct modelling purposes for the same language (such

[45] In this paper integration of models comprises the *exchange of models* between distinct EMTs.

as a UEML). With ERAs, it is therefore possible to differentiate between the language and its various purposes.

The paper is organised as follow. Section 2 describes problems to be approached by a UEML. Section 3 summarises the foundations of a UEML as defined in the UEML project: the section especially describes the links between modelling languages and databases. Section 4 provides an overview about data-integration approaches; it also describes some simple examples. Section 5 describes how the approach undertaken in the UEML project can be reformulated in term of data-integration. Section 6 describes how data-integration approaches allow to easily introduce ERAs with a UEML. Finally, section 7 summarises the contributions of this paper.

## 2. UEML: PROBLEMS

The state of the art (Petit *et al.*, 2002b) issued from the UEML project reveals that distinct tools and languages are required because they can be used for achieving, probably in the best way, specific objectives (i.e. UEML should not substitute existing languages: though, UEML should be focused on model integration);

- most of the languages for enterprise modelling are not formalised in their semantics (i.e. the semantics is not "mathematically" described) but they are in their syntax i.e. it is known what is a model and what is not a model but it is less known if the model is meaningful or not;
- this lack of semantics is managed by a correct understanding and usage of modelling methodologies (methods) which allows to make right models;
- some semantics is added to models under specific purposes (for instance, simulation of models) but this semantics is not explicitly stated (e.g. it is part of simulation tools);
- it is very difficult and probably impossible to provide one formal semantics which is good for every purpose;
- enterprise models are intended for a broad usage even by humans; models can be used for teaching how the work should be, what an enterprise is, how it evolves, why it evolves, how it can be improved and so on.

As a consequence, the first problem to be approached by a UEML is that *models made in languages with no semantics* or an *informal semantics* or *hidden semantics*, are *more or less free of interpretation*. The second problem is that, having some *formal semantics* for languages and models is not enough. In fact, two distinct models with the same *mathematical semantics* (i.e. formal) may be partially equivalent in term of the *real world phenomena they represent*. A practical test for understanding this problem is a *process with just three activities*: mathematically, it may mean that there are three *activities in sequence*. However, two employees looking to this process may interpret it in distinct ways, just because the *names of the activities* suggest that the right interpretation *is not a sequence* but it is a *sequence of requests* that may be not fulfilled. There, w*here is the semantics* (Ushlod, 2003)? Part of the semantics is likely to be in the names of the activities which can only be interpreted by the employees (because of their knowledge). Therefore, given an enterprise model, its *context of interpretation*, mainly distinguished in *machines* and *humans*, still remains an important aspect.

Researches about *ontologies* (Gruber, 1993, Guarino (Ed.), 1998) try to take into account the context of interpretation of a model both for humans and machines. In fact, this is part of the following definition: "*an ontology is a specification of a conceptualisation shared by a community*". Much of the *power of ontologies* is in the fact every ontology specification should be "shared by a community". However, what does "sharing by a community" mean? What is a "community" and how is it organised? Interesting examples come from the *laws genesis*. You have who defines the laws, you have structures able to interpret the laws in the *context, at different levels*, in a *continuous cycle*: however, contradictions are possible and acceptable. Therefore, generally speaking, *full sharing* by a *community of humans* seems to be rather difficult to be achieved. However, we may constrain as much as possible the conceptual domains and try to achieve full sharing for a very limited number of concepts. Nevertheless, it is not clear if the complexity is in the number of concepts or is in the inherent complexity of the concepts (Corrrea da Silva, *et al.*).

Table 1: Analogies between modelling languages and databases

| ***Database Glossary*** | **Modelling Language Glossary** | **Samples of Model, Model artefacts and Meta-model** |
|---|---|---|
| Data (instances) | Model artefacts | "Mount", "Employee", "Robin" |
| Database (a set of related instances) | Model (a set of related model artefacts)<br>Models (a set of models) | {"Mount", "Dismount", "Mechanical Part", "Employee", "Robin", "Edgar"} |
| Schema | Meta-model (representing the way for providing an *abstract syntax of a language*) | Activity, ObjectClass, Role, Object |
| Integrated schema | Integrated meta-model | Activity, ObjectClass, Role, Object |

The UEML project team has performed a study concerning possible solutions to the problems mentioned above. On one hand, this study especially recognises that the interpretation of an *enterprise model* is provided by its *context of interpretation* (machines or humans). Whenever there are various contexts of interpretation and various models, an *integrated enterprise model* (better defined in the remainder) is a way to guarantee a *consistent usage of distinct models within distinct contexts*. To make integrated enterprise models, a UEML is a prerequisite. On the other hand, the study recognises the fact that *extensive formalisations of semantics* of a UEML is probably not a key point towards effective solutions to model integration. Some of the reasons are technical (Berio, Petit, 2003). In fact, if it is needed to *formally check a property*, the techniques for checking this property are really various and based on distinct formalisations (because *incompleteness* of some formalisations or *complexity* of the checking technique). Thus, while formalisations are needed for proving if an integrated model makes sense (i.e. no contradiction occurs), satisfies some properties, and they might also be useful for driving the model integration, they do not allow to infer how model integration should be performed and what an integrated model should be.

## 3. UEML: FOUNDATIONS

Table 1 states some analogies between databases and modelling languages. These analogies suggest that within the *database area*, *database integration*, *database architectures* and *schema integration techniques*, have already approached the two problems that should also be approached by a UEML.

These analogies can be stated because a syntax of a EML has not meaning per se: its meaning is given elsewhere (i.e. by the context of interpretation). As data, a syntax of a EML is interpreted by users or software components in various ways: some of these ways are correct, some ones are not; some ones are mutually consistent, some ones are contradictory. The main difference between databases and languages is probably that databases have a narrow scope than languages.

## 4. DATA-INTEGRATION

The objective of this section is not to present data-integration *per se*: though, this section describes some important aspects which are required for understanding the contributions of this paper to the UEML development.

*Data-integration* (Calvanese, *et al.*, 2002, 2003) provides the theoretical base for *database integration. Database integration is much more than the well known classical schema integration* because it directly works on data: in fact, schema-integration and data-integration may refer to *distinct phases* of the *lifecycle*, respectively *design* and *implementation*. Within data-integration, a *query result* is based on *data currently stored in several databases*. Data-integration is very relevant whenever these databases are *highly autonomous* both at schema and data levels. There are *four approaches* to data-integration:

- GAV (global as view),
- LAV (local as view),
- GLAV (generalisation of GAV and LAV),
- P2P (peer to peer).

Apart the P2P approach, the other ones need an explicit *integrated schema* (i.e. schema integration is a prerequisite to data-integration).The LAV and GLAV approaches explicitly acknowledge that it is not possible *to integrate distinct databases* but these distinct databases (qualified as local) can be understood as *views* on a *global database* (and also *vice-versa* in the GLAV approach). In other words, they do not provide effective ways for directly making *one integrated database*, i.e. a database which represents some *equivalence of data* stored in the various databases (as we will see in the remainder). Therefore, Table 1 can be completed by the following analogy:

| ***Database Glossary*** | **Modelling Language Glossary** |
|---|---|
| Integrated database | Integrated model |

being an *integrated model* defined as a model where two *distinct model artefacts* should never be *equivalent* (in term of the real world phenomena they represent).

Because of their relevance to the UEML, two important aspects concerning data-integration need to be carefully analysed:

- differences with schema integration,
- mappings.

Both aspects are discussed using some simple examples which are also useful in other sections of the paper.

Let suppose to have *three relational schema* (two qualified as local and one qualified as integrated) containing respectively three tables (T1, T2, T3), and *three databases* (two qualified as local and one qualified as global) (Figure 1). In the integrated schema, we have deliberately added the column *Database* to clearly show differences with schema integration.

Following the *LAV approach*, it is possible to represent *mappings* between one database and another one, by using any available *query language* and schema information. In the example of Figure 1 below, *mappings* may be as follows:

*Select(T2(OrderN, Supplier Name) where database=1) = T1(OrderCode, Supplier);*

*Select(T2(OrderN, Supplier Name) where database=2) = T3(Order, Supplier ID).*

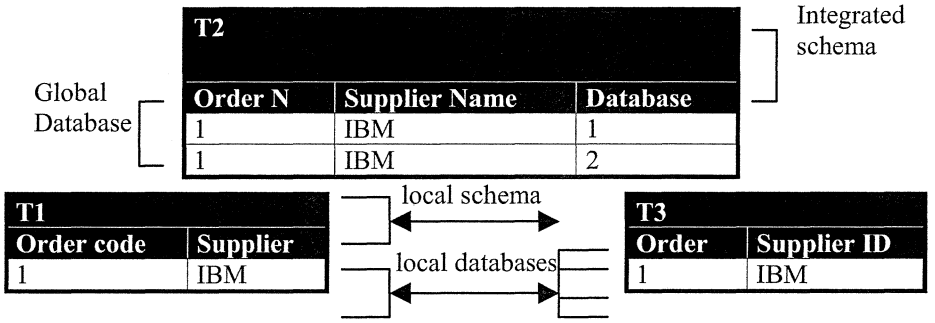


Figure 1: Example of data-integration

In general, the LAV approach allows:

- given one global database, *to derive* the local databases; i.e. mappings can be used as kind of *export mechanism*;
- given the local databases *to partially derive* a global database; i.e. mappings can be used as kind of *import mechanism*.

In the example of Figure 1, for instance, the export mechanism allows to derive data in T1 by applying the related query (because of the equivalence "="). The import mechanism is more interesting. In fact, the equivalence "=" in the mappings makes possible to fully derive the global database from the local ones: however, from a strictly theoretical point of view, this derived global database is only one of the possible global databases (for instance, in the global database we may freely add data unrelated with the two local databases i.e. with *database<>1,2*). In the general case of LAV (and GLAV), the '=' can be a *set inclusion* '$\supseteq$'.

In the *GLAV approach* both import and export mechanisms are partial. To illustrate this point, the previous example is further extended to the GLAV approach by introducing the couple of mappings below:

*Select(T2(OrderN, Supplier Name) where database=1) =*
*SomeQuery1(T1(OrderCode, Supplier))*
*Select(T2(OrderN, Supplier Name) where database=2) =*
*SomeQuery2(/T3(Order, Supplier ID)).*

where *SomeQuery1* and *SomeQuery2* indicate some queries involving available tables and columns (for instance, *SomeQuery1* may be *Select(T1(OrderCode, Supplier) where Supplier=IBM)*).

With these set of mappings, on one hand, we may freely add data to the global database (as before) and, on the other hand, we can add data to the local databases which do not satisfy both SomeQuery1 and SomeQuery2 respectively (for instance, orders of any supplier which is not IBM).

Now, the field Database has deliberately been added: this allows to manage situations in which we do not know how much data in the local databases are related (e.g. referring to the example in Figure 1, if the two orders represent the same order) but we are able to integrate, at some extent, the schema. In this sense, the integrated schema in Figure 1 might be "correct" but while "IBM is a supplier", the meaning of orders might not be the same: "order 1" in T1 could be "order to supplier", "order 1" in T3 could be "order from supplier". Which would be the meaning of the table T2 in the integrated schema? T2 would be a *generic relationship* between *order* and *supplier* with specific interpretation in specific contexts (i.e. local databases): thus, the column Database takes into account these contexts of interpretation for the same generic relationship. The previous examples show that while local schema have been integrated, local databases remain distinct in the global database: if it is known the Database 1 only contains "orders to supplier" and Database 2 only contains "orders from supplier", the global database is also an integrated database.

So far, the global database depicted in Figure 1 still distinguishes between the two orders stored respectively in T1 and T3. It is possible that after some extensive analysis, it is decided (by contexts of interpretation) that there is a final *integrated database* in which these two orders have been compared and their equivalence has finally been stated.

The *equivalence of the two orders* can be represented within the *GLAV approach* by a couple of mappings involving an *integrated database* (numbered as 3 on Figure 2) in which equivalent data are never replicated:

*Select(T2(OrderN, Supplier Name) where database=3) =*
*SomeQuery1(T1(OrderCode, Supplier));*
*Select(T2(OrderN, Supplier Name) where database=3) =*
*SomeQuery2(/T3(Order, Supplier ID)).*

Therefore, the table T2 in the integrated schema is representing the *same relationship* which exists in the two local schema.

Integrated database

| T2 | | |
|---|---|---|
| **Order N** | **Supplier Name** | **Database** |
| 1 | IBM | 3 |

| T1 | |
|---|---|
| **Order code** | **Supplier** |
| 1 | IBM |

| T3 | |
|---|---|
| **Order** | **Supplier ID** |
| 1 | IBM |

Figure 2: Example of integrated database

As it can be noted, the same *integrated schema* (i.e. T2) is able to "host" both (the interesting data of) the local databases and *integrated databases*. Therefore, the following GLAV mappings *without the Database column* generalises the situations depicted in Figures1 and 2:

*Select(T2(Order N, Supplier Name)) ⊇ SomeQuery1(T1(OrderCode, Supplier));*
*Select(T2(Order N, Supplier Name)) ⊇ SomeQuery2(T3(Order, Supplier ID)).*

The couple of mappings above which does not differentiate between situations depicted in Figures 1 and 2, can be specialised if some information concerning the contexts of interpretation suggest that some data(bases) should never be integrated (as the case depicted in Figure 1). On the other hand, integrated databases cannot be inferred from the two local databases by using this last couple of mappings: additional external information is eventually required.

## 5. UEML AND DATA-INTEGRATION

The UEML project states that at least two main components should be developed for a UEML:

- An integrated meta-model called *UEML meta-model* which should be able to accommodate both models to be integrated and results of integration; however, why and how to integrate models is provided elsewhere;
- A *set of mappings* which relates *meta-models of EMLs* (called *originating meta-models*) to the UEML meta-model, and *traces* how meta-models of EMLs contribute to the UEML meta-model.
- The *UEML set of mappings* is characterised by two facts:
- Mappings should be *stable* i.e. they should remain valid across specific situations;
- Mappings should be *standardised.*

In the UEML project, three (originating) EMLs, IEM (Jochem and Mertins, 1999), EEML (External, 2000) and GRAI/Actigrams (Doumeingts, *et al.*, 1992, Doumeingts, *et al.*, 1998), have been selected for making a first version of a UEML named *UEML 1.0*. An *integrated meta-model* has been defined by following some steps (Berio *et al.*, 2003) which are based on database schema integration suggestions (Petit, 2002a). The key point is the usage of a *scenario* (i.e. a set of models, analogous to databases in Table 1, in which it is possible to recognise equivalent model artefacts belonging to the distinct models). Specifically, given two models (part of the scenario) represented in two modelling languages, the first step

is to make *meta-models* (as UML class models (OMG, 2002b)) corresponding to the *abstract syntax* of these *modelling languages*. Then, *model artefacts* are explicitly related with the *meta-model artefacts* (e.g. it should be clear that the model artefact "produce" is related to the meta-model artefact "Activity" because the former is "instance" of the latter).

As explained in section 4, the data-integration approaches take into consideration both schema integration (as a prerequisite) and mappings between databases. Therefore, in (Berio *et al.*, 2004) the steps for building the two components mentioned above have been reformulated in term of data-integration by using the *P2P and GLAV approaches*. The resulting rule for building an *integrated meta-model* and *a set of mappings* is as follow:

Given two meta-models artefacts, C and K, belonging to two distinct EMLs respectively, if there exist two predicates Query1 and Query2 (expressed, for instance, in OCL (OMG 2002a) if meta-models are represented in UML) such that $Query1(C) \supseteq Query2(K)$ *or* $Query1(C) = Query2(K)$ are true according to the *scenario*, then a *meta-model artefact H* is introduced in the *UEML meta-model*;

H satisfies the two *GLAV mappings*

$Select(H) \supseteq Query1(C);$
$Select(H) \supseteq Query2(K).$

The next example (Figure 3 below) is very similar to the previous ones on databases. It is however based on "names" often found in EMLs and the rule provided above (it should be noted that, even possible, we do not add any identifier that may be used to identify distinct model artefacts inside each model).

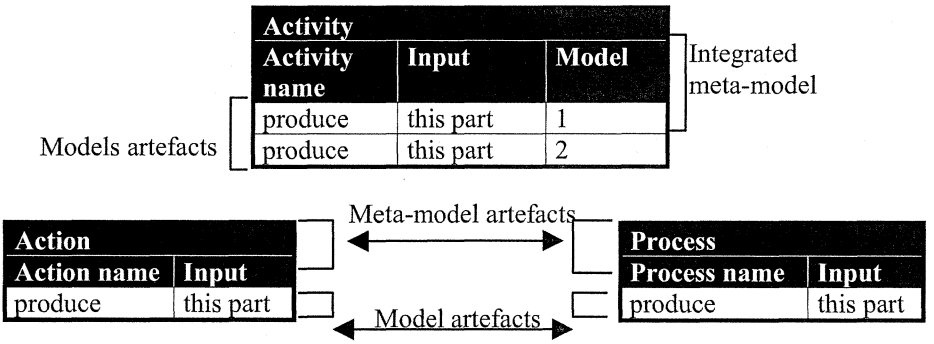


Figure 3: Example of integrated meta-model

A set of GLAV mappings is:

$Select(Activity(Activity\ Name, Input)) \supseteq$
$SomeQuery1(Action(Action\ Name, Input));$
$Select(Activity(Activity\ Name, Input)) \supseteq$
$SomeQuery2(Process(Process\ Name, Input))$

# 6. UEML AND ENTERPRISE REFERENCE ARCHITECTURES

The column *Model* on Figure 3 is a generic way to maintain the context of interpretation of models as in previous examples with the *Database* column. As explained in section 4, any information about the context of interpretation is important to differentiate data in a set of databases (then to differentiate model artefacts in a set of models). A possible generalisation simply states that *meta-model artefacts* should be related to an *Enterprise Reference Architecture*. In fact, ERAs concepts can be used for representing information about contexts of interpretation. For instance, concepts as *lifecycle (phases)*, *life history, enterprise entity type* and *modelling framework* (with views) within the *GERA nomenclature* (IFAC-IFIP Task Force, 1999), are really useful for distinguishing if a generic meta-model artefact such as *Activity* is referring to processes built in the requirement phase. Other concepts, such as *domains, layers, usage context*, may also be introduced.

A UEML meta-model and an ERA can be related by using some kind of *projection* of *meta-model artefacts* (e.g. *Activity*) onto the *ERA* (instead of using additional columns as in the examples about databases in section 4). For instance, we can project *Activity* that we are assuming part of the *UEML meta-model*, to phases such as *requirement* or *design*. These projections of *Activity* state that one meta-model artefact (i.e. *Activity*) is used according to the specifically related ERA concepts.

These projections essentially make *copies of the specified UEML meta-models artefact*. This also means that one *meta-model artefact can be projected several times*. The idea underlying copies is to make explicit as much as possible that the same meta-model artefact (e.g. *Activity*) whenever used for distinct purposes (i.e. distinct contexts of interpretation) also requires distinct model artefacts. For instance, *Activity* whenever projected onto *requirement* and *design* phases, makes two copies of *Activity* itself (possibly renamed) which should not share model artefacts but they are the same in term of the original meta-model artefact.

Copies of a meta-model artefact can be used to define *new mappings* which should be consistent with the previously stated (i.e. part of the set of GLAV mappings associated to a UEML). New mappings are useful to represent specific situations and they should represent *explicit decisions* on how to use languages (such as a UEML) inside a given ERA. The *consistency condition* is:

If $Query1(C) \supseteq Query2(K)$ is in the *UEML set of GLAV mappings* and
$Query3(H) \supseteq Query4(K)$ or $Query3(H) = Query4(K)$ is the *additional set of mappings* involving copies,
Then $H=Projection(C;...)$ (i.e. H should be a copy of C)

The consistency condition is represented by the diagram below (Figure 4).

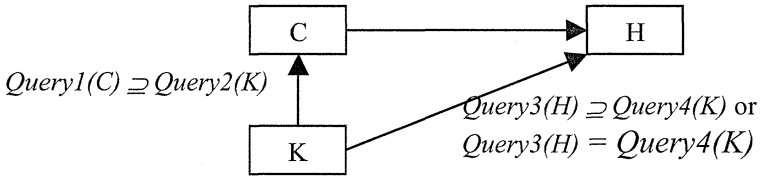


Figure 4: Consistency condition on set of mappings involving copies

referring to the GLAV mappings corresponding to the example shown in Figure 3, if the following projections are defined (represented as arrows and boxes, still named as *Activity* on Figure 5 below):

- **Projection(Activity; Design, Process View)** defining a copy of Activity which is used in the design phase for modelling processes;
- **Projection(Activity; Requirement)** defining a copy of Activity which is used in the requirement phase;
- it is not allowed to map Action onto any copy of Object.

Copies can explicitly be related by using *new relationships*: in fact, the GLAV approach allows to define new relationships in the integrated schema that do not appear in the local databases and schema. These new relationships are really relevant in case of modelling languages because they allow to *situate models* in *specific contexts of interpretation*. For instance, if there are two languages, one for representing (aspects of) *processes* and the other one for representing (aspects of) *services*, a new relationship might be introduced between *process* and *services* with the following meaning: "*process delivers service*". These new relationships might also be due to *methodologies* used with ERAs. This point is shared with the *method engineering discipline* (Brinkkemper, *et al.*, 1999).

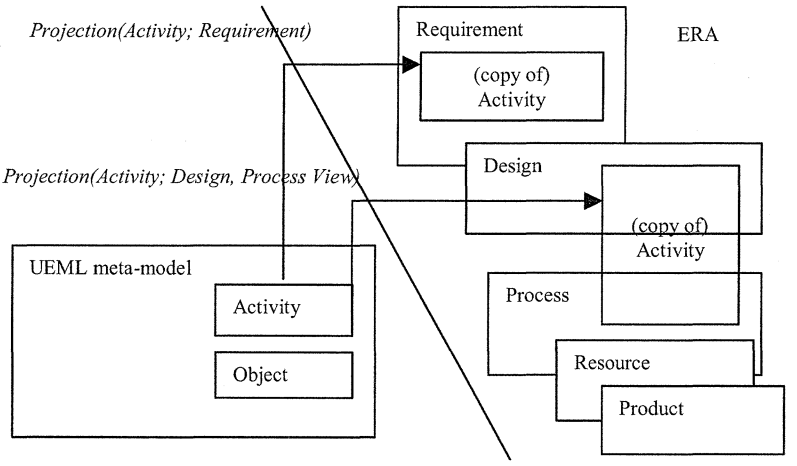


Figure 5: Example of UEML with ERA

## 7. CONCLUSIONS

This paper discusses how the perspective of *data-integration* can be used for improving the approach undertaken in the UEML project to a UEML development. The data integration perspective allows:

- To make clearer the distinction between *integrated meta-model* (analogous to integrated schema) and *integrated model* (analogous to integrated database) (sections 3 and 4);
- To clarify the role of *mappings* between a UEML and other EMLs as
  - mechanisms to trace how a UEML meta-model has been generated from meta-models of other EMLs (which is often not traced in classical schema integration) (see Section 5);
  - mechanisms to help model integration (section 4);
- To provide a base for taking into account information provided by ERAs about the *contexts of interpretation* of enterprise models (by re-using the mechanisms of mappings); in this way, model integration can become safer because model artefacts are much better differentiated (section 6).

**Acknowledgement**

The author would like to thank all the UEML IST-2001-34229 core members for their scientific contribution to the work. This work is partially supported by the Commission of the European Communities under the sixth framework programme (INTEROP Network of Excellence, Contract N° 508011, <http://www.interop-noe.org>).

## 8. REFERENCES

Berio, G. Anaya Fons, V. Ortiz Bas, A. (2004). Supporting Enterprise Integration through (a) UEML. To appear in Proceedings of EMOI Workshop joint with CaiSE04 – Riga – Latvia – June 2004.

Berio, G. Petit, M. (2003). Enterprise Modelling and the UML: (sometimes) a conflict without a case. In Proc. of Concurrent Engineering Conference 03, July 26-30, Madeira Island, Portugal.

Berio, G. *et al.* (2003). D3.1: Definition of UEML – UEML project, IST–2001-34229, www.ueml.org.

Brinkkemper, S. Saeki, M. and Harmsen, F. (1999). Meta-modelling based assebly techniques for situational method engineering, Information Systems, Vol 24, N.3, pp.209-228.

Calvanese, D. De Giacomo, G. Lenzerini, M. (2002). Description logics for information integration. In Computational Logic: From Logic Programming into the Future (In honour of Bob Kowalski), Lecture Notes in Computer Science. SpringerVerlag

Calvanese, D. Damaggio, E. De Giacomo, G. Lenzerini, M. and Rosati, R. (2003). Semantic data-integration in P2P systems. In Proc. of the Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing.

Corrrea da Silva, F. Vasconcelos, W. Robertson, D. Brilhante, V. de Melo, A. Finger, M. Augusti, J.. On the insufficiency of Ontologies. Knowledge Based Systems Journal.

Doumeingts, G. Vallespir, B. and Chen, D. (1998). Decision modelling GRAI grid. In, P. Bernus, K. Mertins, G. Schmidt (Eds.) Handbook on architecture for Information Systems, Springer-Verlag.

Doumeingts, G. Vallespir, B. Zanettin, B. and Chen, D. (1992). GIM, GRAI Integrated Methodology - A methodology for designing CIM systems, Version 1.0, Unnumbered report, LAP/GRAI, University of Bordeaux 1, France.

DoW (2002). Description of the Work – UEML project, IST–2001-34229, www.ueml.org.

EXTERNAL (2000). Extended Enterprise Resources, Networks and Learning, External project, IST- 1999-10091.

Gruber, T.R. (1993). A translation approach to portable ontologies. Knowledge Acquisition, vol. 5, no. 2, 199-220.

Guarino, N. (Ed.) (1998). Formal Ontology in Information Systesm. IOS Press. Amsterdam.

IFAC-IFIP Task Force (1999). GERAM: Generalised Enterprise Reference Architecture and Methodology, Version 1.6.

ISO (1995). ODP-Open Distributed Processing. ISO/IEC 10746.

ISO (1998). ISO DIS 15 704 Requirements for Enterprise Reference Architectures and Methodologies, ISO TC 184/SC5/WG1.

Jochem, R. (2002). Common representation through UEML – requirement and approach. In proceedings of ICEIMT 2002 (Kosanke K., Jochem R., Nell J., Ortiz Bas A. (Eds.)),, April 24-26 Polytechnic Univeristy of Valencia, Valencia, Spain, Kluwer. IFIP TC 5/WG5.12.

Jochem, R. Mertins, K. (1999). Quality-Oriented Design of Business Processes. Kluwer, Boston.

OMG (2002a). OCL 2.0 specification, www.omg.org.

OMG (2002b). UML 1.5 specification, www.uml.org.

Petit, M. (2002a). Some methodological clues for defining a UEML. In Proc. of ICEIMT 2002 (Kosanke K., Jochem R., Nell J., Ortiz Bas A. (Eds.)), April 24-26 Polytechnic University of Valencia, Valencia, Spain, Kluwer. IFIP TC 5/WG5.12.

Petit, M. *et al.* (2002b). D1.1: State of the Art in Enterprise Modelling, UEML-IST–2001-34229, www.ueml.org.

Petit, M. Gossenaert, J. Gruninger, M. Nell, J.G. and Vernadat, F. (1997). Formal Semantics of Enterprise Models. In K.Kosanke and J.G Nell. (Eds.), Enterprise Engineering and Integration, Springer-Verlag.

Ushlod, M. (2003) Where are the semantics in the semantic web? AI Magazine 24(3).

Vernadat, F. (2002). UEML: Towards a unified enterprise modelling language, International Journal of Production Research, 40 (17), pp. 4309-4321.