A Model-based Ontology of the Software Interoperability Problem: Preliminary Results.

Vincent Rosener, Thibaud Latour, Eric Dubois

Centre de Recherche Public Henri Tudor 29, avenue J.F. Kennedy L-1855 Luxembourg
vincent.rosener@tudor.lu, thibaud.latour@tudor.lu, eric.dubois@tudor.lu

Abstract. Interoperability usually refers to software system communication. Although there is no widely accepted definition, and therefore no common understanding of the context, there are multiple solutions (protocols, architectures, components) that promise to solve integration issues. The INTEROP network of excellence aims at proposing a large view of interoperability issues, and hence requires a unified definition. As an INTEROP participant, we suggest in this paper, as a first attempt, an ontology of interoperability. We first present the general software engineering concepts our work is based on. We then propose the decisional meta-model and the technical aspects meta-model, as prerequisite to the introduction of the actual interoperability model. Finally, we discuss the pros and cons, as well as different ways the model can be used. **Keywords**: Interoperability, model, ontology, software development, decision-aid.

1 Introduction.

Today, interoperability is widely recognized as a critical aspect in many businesses. It is caused by several factors that can less and less be segregated from the business itself: outsourcing, merge, acquisition and cost cuttings in IT typically let appear an issue of interoperability.

There are many solutions proposed to tackle this issue, from dedicated architectures to Commercial Of-The-Shelf (COTS) components. But beside those technical answers, it seems to us that there is a lack of formalization of the problem itself. Indeed, interoperability remains a vague concept, leaving many open questions: What is the nature of the systems that should interoperate? Are they only related to software, or also maybe to business? Is interoperability only relevant to executable systems, or also when considering higher level of abstraction? What is the exact scope of interoperability (is communication typically part of it, or a cause of it)? Is it possible to ensure interoperability at early stages of the software development? Is interoperability an independent problem in software development, can it be treated separately from all other technical issues? In this paper, we shall discuss our preliminary proposal regarding an ontology of interoperability, which gives a roadmap to answer these questions.

This work is a contribution to the work program of the FP6-IST INTEROP Network of Excellence (NoE) Joint Research Activities [1]. INTEROP scientific objectives originate from the vision and roadmap for interoperability of enterprise software established as an outcome of the FP5 Thematic Network IDEAS [2] (Interoperability Development for Enterprise Application and Software) and from work done around the UEML (Unified Enterprise Modeling Language) [3].
Far beyond past approaches specifically centered on Information Technologies (IT) issues, IDEAS developed an interoperability framework integrating both business and knowledge as the drivers and requirement substratum of integration in addition to traditional IT-based (and hence solution-centric) view of the problem. From this, it has been claimed that interoperability aspects pertaining to Enterprise Modeling, Architecture and Platform, and Ontology, should be considered together in order to obtain substantial and effective results as well as pragmatic applications in today business.

With respect to this context and in line with the "Ontology-based integration of enterprise modeling and architecture & Platform", we propose a first overview of a more formal definition of the interoperability general problem and of its possible solutions. Such an ontology is expected to help reaching a common understanding of the various contributions in this field, as well as limiting potential ambiguity. For validation purpose, we have adopted a wide and abstract perspective of the interoperability problem. Nevertheless we have always kept in mind that the final goal was the application of this model by software engineers in their activities.

The paper is structured as follows: we first present the general software and modeling concepts our work is built upon. Then, as a basis for the discussion of the proposed model-based ontology, Section 3 also introduces two important meta-models: the decisional meta-model and the technical aspects meta-model. Central in this paper is Section 4 where is presented our interoperability model. Finally we conclude with a discussion of the pros and cons, as well as the different ways according to which this model can be used.

2 General software engineering and modeling concepts

As it has already been stated, interoperability is a complex topic. In order to address the problem of building an ontology to describe the interoperability problem in a consistent and systematic way, some basic assumptions must be made which position our work in the context of Software Engineering (SE) and of a systemic approach of the problem. Figure 1 illustrates the main concepts that define such a context for our ontology. In our view, an interoperability problem may arise at a certain number of steps in the SE process. It may happen at a high-level software architecture step or at a low-level implementation step. In addition, it fundamentally consists in finding a suitable solution for solving a particular problem, which happens to be of technical nature[1]. The identification of the problem as well as its solution will depend on a large number of elements, among which SE models and meta-models are important ones.

[1] Interoperability issues at business level which do not impact IT solutions are not considered here

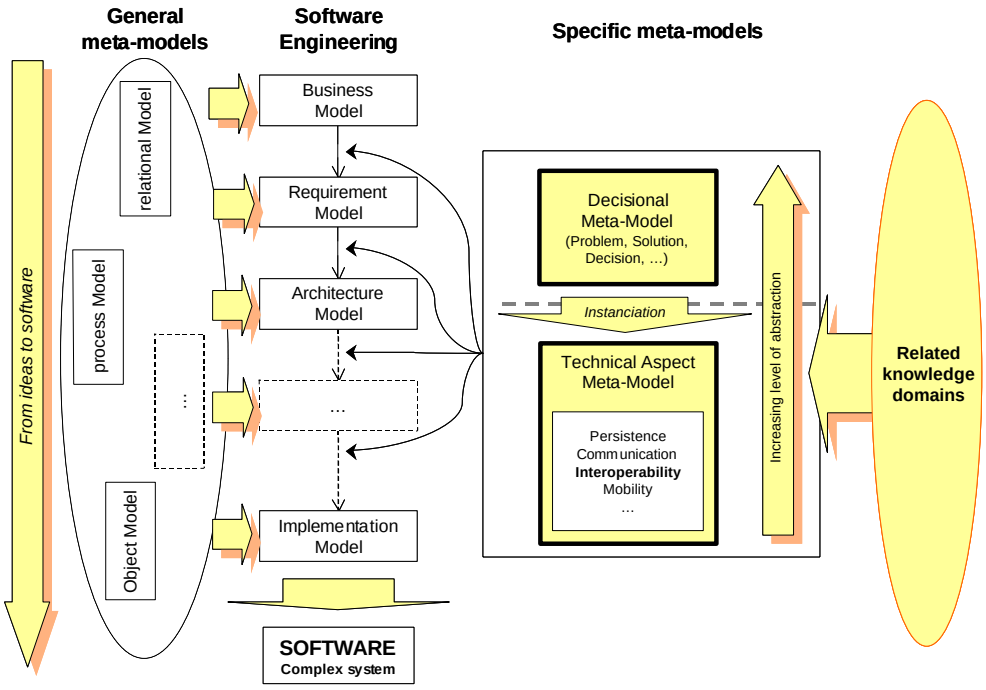

Fig. 1. General overview of Software Engineering and of its interconnections with our systemic approach

The next two sections will be devoted to the presentation of these elements driving the instantiation of our interoperability ontology.

2.1 Software development is based on models

Nowadays software products are complex systems. Therefore, the classical approach to software development that separates the phases in analysis, design, implementation and testing (even in an iterative process) is probably obsolete. There are now so many technical issues to be addressed in the development, that design and implementation are not sufficient to face those problems. Since the OMG standardization of the Model Driven Architecture [4], it has been largely accepted that software development is about producing models. In fact, modeling is not new in the engineering field, and more globally in science: systemic analysis or simulation have introduced this notion a long time ago, as a way to reduce complexity [5][6][7].

Defining software engineering as producing models is one thing, identifying the relevant models (i.e. the relevant aspects that need to be considered) is another. This identification is also complex, and at the moment, there is no consensus for a unified approach within the software engineering community with the following exception. Understanding the business (i.e. the so-called “domain”) is recognized as a key

ingredient for which one or more business models should be produced. But regarding the technical software aspects, such agreement does not exist and several viewpoints are available. **The MDA view** [4] [8] for instance, is based on the Platform Independent Model (PIM) and the Platform Specific Models (PSM). In this view, the "platform" is the relevant aspect. There is no common agreement on the definition of this notion, but as MDA was build in the context of the Middleware war, we can assume that a platform corresponds more or less to a type of middleware (like an Object Request Broker (ORB), a Hub, or an Application Server). Another approach we can refer to is **the component-based view** [9], which considers the architecture as the central aspect.

Many other points of view are available, and an ontological unification is probably needed to face the multiple challenges of the software complexity.

On our side, we think that identifying the core technical aspects is particularly relevant in software modeling. These aspects are part of the problem domain, and most of them are shared among different software development. Examples of those recurrent technical aspects are: the persistence, the communication between resources, the security, the mobility or the interoperability. Having a formal description of these aspects would help to define their scope, and to understand and organize their possible associated solutions.

2.2 Software development is based on meta-models

Even if central aspects of the software have been identified, the next issue is about how to represent the knowledge associated with them. As the production of models is a key concept in the SE approach, the knowledge besides this production of models is at the best captured in terms of meta-models. As an example, we can refer to UML where models need to be elaborated in terms of concepts like classes, objects, relationships, etc. Knowledge about these concepts is captured in terms of the UML meta-model. A meta-model is thus a model of the model. Hence the main relation between a model and its meta-models is the instantiation relation. Theoretically, there is no limitation in this approach: meta-meta-models are themselves describing meta-models (the UML meta-model is itself expressed in the Meta Object Facilities (MOF)), and so on. It is up to each discipline to define their appropriate meta-models, and where it is necessary to stop the description. It is a trade-off between genericity and usability. All these meta-layers of models form the body of knowledge for the domain, including what should be known from a more abstract point of view to understand the less abstract concepts.

The body of knowledge corresponds to an ontology as it results in a shared conceptualization of things that exist and make up the world or a subset of it, i.e. the domain of interest [10], [11]. An inherent characteristic of ontologies is that they bear intrinsically the semantics of the concepts they describe [12]. This constitutes the necessary condition for an ontology to provide the semantics of something. In the IT field, an ontology is explicitly capturing the structural part of a knowledge domain.

All the different approaches discussed in 2.1 (MDA, component-based) are therefore based on meta-models. Some high-level meta-models can be considered common to these approaches as: the object model, the relational model, and some

other formalism, which are used for knowledge description. In a similar fashion, the process model deals with management and project issues. There are obviously many others.

On top of these high-level models, one can find specific meta-models, depending on the modeling view. For instance in the MDA view, a major concept is the Model Transformation: models are transformed all along the software development. One of the most representative transformations is the PIM-to-PSM one, which produced a PSM (e.g. an Enterprise Java Bean-based accounting model) from a PIM (an accounting specification model). To describe this transformation, a meta-model called the Platform Description Model (PDM) is introduced. In the component-based approach, a specific meta-model is needed to describe the relevant architectural concepts, like “component”, “connector”, “architectural styles”, “pattern” or “framework”.

We would like to stress the attention of the reader that the terms “model” and “meta-model” are indifferently used in this paper. Indeed, a meta-model is intrinsically a model, and it’s only by considering a given level of abstraction (e.g. the software), that the term “meta-model” should be used. As such, the Object model acts as a meta-model for the software.

We have shown in the previous section the importance of the recurrent technical aspects. Those aspects need to be understood, but also solved. Therefore, as introduced in Figure 1, we present in the next chapter two meta-models to take this last point into account.

3 The decisional and technical aspects meta-model

Regarding the syntax, our models are written in the Unified Modeling Language (UML). Class diagrams are used for the structural part, and collaboration diagrams for the behavioral part. Only the structural view is proposed in this paper.

The decisional model

Decision-making is recurrent in software engineering, and more globally in engineering. We therefore propose at the meta-level a decisional model. As shown in Figure 2, a particular focus is put on:

- Separating the problem from the solution, and as a consequence analyzing the problem before any suggestion of solutions [13],
- Modeling problems as independently as possible, to reduce the decision complexity,
- Acknowledging that a solution of any kind will be the source of new problems, which recursively have to be solved.

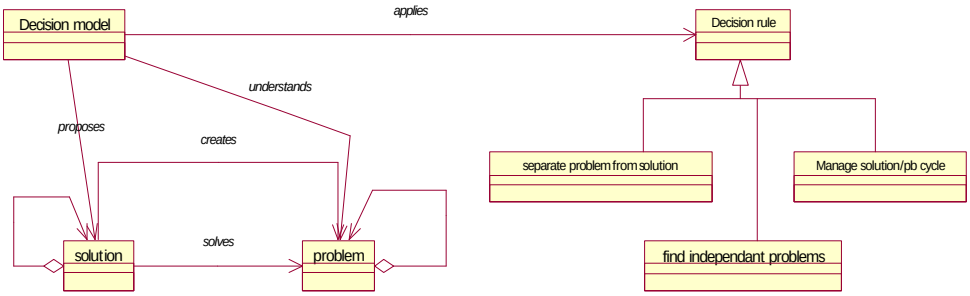

Fig. 2. The decisional model

The technical aspects model

The technical aspects model aims at referencing the recurrent technical problems of software like persistence, communication, interoperability, mobility, security, performance and, in general all the technical aspects that are recurrent and relevant for a given class of software.

On top of this taxonomy, a model that describes each of the identified problems is needed. Several benefits result from those models. First of all, they help to define clearly the technical problem, i.e. what it is and what is it not. Secondly they propose different solutions to address these technical issues. In that respect, they are instances of the decisional model.

For instance, a security meta-model would hold important entities like: software resource, access integrity, and modification integrity. It will also list possible solutions: access control for application integrity, encryption to ensure document confidentiality, or digital signature to check the possible modification of documents.

In a similar way, Section 4 is developing and proposing an interoperability meta-model based on the decisional meta-model.

4 The interoperability model

The notion of interoperability is not restricted to computer science. As a consequence, plethora of definitions have been introduced by different players. The study of some representative definitions is thus interesting to start our model-based ontology.

The IEEE [14] defines the interoperability as “the ability of two or more systems or components to exchange information and to use the information that has been exchanged”[2]. The hyperdictionary is maybe more business oriented: “The ability of software and hardware on multiple machines from multiple vendors to

[2] http://www.sei.cmu.edu/str/indexes/glossary/interoperability.html

communicate."[3]. The US Department of Defense proposes itself multiple definitions, mainly focused on weapons systems, while including some generalization effort[4]. The full definitions are available on [15].

From the IDEAS [2] viewpoint, interoperability is "the ability of a system (as a weapons system) to use the parts or equipment of another system" [5]. INTEROP [1] extends this definition in a more general fashion as "the ability of a system or a product to work with other systems or products without special effort of the part of the customer".

Finally, the Object Management Group (OMG) identifies interoperability as one of the major objectives within its Model Driven Architecture (MDA): "the tree primary goals of MDA are portability, interoperability and reusability through architectural separation of concerns" [4].

There are some relevant commonalities between all those definitions: the concepts of **system**, **communication**, **legacy reuse**, and **heterogeneity**. Those concepts are the basis of our model. In order to consider the different solutions that can be applied to those problems, through the instantiation of the decisional model, we produce an interoperability model, which on one hand introduces the definition of interoperability as problems, and on the other hand proposes a taxonomy of solutions to those problems.

It is of primary importance to note that heterogeneity is not only considered as a runtime issue, but that it has been introduced as a high-level concept (as it appears in the different definitions). In that respect, heterogeneity exists for resources such as artifacts or models.

Typically, high-level heterogeneity (at the level of the business model) can lead to low-level heterogeneity (at runtime at the data level). Therefore it is important to identify and solve the interoperability as soon as possible (i.e. as soon as the problem is introduced).

The simplified model

We propose a model for defining interoperability as a heterogeneity problem induced by a communication problem (Fig. 3). On the solution side, two major alternatives are introduced: the homogenization and the bridging solution. The notion of software resource is generic: it could be an object, a procedure, a table, a global model, or any kind of entities that constitute the software.

[3] http://www.hyperdictionary.com/computing/interoperability
[4] http://www.atis.org/tg2k/_interoperability.html
[5] http://www.ideas-roadmap.net

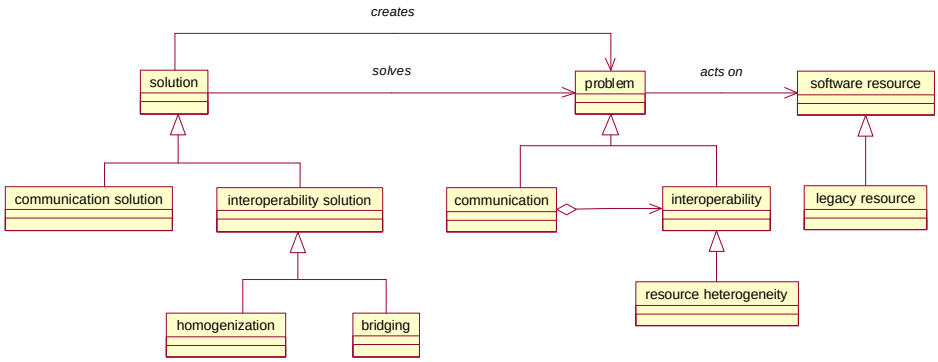

Fig. 3. The simplified interoperability model

The next section will detail the solution part: the communication solution, and obviously the homogenization and the bridging.

The detailed model

The communication meta-model

As stated above, the interoperability problem is induced from a communication problem. As a consequence, the heterogeneity problem has also to be solved only if the resources need to communicate.

It is therefore important to first understand what communication means, and separate it from interoperability. Communication and interoperability are part of the technical aspects model introduced in the previous section. In Figure 4, a communication meta-model is proposed, which introduces the general solution concepts to the communication problem. We can see here that most of the inducted problems (concurrence, performance, integrity) are directly dependant on communication problems. This means that even without any heterogeneity issue, we shall have to tackle theses problems.

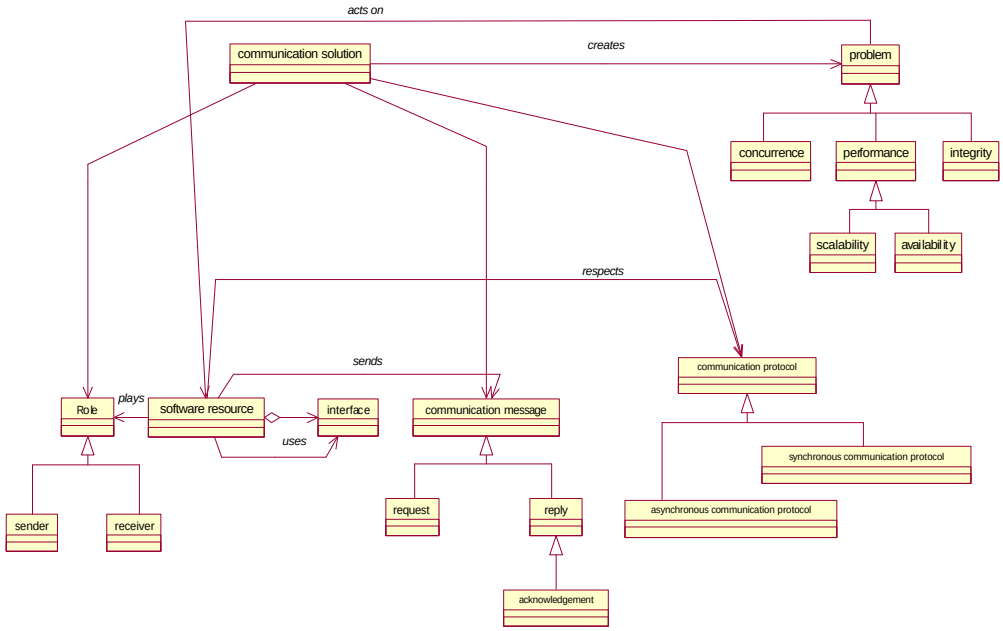

Fig. 4. The communication meta-model

The homogenization model

Homogenization is one possible way of solving interoperability. Figure 5 presents the detailed view of it. The unification of language, or of concepts may significantly reduce the heterogeneity problem. UML, or UEML [3] (dealing with enterprise modeling) are examples of unified languages.

Early use of unified models avoids heterogeneity. Unfortunately, we are facing integration issues, i.e. that legacy software is involved. Thus, besides unified models, the concept of transformation becomes again relevant: homogenization means that a given software resource, available in an initial format, should be translated in another one. A difference is made between heterogeneous syntax, and heterogeneous semantic. Indeed, the translation from a UML class diagram to an OMT static diagram is a simple syntaxic transformation, when transforming a UML class diagram in a relational diagram is more difficult due to semantics aspects. An important future research activity will be to define the most relevant translations.

As part of the decisional model, the homogenization solution creates new problems. The major issue is the resource modification. Indeed, having the opportunity to modify existing resources (models, artifacts, code) is not always something possible. On the other hand, the homogenization solution will guarantee the general validity and correctness of the software. The other problems are related to the validation of the unified models, which are introducing new complexity.

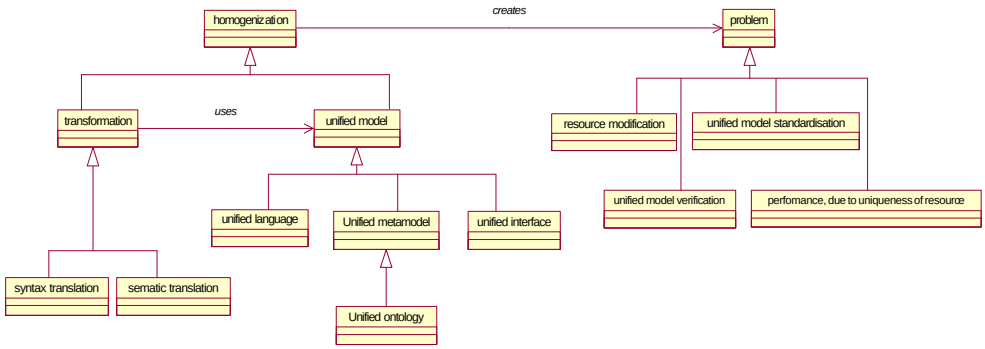

Fig. 5. The homogenization solution

The bridging model

Bridging is the last solution. When no information about the software system is available, when no resource modification is possible, then the only way to solve interoperability issues is to bridge and to adapt at runtime (Fig. 6). This is typically what message oriented middleware (MOM), Hub's, and more globally Enterprise Application Integration (EAI) are doing. On top of the communication solution, they define adapters between the heterogeneous resources, which are using a translation mechanism.

The inducted problems are mostly related to performance and integrity. Performance problem arise because most of the time it is necessary to translate messages from a given format to another one. This is resource consuming and all performance and availability requirements (like the response time) should be verified again. Adapters and translation methods may also introduce new vulnerabilities, i.e. possible ways to access or modify the data. In a general way, as something "external" to the software is plugged in, the global correctness of the system may be affected.

General heterogeneity solution, across hardware architecture like the XDR protocol is also typically a bridging solution.

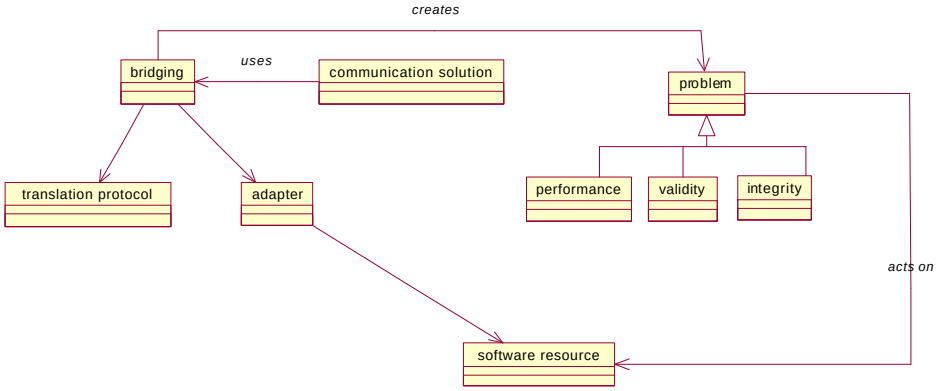

Fig. 6. The bridging solution

5 Conclusion and perspectives

In this paper, we have presented our preliminary results regarding a model-based ontology of the interoperability problem in software development. The first advantage of the model is that interoperability is considered as a global problem along the software development: business models, technical models, documents, code and more globally software resources can be integrated. As such, several SE actors (developer, architect, manager) can potentially take benefit from the use of the model.

Secondly, our model is based on a decisional model, which is probably one of the fundamental meta-models for software engineering (as shown for example in [16]). Effectively, we know that software development is a trade-off between multiple factors: business requirements, technical requirements, quality criteria, project/management issues, and many others. Consequently, interoperability should not be considered as an isolated problem within the development. Each interoperability solution may possibly have a huge impact on other issues. The added value of the decisional meta-model is to centralize the decision, and to enforce a global view of the problems. As a consequence, the model could be used for decision–aid, as for instance in COTS selection. Indeed, as stated before, there are many available integrated solutions, and justifying one of them is not obvious. Practical questions are typically: “Do I choose a Hub or a MOM to integrate my business?”, “Am I sure that this solution will really allow me to integrate my future systems ?”. Moreover, all those solutions are not only supposed to solve interoperability problems. Some of them are also able to guarantee a distributed transaction; most of them propose core communication protocols. As we already claimed, software engineering is a trade-off between multiple and contradictory aspects, and any kind of solution will introduce new problems. In this sense, being able to know as precisely as possible all purposes and impacts of integrated solutions is therefore a key issue in making the appropriate decision.

The model we present can be a starting point to a decision-aid method. Nevertheless, the real benefit will only be available when the models will be fully validated and enriched, in particular with integrated solutions (COTS, architecture). Moreover, as the model is part of a global decisional model, its applicability is directly dependant on the modeling of all other important aspects. We are currently working on meta-models for persistence and security in line with research projects on these topics.

Finally, we would like to stress the attention of the reader on the fundamental hypothesis we adopted concerning the driving force of the integration problem this work is based on. Indeed, we assumed in this paper that business integration is a pre-requisite for technical integration. This means that the fundamental heterogeneities that trigger the need for integration always occur at the business level. Technical integration is thus a consequence of a business integration need. In other words, when willing to solve a business heterogeneity problem, one may observe subsequent technical heterogeneities that need to be solved as well. Therefore, we assume that solving a technical heterogeneity problem without considering business integration issues probably pertains more to a migration problem than to an integration problem.

From the perspectives point of view, the actual model is not finalized: an important missing part lies in defining homogenization transformations (Fig. 5).

Another issue concerns the formalism: as UML is semi-formal, switching to a more appropriate language would possibly be beneficial to ensure rigor and future automation.

Acknowledgement

The authors would like to thanks Eric Grandry for his reviewing efforts and helpful comments.

References

1. INTEROP, European network of excellence (2004)
2. Interoperability Development for Enterprise Application and Software (IDEAS, European project) (2002). http://www.ideas-roadmap.net/
3. Unified Enterprise Modeling Language. http://www.ueml.org
4. Object Management Group: The Model Driven Architecture. http://www.omg.org/mda
5. Le Moigne, J.L. : La théorie du système général : théorie de la modélisation . Presses universitaires de France (1977)
6. Von Bertalanffy, L. : General system theory. Foundations, Development, Applications. New York (1968)
7. Dahl, O.J., Nygaard, K.: Simula – an Algol based simulation language, Vol 9 N°9. Communication of the ACM (1966) 671-678
8. Bezivin, J., Blanc, X. : Promesses et interrogations de l'approche MDA . In Developpeur Reference, (2002). http://www.devreference.net.
9. Garlan, D., Shaw, M.: An Introduction to Software Architecture. Advances in Software Engineering and Knowledge Engineering Volume 2. New York, NY. World Scientific Press (1993) 1-39
10. Sowa, J.F.: Knowledge Representation: Logical, philosophical, and Computational Foundations. Brooks-Cole, Pacific Grove, CA, USA (2000)
11. Grubber, T.R.: A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, 5 (1993) 199-220
12. Grubber, T.R.: The Role of Common Ontology in Achieving Sharable, Reusable Knowledge Bases. In Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning, Cambridge, Massachusetts, USA. Morgan Kaufmann Publishers (1991) 601-602
13. Jackson, M.: Problem Frames: analysing and structuring software development problems. Addison-Wesley (2001)
14. Institute of Electrical and Electronics Engineers: IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York (1990)
15. Joint Chiefs of Staff Publication: Department of Defense Dictionary of Military and Associated Terms, No. 1-02 (1999). http://huachuca-dcd.army.mil/nsto/lexicon/lexicon.htm
16. Subramanian, N, Chung, L.: Software Architecture Adaptability: An NFR Approach. Proc., Int. Workshop on Principles of Software Evolution (IWPSE`01), Vienna, Austria. ACM Press, (September 2001) 52-61.