Intentional Modeling with Institution Theory

Mohamed Bettaz, Mourad Maouche

Computer Science Department, Philadelphia University
Amman, Jordan
mbettaz@philadelphia.edu.jo
mmaouch@philadelphia.edu.jo

Abstract — Institution theory is a categorical approach to abstract model theory free of commitment to any particular logical system. As regards to modeling, the concept of institution contributed to a better understanding of various modeling concepts such as heterogeneous specification and multi-modeling, and permitted the elaboration of satisfactory answers to questions in relation with the semantic correctness of model transformation. Intentional modeling is an approach to software system’s specification that puts focus on intentions and motivations of software systems, rather than on their essence or behaviors. One might here rightly think that requirements engineering is there to take care of this type of concern. However, to quote the authors of the institution theory, “experience in software engineering shows that there are major difficulties in producing consistent, rigorous specifications that adequately reflect users’ requirements for complex systems”. In this contribution we project to use the concept of institution to represent models together with their intentions, and to relate models having different intentions in a semantically consistent way. The underlying motivation is to contribute to a better understanding of the concepts of modeling and model transformations.

Keywords — model driven engineering; intentional modeling; category theory; institution theory;

I. INTRODUCTION

Model-driven engineering came with two main concepts: modeling and model transformations. In spite of the plethora of definitions introduced by concerned communities [11, 12], no precise definition has so far brought a consensus among domain experts. This led the authors of [1] to adopt an approach, inspired by category theory [8], which abstracts from concrete representations of models, to focus on the potential relations between them. Models are then simply called things, while a relation between two things is designated by the expression RepresentationOf. Things, in fact, correspond to objects in category theory, and RepresentationOf relations to category morphisms. A RepresentationOf relation is graphically represented by arrows as in category theory (Fig.1). X and Y are names of things and μ is the name of a relation between them. This has to be read “X is a representation of Y”. Out of this, little from the stuff of category theory is used in the cited work.

The used approach allowed its authors to be in conformity with various points of views expressed by other authors in their attempt to give “useful” definitions to the way of modeling models, and to answer many other interesting questions related to the subject [11, 12].

However when coming to give a more precise meaning (a semantics) to their arrows, the authors of [1] faced the need to use the concept of intention, attached to things but “captured” by the relations between them. The notion of intentional modeling is not new in itself and has already been used in contributions related to software process modeling more than two decades ago [3]. This kind of modeling puts the focus on intentions and motivations of software systems, in order to understand the whys underlying the whats and the hows behind the existence of the systems under study. This gave rise to the so-called intentional modeling languages used mainly in requirement engineering [3, 4].

In order to represent their intention-based semantics captured in the arrows, the authors use a graphical notation based on the notation used in Fig.1. Five kinds of μ relations are identified that are represented by different kinds of arrows. The kind of arrow used in Fig.1 to represent relations between things is also used to represent relations between things sharing the same intention. That is X \(\rightarrow\) Y is also interpreted as: The intention of X is the same as the intention of Y. The intention of a thing T is denoted by I(T). The representations for various kinds of μ relations defined in [1] are summarized in Table I, where we just replaced Venn diagrams by a set-theoretic notation, and simplified some parts of the description.

The rest of the paper is organized as follows. Section II is devoted to related works. In section III, we present a running example. Section IV introduces institutions. In section V we show how to associate institutions to SE-models and to their intentions. Section VI emphasizes some of the results. Section VII draws conclusions and outlines future work.

II. RELATED WORKS: THINGS, INTERPRETATIONS, AND INTENTIONS

A. Things and Interpretations

The use of only one kind of arrow to express relations between things (Fig.1), and more than one kind of arrows to express
relations between intentions (Table I) might probably be explained as follows.

It is obvious that, the definition of “meaningful” relations between things might be satisfied just by providing names for them. This is because names are - by their nature - abstractions at the highest possible level. In category theory names of objects might be interpreted as sets, algebras or whatever else, including degenerated objects (for instance sets with no elements). In this sense, we talk about the class of all interpretations (category of all models). This explains probably why the authors in [1] are using only one kind of arrow when talking about relations between things. Fig.2 illustrates the correspondence between names (of things X and Y), names and their intended categories of models (X, M_X and Y, M_Y), and between categories of models corresponding to different names (M_X and M_Y).

Table I: Relations between Things and Things Intentions

<table>
<thead>
<tr>
<th>Intention</th>
<th>Description</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>I(X) = I(Y)</td>
<td>X and Y have the same intention. They can represent each other.</td>
<td>X [ \mu ] Y</td>
</tr>
<tr>
<td>I(X) \cap I(Y) = \emptyset</td>
<td>X and Y have totally different intentions.</td>
<td>X [ \mu ] Y</td>
</tr>
<tr>
<td>I(X) \cap I(Y) \neq \emptyset</td>
<td>X and Y share some intention. X and Y can be partially represented by each other.</td>
<td>X [ \mu ] Y</td>
</tr>
<tr>
<td>I(X) \subseteq I(Y)</td>
<td>The intention of X is part of the intention of Y. Y can be partially represented by X.</td>
<td>I</td>
</tr>
<tr>
<td>I(X) \supseteq I(Y)</td>
<td>X covers the intention of Y. X can represent Y.</td>
<td>X [ \mu ] Y</td>
</tr>
</tbody>
</table>

Names and potential interpretations

However, our experience with software engineering modeling (SE-models [2]) shows that it is not always “useful” to consider intentions of “dummy” models; this is because intention definition requires knowing more than just the name of the model. This means that it is necessary to use a more elaborated syntax than just names to designate things and to get a more precise interpretation for them, as well as for their intentions.

In this work we show that if such an elaborated syntax (for designating things) consists of signatures (informing a little bit more on the essence of things than just a name is supposed to do), then intentions will be expressed by sentences formed by the vocabulary provided by the signatures; models will then be interpretations of signatures (not just names) satisfying sentences (representing the intentions). This is behind our idea of suggesting the use of institution theory [5] for representing things (and their intentions) such as described in [1]. Compared to Fig.2, Fig.3 shows the difference between things abstracted to their names and things abstracted to a signature informing more on their nature. X and Y here are things signatures, and M_{I(X)} and M_{I(Y)} are interpretations of signatures satisfying their intentions (represented by sentences denoted by I(X) and I(Y)). Actually in Fig.2, if things have different names, it is not sure that they got necessarily different interpretations. This is no more the case when things are represented by more appropriate signatures.

A more elaborated explanation of our idea requires however a brief introduction to institution theory (Cf. Section IV). Signatures, sentences and institutional models (Ins-models) form in fact the basic ingredients of an institution in the sense of [5]. It is worth to mention here the crucial difference between SE-models and Ins-models; the former are partial specifications whereas the latter are closer to the implementation [2].

Names, intentions and their interpretations

B. More about Intentions

According to [14], the intentional modeling focuses on intentions and motivations of software systems rather than their behaviors. Intentions define the why for existing systems. KAOS [15] and i* [17] are two popular intentional modeling languages mainly used by the requirement engineering community. These languages, while different, support a set of common core intentional elements. Both identify various (shared but also different) kinds of intentional elements (goals, system constraints, and others). Goals, which are statements that describe what a system is designed to achieve [14], are supported by both languages. Such goals, seen as a kind of
constraints, are generally expressed using an appropriate logic-based formalism such as temporal logic, OCL (Object Constraints Language), and others. In this work OCL [16] is used to express intentions (restricted to goals seen as constraints). More concretely, we address only the goal intentional element, through the KAOS (goal) vision. The KAOS approach adopts a separation of modeling concern strategy by separating between SE-models and their intentions, whilst providing means to relate them; just like institution theory provides means (through morphisms and co-morphisms) to define various logical frameworks and to relate them. It is also worth to mention that such a separation of modeling concern strategy is also adopted by the authors of [1], differentiating between a thing T and its intention denoted I(T). As the reader should have noticed, this case is not taken into account in the illustration given in Fig.3. We will come back later to this issue, once institutions have been introduced (Cf. section IV).

III. A RUNNING EXAMPLE

As said earlier, things in which we are interested are SE-models. It is not uncommon to represent such models by various formalisms (semi-formalisms sometimes) depending on the views or system facets such (sub-) models are supposed to represent. In the context of the Model Driven Architecture (MDA), and the Model Driven Engineering (MDE) methodology, software engineers use CIM (Computation Independent Models) to address the domain or requirement level, PIM (Platform Independent Models) for the design level, and PSM (Platform Specific Models) for tackling the implementation level. In addition to that, the MDE methodology preconizes a set of useful model transformations such as CIM-PIM, PIM-PIM, PIM-PSM, and others. These transformations are mainly refinements operated on SE-models. To illustrate our idea, we opted for an example extracted from a case study related to train control systems and treated in [13]; the example will be used for modeling at two (MDA) viewpoints: the CIM and PIM ones. The CIM viewpoint will be represented by a UML class diagram while the PIM viewpoint by an Object-Z specification; the motivation for such a choice will be explained at due place in this paper. However for lack of space, we treat in detail only the CIM viewpoint in this paper. So we firstly give an intuitive description of the example, then a presentation of the CIM followed by a description of its intention.

A. Train Control System

In this context, the domain of interest deals with trains, transporting of passengers, and moving of trains from platforms to platforms. Trains are either moving or temporarily stopped. We assume that trains speed sensors work well, and that trains speed equals to zero means that trains are stopped; trains may be stopped either at platforms or outside the platforms (for some reasons); no more than one train might be stopped at a given platform at a given time, but different trains might be stopped at the same platform at different times.

B. The CIM Viewpoint

The UML class diagram in Fig.4 gives a concrete representation for our CIM.

The CIM consists of two classes (Train and Platform) and an association (At). Here a train is characterized by its identifier (T-id), its current speed (CurrentSpeed), and the state of its doors (DoorState). A stereotype-like notation is used to define the Status data type. A platform is characterized just by its identifier (P-id). Classes representing trains and platforms provide the usual Get and Set operations. The unique association (At) expresses the fact that a train may be either stopped at a given platform or located between two platforms that follow one another. The association At has two association roles (guest and hostedBy).

Fig. 4. CIM viewpoint representation by a UML class diagram

C. The CIM Intention

As regards to train transportation systems, safety might be considered as one of the rationale (why) formulated by concerned stakeholders. In our example (represented in Fig.4), we consider only one safety aspect that might be formulated as follows. Train doors are maintained closed while moving:

Train doors cannot be opened when trains are stopped outside platforms. Their OCL specifications are given by the following OCL invariants [16].

context: t : Train
inv: t.CurrentSpeed = 0 or t.DoorState = closed
context: t : Train
inv: t.hostedBy ≠ ∅ or t.DoorState = closed

IV. INSTITUTIONS

A. Background

Institutions [5] are an abstract formalization of the notion of a logical system.

Informally, an institution consists of

- a collection of signatures and signature morphisms, together with for each signature
• a collection of sentences,
• a collection of models, and
• a satisfaction relation of sentences by institutional models (Ins-models),

such that when signatures are changed by signature morphism, satisfaction of sentences by Ins-models changes consistently.

As stated in [9], the exact nature of signatures, sentences and models is left unspecified, which leads to a great flexibility. This allows providing various SE-models (which do not at first sight look like logics) with an institutional semantics. Examples are class diagram models [7, 10], database relational schema models [9], and Object-Z models [6].

Signature morphisms can be seen as mappings between signatures. When a signature is changed (by signature morphism), sentences (over such a signature) can be translated along the signature morphism and models (interpreting the signature) reduced against the signature morphism. By “satisfaction of sentences by models changes consistently” it is meant that satisfaction is invariant under change of notation and enlargement of context (along signature morphism). For more details, the reader is kindly referred to [5, 9].

Signatures are used to define the vocabularies used in constructing the sentences in the (formalized) logical system.

Using a collection of signatures allows the logical system to work with different vocabularies at once, instead of a fixed vocabulary as in traditional model theory.

Morphisms and co-morphisms between institutions form the foundation for relating logical systems. This can be used to catch model transformations in a semantically coherent way. Intuitively [2], co-morphisms map a “poorer” institution into a “richer” one, whereas institution morphisms forget logical structure by mapping a “richer” institution to a “poorer” one. However given two institutions, say $\mathcal{I}$ and $\mathcal{J}$ it is not always possible to relate $\mathcal{I}$ and $\mathcal{J}$ either by a co-morphism or by a morphism. In this case one has to find an intermediary institution $\mathcal{K}$ which is “poorer” than both concerned institutions. Such an intermediary institution plays the role of a “lowest common denominator”. The intermediate institution can then be used to relate both institutions $\mathcal{I}$ and $\mathcal{J}$ by what is called in [2] a semantic connection.

B. Institutions for Things Considered as SE-models

We now come back to the illustration in Fig.3, and explain in more detail the extracted part represented in Fig.5

Instantiation of thing name by SE-model / intention signature

The illustration in Fig.3 represents in fact the point of view expressed in [1] in the sense that intentions are expressed for things regardless of what things are supposed to represent. In our case however things are seen as SE-models having a concrete representation. Applying a separation of modeling concern strategy (in conformity with [1] and [15], and MDA in general) suggests to first associate an institution to the SE-model, then to associate an institution to the SE-model together with its intention. The intention of the SE-model is then expressed through an (embedding) morphism (in fact a co-morphism) between the first institution and the second one. Fig.6 and Fig.7 illustrate both institutions. The reader should have noticed that (for seek of simplicity) the term SE-model intention institution is used to mean associating an institution to an SE-model together with its intention.

Associating institutions to SE-models

![Fig. 7. Associating institutions to SE-models with intentions](image-url)
V. ASSOCIATING INSTITUTIONS TO SE-MODELS AND TO THEIR INTENTIONS

For lack of space we show here, through our Train Control System example, how to associate an institution to the corresponding CIM, an institution to the CIM with its intention, and how to define the co-morphism expressing the relation between both institutions. Associating an institution to the corresponding PIM, an institution to the PIM with its intention, as well as defining the co-morphism expressing the relation between both institutions might be (up to the formalism used to represent the PIM) obtained following a similar approach. It is worth to mention at this place that institution theory doesn’t give any suggestion on how to specify signatures, sentences and models, which leaves a great deal of liberty to the specifier. Nevertheless in this work we don’t intend to “reinvent the wheel” and use the nice results in [7] and [10] showing how to associate an institution to UML static structures and to OCL respectively.

A. The CIM Institution

1) The Signature

The signature for a UML class diagram is given by $\Sigma = C \cup T \cup O$ where $C$ is the set of class names, $T$ the set of data type names, $O$ the set of method names, $A_t$ the set of class attribute names, and $A_s$ the set of association names. So, for our CIM viewpoint (Fig.4), we do have:

$C = \{\text{Train, Platform}\}$

$T = \{\text{String, Real, Status, Void}\}$

$A_t = \{\text{t-id : Train} \rightarrow \text{String}, \text{CurrentSpeed : Train} \rightarrow \text{Real}, \text{DoorState : Train} \rightarrow \text{Status}, \text{P-id : Platform} \rightarrow \text{String}\}$

$A_s = \{A_t \subseteq \text{guest} : \text{Train} \times \text{hostedBy} : \text{Platform}\}$

$O = \{\text{Set-tid : Train} \times \text{String} \rightarrow \text{Void}, \text{Set-dos : Train} \times \text{Status} \rightarrow \text{Void}, \text{Set-csp : Train} \times \text{Real} \rightarrow \text{Void}, \text{Get-tid : Train} \rightarrow \text{String}, \text{Get-dos : Train} \rightarrow \text{Status}, \text{Get-csp : Train} \rightarrow \text{Real}, \text{Set-pid : Platform} \times \text{String} \rightarrow \text{Void}, \text{Get-pid : Platform} \rightarrow \text{String}\}$

2) The Sentences

The general form of the (unique kind of) sentences is defined by the following syntax.

$\text{association}(\text{association-name}, r_1:c_1:mult_1, r_2:c_2:mult_2)$

where association-name is the name of the (binary) association between the classes $c_1$ and $c_2$, $r_1$ and $r_2$ are the roles of the association, and $mult_1$ and $mult_2$ are the corresponding multiplicities. So, the unique sentence for our train control system CIM is given by:

$\text{association}(\text{At, guest : Train : 0..1, hostedBy : Platform : 0..1})$

3) The Models

According to [10], models of class diagrams are defined as sets of object states. Object states are sets of created object identifiers of the declared class names, together with functions that interpret attributes and methods, as well as relations that interpret associations’ names. The attributes values of these sets of object identifiers capture their states.

B) The CIM Intention Institution

According to [10], the signature of an OCL institution consists of a set of class names, query names (attributes and their associated get operations names), and method names. A partial order relation between class names captures class inheritance hierarchies (if any). An extended type system (including the usual primitive data types augmented with the Set, Sequence and Enumeration data types) is used to define a unique type for attribute names and method names.

1) The Signature

The signature is given by $\Sigma = C \cup T \cup O$

$C$ is a set of class names, $C = \{\text{Train, Platform}\}$

$T$ is a set of data types in the CIM,

$T = \{\text{String, Real, Status, Set of Platform, Set of Train}\}$

$O$ is the union of the set of query operations $Q$ and the set of methods,

$Q = \{\text{T-id, P-id, CurrentSpeed, DoorState, guest, hostedBy}\}$

$M = \{\text{Set-tid, Set-pid, Set-csp, Set-dos, Set-guest, Set-hostedBy}\}$

2) The Sentences

Institutions associated to OCL specifications define three kinds of sentences: invariant sentences, and pre-condition and post-condition sentences. For lack of space, we present in the following only the invariant sentences expressing the CIM intention.

context: $t : \text{Train}$

$\text{inv : } t.\text{CurrentSpeed} = 0 \text{ or } t.\text{DoorState} = \text{closed}$

$\text{inv : } t.\text{hostedBy} \neq \emptyset \text{ or } t.\text{DoorState} = \text{closed}$

3) The Models

OCL signatures can be interpreted by collections of models. These models are mathematical structures that interpret OCL specifications signatures. Class names are mapped to a set of
object identifiers while query names and methods names are mapped to a set of appropriate (mathematical) functions.

**C) Linking CIM Institution with CIM Intention Institution**

The link between CIM institution and CIM intention institution is captured by an (embedded) morphism, defined as follows.

1) **Signature Mapping**

The mapping between CIM institution signature and CIM intention institution signature is straightforward: Class names are mapped into class names, (primitive) attributes into queries names, methods into methods’ names and role names into set-valued query names.

**Remark:** Set-valued query names correspond to the get operations returning set of objects (corresponding to reference attributes’ values).

2) **Sentence Mapping**

Because in this version of the paper we limited ourselves only to invariant sentences expressing the CIM intention, we cannot illustrate the mapping.

3) **Correspondance between Models**

According to [10], models of embedded class diagram signatures are extracted from models of OCL signatures by taking the set of states of the OCL models.

**VI. THINGS AND ARROWS VS INSTITUTIONS AND SEMANTIC CONNECTIONS**

Considering things as institutions allows us to capture in a precise and uniform way the meaning of the various kinds of arrows in Table I (column 3). This idea was clarified throughout the previous example and illustrations. Table II summarizes the correspondence between the arrows such as described in Table I and the corresponding arrows used in institutions (i.e., morphisms and co-morphisms). Semantic connections through intermediary institutions use morphisms and co-morphisms as well. As the reader might guess, the example treated in Section V corresponds to the fourth row in Table II.

**Table II: notation used in [1] and institution theory terminology**

<table>
<thead>
<tr>
<th>Intention</th>
<th>Arrows in [1]</th>
<th>Arrows in institutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>I(X) ⊆ I(Y)</td>
<td>X [\mu] Y</td>
<td>Either morphism or co-morphism</td>
</tr>
<tr>
<td>I(X) (\cap) I(Y) = (\emptyset)</td>
<td>X [\mu] Y</td>
<td>No semantic connection</td>
</tr>
<tr>
<td>I(X) (\cap) I(Y) (\neq) (\emptyset)</td>
<td>X [\mu] Y</td>
<td>Semantic connection through an intermediary institution</td>
</tr>
</tbody>
</table>

**VII. CONCLUSION**

In this work we have shown how institution theory, a categorical approach to abstract model theory, might contribute to a better understanding of models and model transformations. Institution theory was introduced intuitively through the concept of things and their intentions developed in [1] in relation with intentional modeling. We have shown that things (seen as dummy SE-models) are in fact a kind of institution where signatures are abstracted to names and Ins-models are “free” interpretations for names, since there are no sentences to satisfy. Building on this idea, we have shown that if names of things are “substituted” by signatures giving rise to sentences expressing a minimum of knowledge on the essence of things (thus becoming SE-models), then defining “concrete” intentions become more feasible, and Ins-models better understood. As a first result, this proposal allowed expressing relations between intentions (denoted by no less than five kinds of arrows, Cf. Table II) by a unique concept: Institution (co)-morphism. A second result consists in the ability to construct the relation between two intentions (of existing models), and not just to characterize this relation as it is the case in [1]. A third result is the possibility to perform model transformations in a semantically coherent way; this is due to the fact that institutions are an abstract model theory with no commitment to particular logical systems. Our next step will be to apply the approach we developed throughout this paper to SE-models within other MDA viewpoints, starting with the PIM. Using Object-Z for this view seems to be adequate for at least two reasons: Object-Z is often used in industry at the design level; an institution for Object-Z does already exist [6].

**ACKNOWLEDGEMENT**

This work is supported by the Deanship of Scientific Research, Philadelphia University Jordan in the framework of the project entitled: An institutional framework for a formal heterogeneous software development methodology based on UML/MARTE, CSP-OZ and PyCSP.

**REFERENCES**


