Mapping Processes Between Parallel, Hierarchical and Orthogonal System Representations

Authors

Francis Dion
Epsilon Technologies inc.
1200, Boul. Chomedey
Laval (QC) Canada H7V 3Z3
fdion@xpertdoc.com

Thanh Khiet Tran
tkhiet@yahoo.com

Alain Abran
Professor and director of the
Software Engineering Management Research Lab.
Université du Québec à Montréal
Département d'informatique
C.P. 8888, Succ. Centre-ville
Montréal (Québec), Canada H3C 3P8
abran.alain@uqam.ca

Abstract

The importance of software system representation through models and visual diagrams is increasing with the steady growth of systems complexity and criticality. Since no single representation is best suited to address all the documentation, communication and expression needs of a typical software development project, the issues related to conversion and coherence between different representations are having a significant impact on team productivity and product as well as process quality.

This paper explores the types of relationships that exist between representations and the impact they have on mapping, generation and synchronization processes. We propose a characterization of those relationships as being parallel, hierarchical or orthogonal. Examples and comments on mapping or transformation processes and automation prospects in the context of software size measurement are also provided.

Keywords
REPRESENTATION, UML, MAPPING, MEASUREMENT

Introduction

In the field of software development and maintenance, we need models whenever the systems we are working on become too complex to be instantly and completely grasped by all involved individuals.

Models and diagrams are used for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling[1]. They are tools for stakeholders to communicate their understanding about the system. They can be used to broadly or precisely specify the work to be done. They can serve as blueprints for construction or as a basis for cost and schedule estimation. They can also be used as input for automated size measurement, property validation, code generation and a host of other purposes.

Local Rules:
Differences Between Organizations Usage of Model Elements

The choice of what models and diagrams one creates has a profound influence upon how a problem is attacked and how a corresponding solution is shaped. Furthermore, the same model elements can be used differently to express different concepts or different levels of abstraction from one organization to another.

Those variations are imputable to many factors, including:

- Organizational culture.
- Available experience and expertise.
- Tools and tools usage.
- Corporate or personal goals and objectives.

Every complex system is best approached through a small set of nearly independent views of a model[2]. No single representation of a system can efficiently express or communicate all needed perspectives on that system. A diagram that is well suited for a specific need or task could be too low level, too high level or conceptually too distant to be used in another context.

Since models and diagrams are capable of expressing the details of a system from a number of perspectives, one of the recurring issues faced when applying modeling surrounds the management of "Enterprise-wide" modeling efforts and models. Specifically,

[1] Adapted from [1], page xi.
[2] [1], page 1-3.

there are concerns about models that exist at different level of abstractions, and how to manage these different models[3].

The cost associated with the independent production of many representations for the same system can be quite substantial. The evolution of the system itself is also likely to be an issue in terms of maintaining the coherence of loosely related representations. It thus seems reasonable to look for ways to somehow relate those representations together. By examining the nature of their relationships, we should be able to assess the potential to produce one representation as a function of another one. In some instances, the mapping would be straightforward and the process would be easy to automate. Other mappings, though, would be much less obvious and require human intervention.

This paper presents a categorization scheme for the relationships between representations in order to assess their potential for systematic mapping. This classification is a fuzzy set of three categories representing decreasing correspondence between elements of the representations. The categories are: parallel, hierarchical and orthogonal.

Please note that, although the examples in this paper are mostly UML diagrams, we believe the concepts and ideas presented here to be as applicable with any other modeling technique and even across notations.

Definitions

A **model** is a simplified representation of a system or a process. A system can be represented by any number of models. A model could also represent a class of systems or processes.

A **view** is a specific expression of a model. It can be seen as a window on the model, exposing part or all of the information it contains in a format that is suitable for some specific use or user. A single model can be expressed through many distinct views.

A **diagram** is a graphical view, by opposition to a textual or tabular view.

Despite their definite differences, these concepts are more or less interchangeable in the context of this paper. To avoid unnecessary confusion, the term “representation” is use here to mean either one of them.

Relationships between representations

When considering the ease and usefulness of establishing a mapping between two distinct representations, one must be able to understand and characterize the relationships that exist between them. To that effect, we suggest the use of the following categories:

- The representations are **parallel** if they express roughly the same concepts at the *same level of abstractions*.
- The representations are **hierarchical** if they express roughly the same concepts but at *different levels of abstraction*.

[3] Adaped from [2]

- The representations are **orthogonal** if they express *unrelated (or at least not directly related) concepts* of the system/unit.

Those categories are not crisp classifiers. They are fuzzy symbols expressing a relative positioning across a continuum from perfect isomorphism to complete independence. As we intend to demonstrate in the following sections, they are useful conceptual tools to assess the potential for systematic mapping between two specific representations.

Parallel Representations

Two distinct representations of a system or unit are said to be parallel if they express roughly the same concepts at roughly the same level of abstraction.

The mapping or conversion process between parallel representations should be straightforward and easily automated since their information content is identical and their differences lie mainly in the way this information is organized and presented.

Another implication of this definition is that either one could be obtained as a function of the other and that they could both be expressions of the same internal representation.

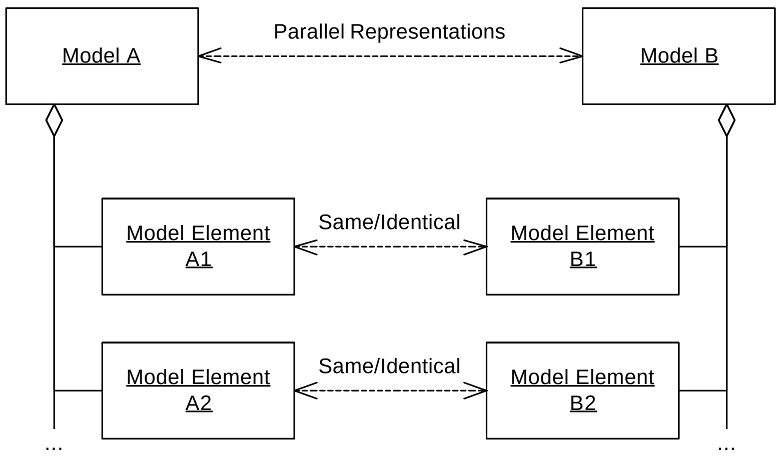

Figure 1: Illustration of a parallel relationship between to representations

One example of parallel representations is illustrated by the well-known isomorphism that exists between *collaboration* and *sequence* diagrams in the UML notation. Both are instances of the abstract *interaction* type of diagrams and one form can readily be converted to the other without any loss of information. Many modeling tools directly support this transformation.

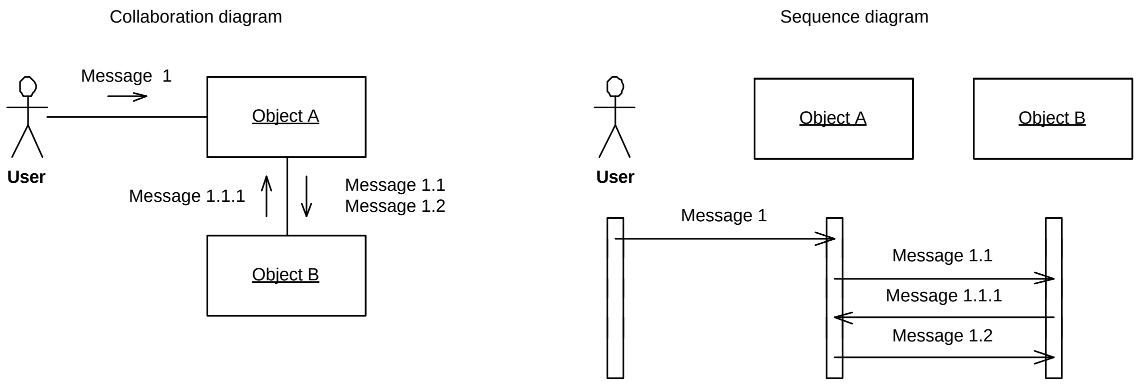

Figure 2: Example of isomorphic collaboration and sequence diagrams

Another less obvious parallel relationship exists between the model of a class and its representation as a use case diagram. The use cases represented here are not those of the overall system but rather of the users of the class, usually other classes or components.

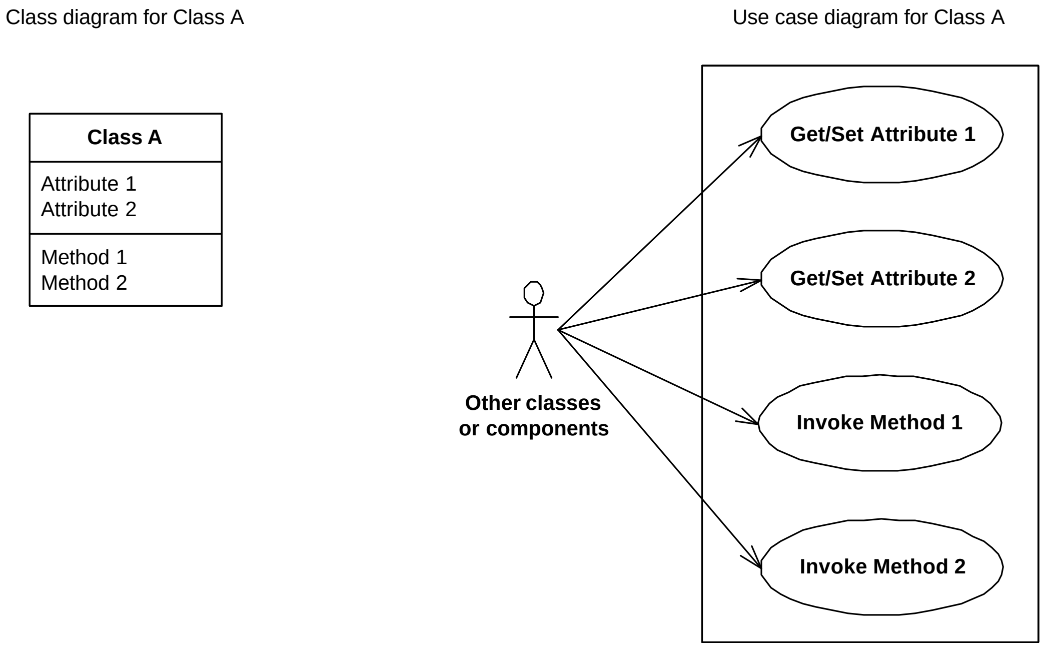

Figure 3: Example of a parallel relationship between a class an a use case diagram

Hierarchical Representations

Two distinct representations of a system or unit are said to be hierarchical if they express roughly the same concepts but at different levels of abstraction. In other words, one of the models or diagrams presents a detailed view of the system while the other is a more synthesized, bird's eye view of that same system. This type of relationship is especially emphasized by top down methodologies where one goes from a high level specification to detailed specification to high and low level design and so on all the way down to implementation. Each level needs to be traceable to its predecessor while adding new, more detailed information. Such models or views could share the same internal representation, although at least one of them would use only part of the available information.

In contrast with parallel relationships, hierarchical transformations require human intervention or comprehensive heuristic rules to either “fill-in-the-blanks” (when moving top-down) or select the significant elements (when moving “bottom-up”).

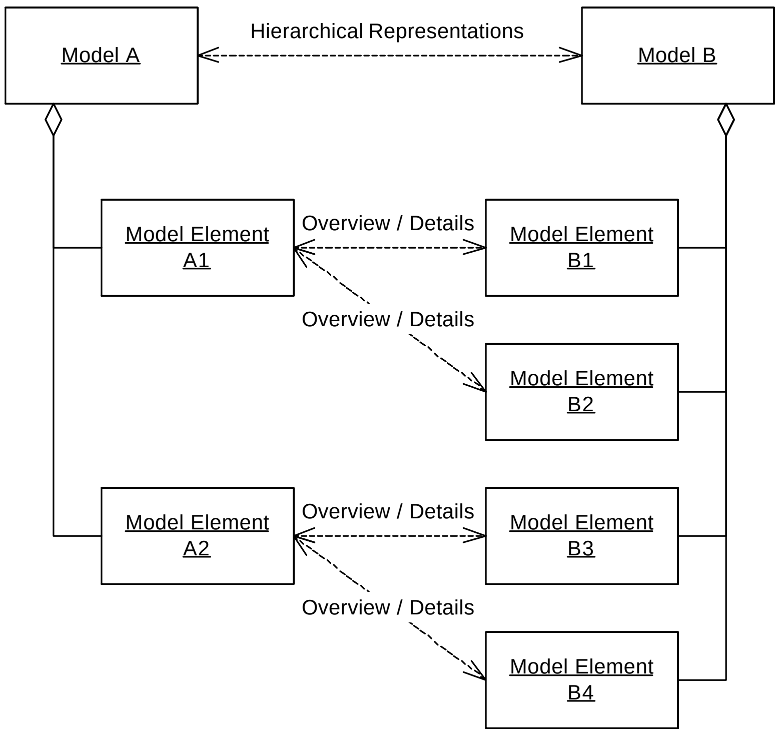

Figure 4: Hierarchical representations

A use case diagram and the various collaborations which represent the detailed specifications of the use cases are a good example of a hierarchical relationship.

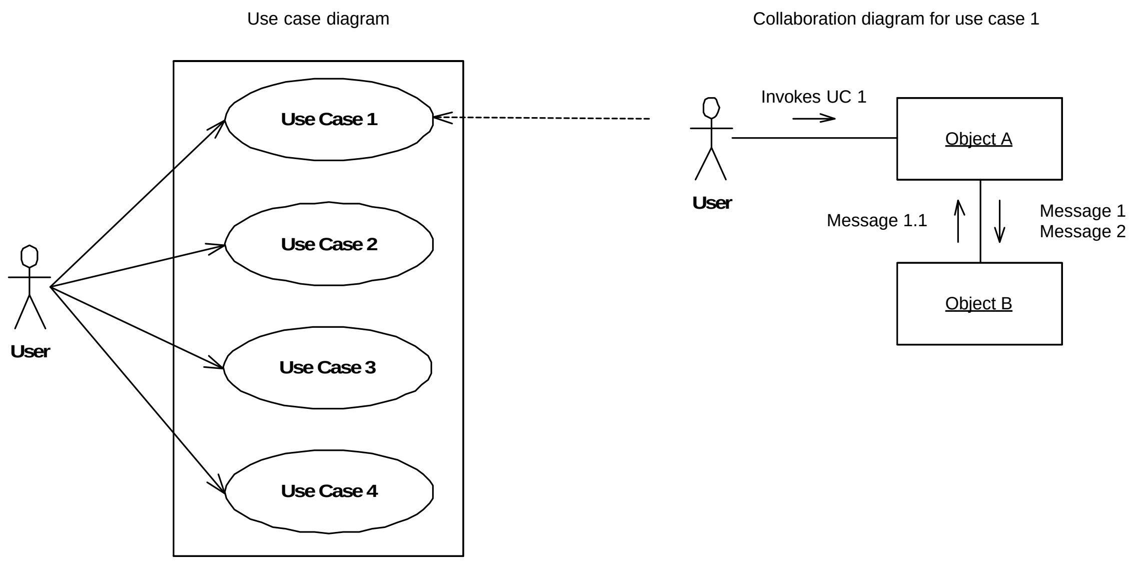

Figure 5: Hierarchical relationship between a use case diagram and a collaboration diagram

Another example of hierarchical relationship can be found when one diagram represents a summarized view where much of the details of the other one have been “folded”, either by way of generalization or by packaging together similar features.

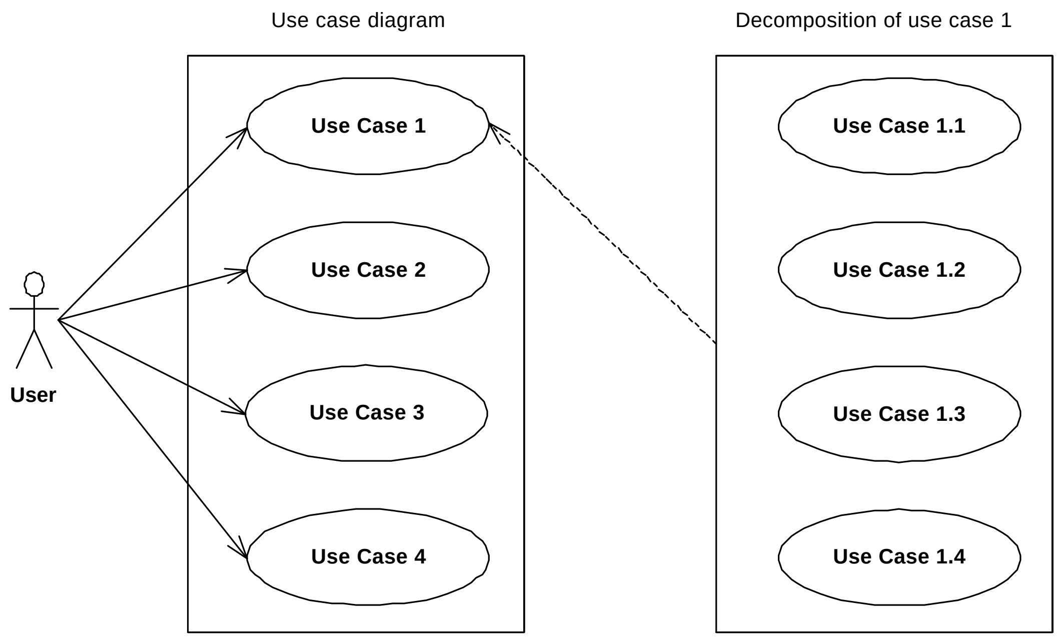

Figure 6: Hierarchical relationship expressing a generalization between use cases

Orthogonal Representations

Two distinct representations of a system or unit are said to be orthogonal if they express unrelated or at least not directly related concepts of the system/unit. This means that their respective items diverge not (or not only) by their form or level of details but by the nature of the objects they represent.

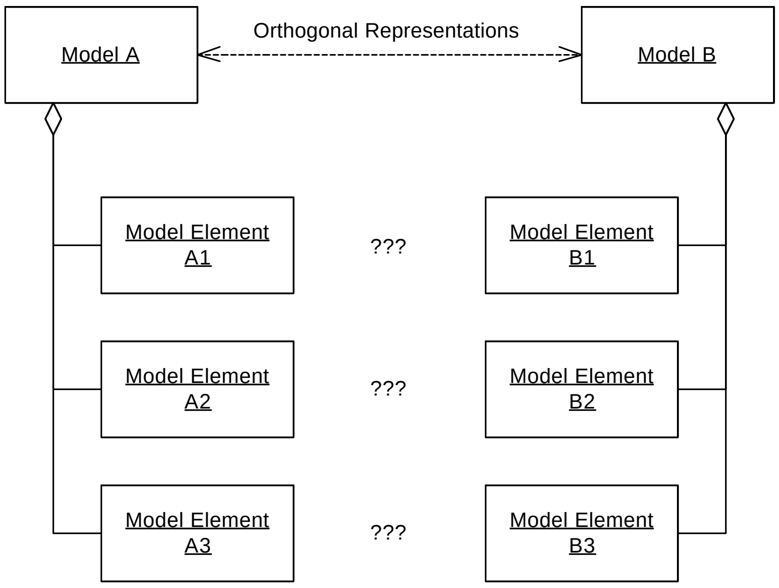

Figure 7: Orthogonal representations

When a purely orthogonal relationship exists between two representations, it can be assumed that there is no systematic mapping by which one could be derived from the other. More common would be situations where only a partial mapping is possible, requiring human intervention or very sophisticated heuristics for such a transformation to be performed. In some situations, a hierarchical or parallel relationship exists but cannot be fully determined because intermediary “levels” are missing, because extensive optimization and reuse have blurred the initial structure or because the representations have not been kept in synch and therefore are linked to visions that have diverged over time.

Application to the measurement process

We would now like to put this discussion in the practical context of the C-FFP software functional size measurement process.

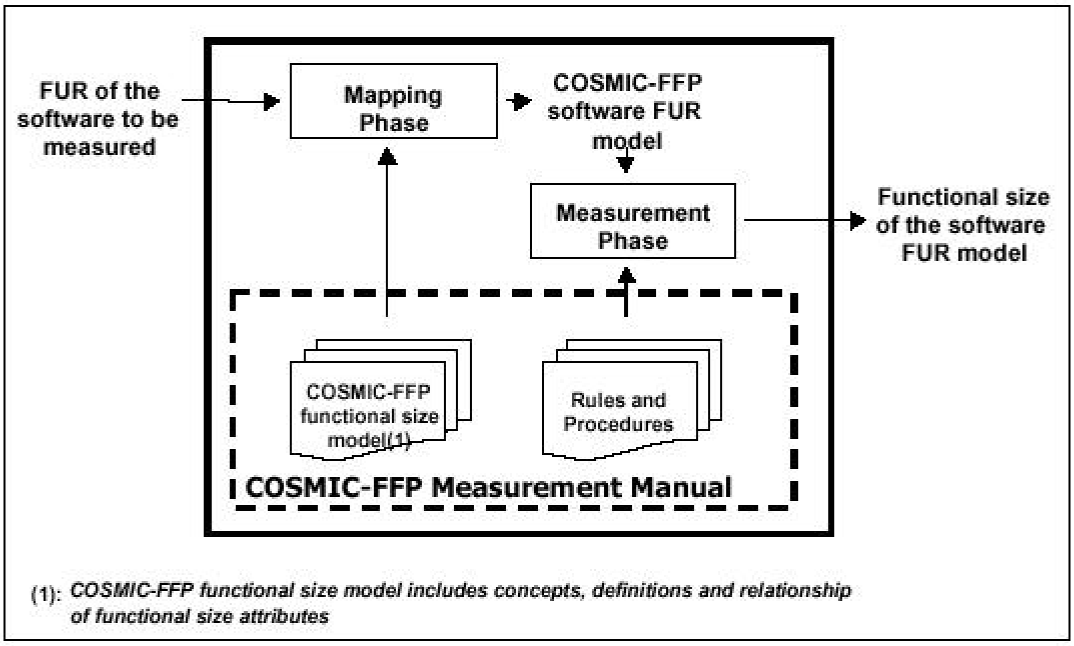
FUR of the software to be measured
Mapping Phase
COSMIC-FFP software FUR model
Measurement Phase
Functional size of the software FUR model
COSMIC-FFP functional size model(1)
Rules and Procedures
COSMIC-FFP Measurement Manual
(1): COSMIC-FFP functional size model includes concepts, definitions and relationship of functional size attributes

Figure 8 COSMIC-FFP Measurement Process Model[4]

The COSMIC-FFP method applies measurement to a generic model of the software functional user requirements onto which actual artifacts of the software to be measured are mapped[5]. The COSMIC-FFP mapping phase takes as input the functional user requirements of a piece of software and produces a specific software model suitable for measuring functional size. In many situations, those functional user requirements have to be obtained from alternate sources, like architecture and design models[6].

This measurement method essentially consists of making a model of the software in which the functionality has been breaking down into series of data movements between layers separated by boundaries. Those series are called “functional processes” and the data movements are classified in four categories: Entries, eXits, Reads and Writes (see [3] for details).

[4] Taken from [3], page 13.
[5] Idem.
[6] Ibidem, page 13 and 14.

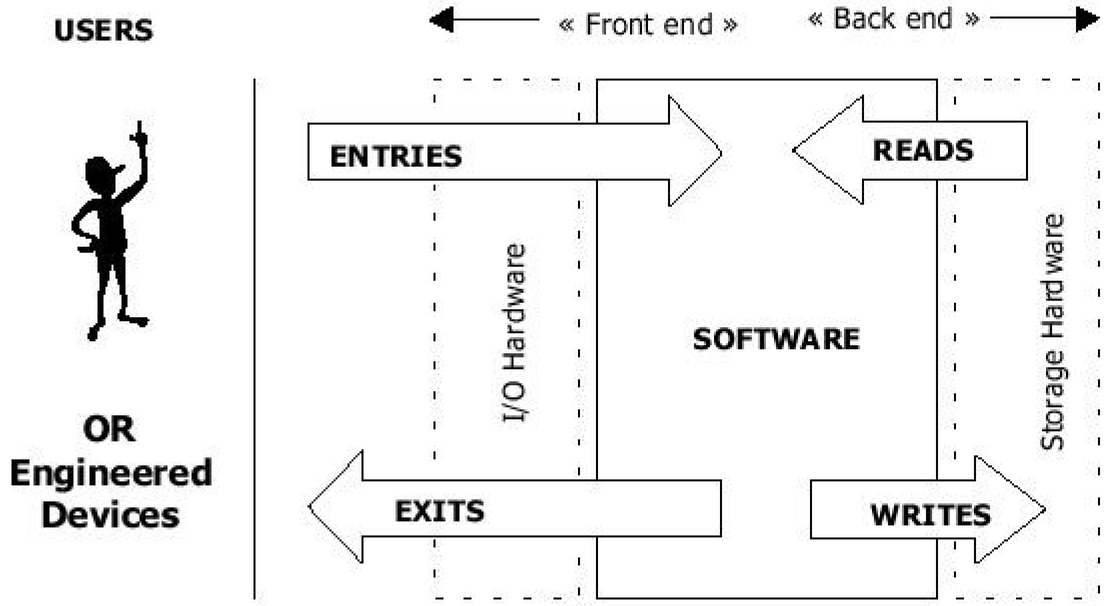

Figure 9: Generic flow of data through software from a functional perspective

One of the first mapping challenges that should be addressed in the context of this measurement method is: “How can the C-FFP software model elements be expressed in terms of the UML notation elements?”. As demonstrated by Bevo et Al. in [4], most C-FFP software model elements map directly to UML notation concepts as follows:

C-FFP software model elements		UML notation concepts
Functional Process	<=>	Use Case
User or Engineered Device	<=>	User
Data Group	<=>	Class
Data attribute	<=>	Attribute

The following elements, on the other hand, do not lend themselves to any obvious mapping:

C-FFP software model elements		UML notation concepts
Functional Sub-Process (Entry, eXit, Read and Write)	**???**	Scenario
Layer	**???**	Package

One way to view this mapping is to consider, for example, that a use case, in any model coming from any organization, should always be mapped to a functional process and that a data group can only be identified by a class in an UML diagram. This, we believe, would not be a very good view because it does not take into account the context and the purpose for which those specific models were created.

Suppose that an organization decided to use a local rule specifying that the verb “manage” should be used in a use case name as a shorthand for the typical “CRUD” database activities: Create, Read, Update and Delete. This organization would then produce models with use cases labeled as such:

- UC1 Manage Entity X
- UC2 Manage Entity Y
- Etc.

For C-FFP measurement purposes, these use cases should be expanded as four functional processes each, one for each activity. Thus a **hierarchical** relationship exists between the original use case model and the expanded one. Since the expanded model **parallels** the requirements of the C-FFP method, the measurement can then be readily performed.

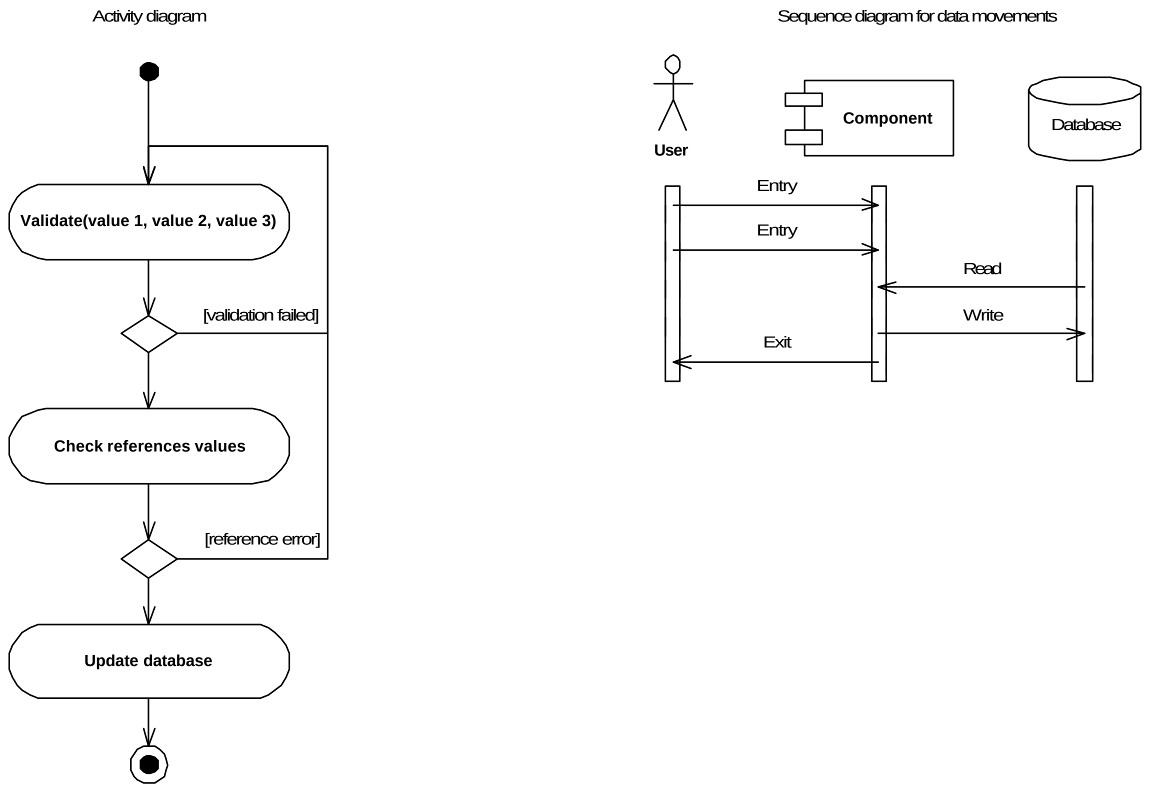

Figure 10: Orthogonal relationship between two representations of the same functional process

Here is a different example. These two diagrams are representations of the same use case / functional process. One is an activity diagram depicting the algorithmic features of the process while the other is a sequence diagram illustrating the data movements coming in and out of the component realizing the process (as for C-FFP measurement). Although they both represent the same function, they do so through very different, almost **orthogonal** sets of concepts. In this example, there is no indication that the “Check references values” activity really represents a read on the database. Only a deep understanding of the system documented, or the use of agreed upon conventions, would allow such a mapping to be performed.

Conclusion

The increasing role that models play in modern software development and maintenance activities makes it more and more important to understand the dynamics of their relationships.

Although it is possible to express almost any aspect of a system using the same model elements, this in itself is not enough to guarantee a simple, straightforward adaptation process that would allow any model to be used as input for every task. We believe that the characterization of representations' relationships presented here could serve as a conceptual framework to guide us on the assessment of mapping, transformation and generation potentials.

Future research directions on this topic include the exploration of the concept of "orthodoxy" versus local rules in modeling as well as more formal and complete specifications of the mapping processes between representations.

Acknowledgements

The authors wish to thank France Cano and Sylvain Hamel for their comments on the first drafts of this paper.

References

[1] OMG Unified Modeling Language Specification, Version 1.3, June 1999, Object Management Group

[2] Multiple Models, Tom Schulz, Rational Software, 1999, http://www.rosearchitect.com/mag/archives/9901/extreme.shtml

[3] COSMIC-FFP Measurement Manual, Field Trials 2.0 Version, October 1999

[4] Application de la méthode FFP à partir d'une spécification selon la notation UML, Valéry Bévo, Ghislain Lévesque et Alain Abran, IWSM 1999.