Abstract: UML/OCL and Object-Z are two technologies complementing each other. They show a "converging" syntax for their basic features, and are distinguished by a "diverging" syntax for their more advanced features. The objective of this paper is twofold. First, we show that for the basic features the mapping between both languages is a co-morphism between their corresponding institutions. Second, we identify the advanced features in order to characterize closely the type of a potential mapping between both languages. Our approach is demonstrated through a simple example related to human resources' management in companies.

Keywords: modeling; formal specification; UML; UML/OCL; Object-Z; institutions; models' mapping

I. INTRODUCTION

This work emerged as a consequence of previous contributions [12,15], where we were aiming at relating (at the institutional level) UML/OCL and Object-Z. The results of our investigations thus showed that the relation between both technologies is manifold and depends on the kinds of features one is interested in. The problem consists, first to identify the features one is interested in, and then to establish the “right” relationship. The “right” question to ask however is how to establish this relationship, and how one might be sure that this relation is semantically consistent. In this work we propose to answer such a question through the use of the formal language of institutions [3,11,6]. The language of institutions permits to back each technology with its “natural” semantics, and then to relate both technologies at various levels (vocabulary, constraints, and semantics), guaranteeing semantic consistency independently of any change in the vocabulary. According to [14], mapping of UML/OCL into Object-Z appears to be "straightforward" for basic features, because of a big convergence between both syntaxes for such features. However, talking about a potential mapping becomes more problematical for some advanced features because of a "divergence" in the syntaxes used in both languages. In addition to these findings, little attention is paid to the kind of mapping corresponding to the wording "straightforward" and "divergence" (out of an intuitive discussion).

The objective of this work is twofold. Firstly, we show that the straightforward mapping between UML/OCL and Object-Z is a co-morphism between the institutions backing both languages; secondly, we show that for the subset of advanced features, the mapping depends on the existence (or even the construction) of an intermediary language (that has to be backed by an institution). The concrete nature of the mapping depends on the nature of the mapping between UML/OCL and the identified intermediary language on one hand, and between Object-Z and this intermediary language on the other hand.

The rest of the paper is organized as follows. Section II is devoted to basics in institutions. In section III, we present a specification of a human resources’ management example over an OCL institution. Section IV presents a specification of the same example over an Object-Z institution. In section V we show how to write refined specifications of the same example over both institutions. Section VI discusses our findings on the raised mapping issues and their potential solutions. In section VII we present related works, and in section VIII we draw conclusions and outline future work.

II. INSTITUTIONS

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 institutional models (Ins-models), and
- a satisfaction relation of sentences by Ins-models, such that when signatures are changed by signature morphism, satisfaction of sentences by Ins-models changes consistently.

As stated in [7], 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, database relational schema models [8, 17], and Object-Z models [4,10].

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 [3, 7].
Signatures are used to define the vocabularies 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.

To show how to associate institutions to UML class diagrams and to OCL we use results presented in [17] and [8]. We also build on [4] to show how to associate institutions to Object-Z specifications. The reader is supposed to be acquainted with UML and to have a minimum knowledge on Object-Z [10].

III. UML/OCL

The Unified Modeling Language (UML) is a popular OMG (object-oriented) technology widely used in industrial environment. UML is endowed with a set of complementary diagram types geared to the representation and documentation of various (static and dynamic aspects) of software systems. Class diagrams, considered as the pillar of UML, are used for various purposes going from domain modeling to the modeling of the UML itself (metamodeling). Using class diagrams, a system might be merely represented in terms of its classes (declaring attributes and operations) and relations among them. Association, aggregation, composition and inheritance are examples of such relations. UML was quickly adopted from its very beginning by diverse communities (engineering modeling, simulation modeling, etc.) probably for its visual appeal, and its ability to sketch in a reasonable time a blueprint of the system under consideration. However as other semi-formal languages and approaches, UML in general and UML class diagrams in particular suffer from a lack of preciseness that can be felt especially when it comes to refine an overall specification into a detailed design. This is behind the idea of integrating the Object Constraint Language (OCL) into UML. Constraints in OCL are specified using invariants that are used to convey some restrictions as regards to some classes’ attributes and pre/post conditions as regards to the applicability and result of classes’ operations.

The following example, based on [1], intending a simple class diagram, illustrates the basic concepts of UML class, UML association, and OCL expression.

The intended class diagram consists of the classes Employee and Company, and of an association, designated by 'works for', that relates the two classes. The Employee class includes the attributes ssn (social security number), and name (employee name), and the query operations Get-ssn() and Get-name().

The Company class includes the attributes cid (identifier of the company), activity (kind of the company business), and the query operation Get-cid (), and the non-query operation Hire (e: Employee).

The 'works for' association has two association ends, employee and employer. Here we assume in this example that employees cannot work in more than one company at the same time, and companies have less than eleven employees.

A. OCL Specifications of the Human Resources' Management Example

The following OCL expressions may be associated with the intended UML class diagram.

Invariants:
context Company
inv: self.employee \rightarrow size() \leq 10
context Employee
inv: self.employer \rightarrow size() \leq 1
Init:
context Employee:: employer: Company
init: self.employer \rightarrow size() = 0
context Company:: employee: Employee
init: self.employee \rightarrow size() = 1
Pre/Post Invariants:
context Company :: Hire (e: Employee): void
pre: not (self.employee \rightarrow includes (e)) and (self.employee \rightarrow size()) < 10
post: self.employee = self.employee@pre union \{e\}
context Employee:: Get-name(): String
post: result = self.name

B. OCL Institution: Illustration through the Human Resources’ Management Example

1. OCL Signature

According to [5,7,8], the OCL signature is defined by
\[ \Sigma = C \cup T \cup O \] where:
- \( C \) is a set of class names, \( C= \{ \text{Employee, Company} \} \)
- \( T \) is a set of data types, \( T= \{ \text{String, Set of Employee, Set of Company, void} \} \)
- \( O \) the union of the set of query operations \( Q \) and the set of methods \( M \): \( Q= \{ \text{ssn, name, cid, activity, employee, employer} \} \), \( M= \{ \text{Hire} \} \).

2. OCL Sentences

Three kinds of sentences are defined in the OCL institutions: pre/post sentences, invariant sentences, sentences related to association multiplicities and navigability.
- Pre/Post sentences
  context Company:: Hire (e: Employee): void
  pre: not (self.employee -> includes (e)) and
       (self.employee ->size()) < 10
  post: self.employee = self.employee@pre union {e}
  context Employee:: Get-name (): String
  post: result = self.name
- Association multiplicities/navigability sentences:
  context Employee
  inv: self.employer -> count() ≤ 1
  context Employee
  inv: self.employer -> forall (x: Company| x.employee -> contains (self))
  context Company
  inv: self.employee -> count() ≥ 1
  context Company
  inv: self.employee -> forall (x: Employee| x.employer -> contains (self))
IV. OBJECT-Z
Like UML, Object-Z [9] is a semi-graphical object-oriented technology. Unlike UML Object-Z is formal language, and like other formal languages, it uses a precise mathematical notation (i.e., a notation with a formal syntax and semantics) to describe software and related systems. A system is described by a so-called system class and a set of classes (called classes of the system). Each class (of the system) defines the behaviors of its individual objects, whilst the system class captures and defines the behavior of all the individual objects resulting from the instantiation of all the classes (of the system) taking into account their interactions [13]. Grosso-modo, an Object-Z class declares a set of state variables (attributes) and operation schemata (operations); constraints on state variables and operation schemata are expressed by logical predicates. Unlike UML, constraints are an integral part of Object-Z and there is no need to contextualize these constraints as this is the case with OCL.

A. Object-Z Specification of the Human Resources’ Management Example

The Object-Z specification of our example (Fig. 1 and Fig. 2) includes the Employee and Company classes. Both classes specify initial schemas: employer = Ø, # employee = 1, and state invariant: # employer ≤ 1, # employee ≤ 10.

The operation schema associated to the Hire operation makes explicit the attributes of the Company class that change and those that do not change (Δ feature).

Fig. 1 Object-Z Specification of the Employee Class

<table>
<thead>
<tr>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>cid : String</td>
</tr>
<tr>
<td>activity : String</td>
</tr>
<tr>
<td>employee : ℙ Employee</td>
</tr>
<tr>
<td>#employee ≤ 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hire</th>
</tr>
</thead>
<tbody>
<tr>
<td>#employee → 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Get – cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>employee : ℙ Employee</td>
</tr>
<tr>
<td>#employee ≤ 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>e? : Employee</th>
</tr>
</thead>
<tbody>
<tr>
<td>employee ≠ e?</td>
</tr>
</tbody>
</table>

Fig. 2 Object-Z Specification of the Company class

B. Object-Z Institution illustrated through the Human Resources’ Management Example

Institutions for Object-Z have been defined in [4,10].

1. Signature
A signature is a triple Σ = (S, F, π)
- S = C ∪ T is a set of class names (C), type names (T), and π is a set of polymorphic class names.
- For our example we have: C = {Employee, Company}; T = {String}, π is empty.
- F = B ∪ R ∪ O is a set of operations representing primitive attributes (B), reference attributes (R), and operation schemata (O).

For our example we have:

B = {BEmployee ->String, BCompany ->String }
B Employee ->String = {ssn, name}; B Company ->String = {cid, activity}
R= {REmployee ->PCompany, RCompany ->PEmployee}
R Employee ->PCompany = {employer}
R Company ->PEmployee = {employee}
O = {O Employee, <->, <String>, O Company, <->, <String>, O Company, <->, Employee}
O Employee, <->, <String> = {Get-ssn, Get-name}
O Company, <->, <String> = {Get-cid}
O Company, <->, Employee = {Hire}
- π = Ø (no polymorphic sentences)
2. Sentences

Object-Z institutions include three kinds of sentences: Sentences for initial state schema, sentences for state schema, and sentences for operation schema.

- Sentences for initial schema
  \[ \text{Init}_C : P \]
  \[ \text{Init}_{\text{Employee}} : \forall e: \text{Employee}. e.\text{employer} = \emptyset \]
  \[ \text{Init}_{\text{Company}} : \forall c: \text{Company} . \# c.\text{employee} = 1 \]

- Sentences for state invariant
  \[ \text{Inv}_C : P \]
  \[ \text{Inv}_{\text{Employee}} : \forall e: \text{Employee}. \# e.\text{employer} \leq 1 \]
  \[ \text{Inv}_{\text{Company}} : \forall c: \text{Company} . \# c.\text{employee} \leq 10 \]

- Sentences for operation schema
  \[ o(\text{id}_1?: \text{id}_1, \ldots, \text{id}_n?: \text{id}_n, \text{id}_1!: \text{id}_1, \ldots, \text{id}_m!: \text{id}_m) [\text{id}_1, \ldots, \text{id}_m] : P \]
  where \( o \) is the operation name, \( \text{id}_i?: \text{id}_i \) are input parameter names with their type names, \( \text{id}_i!: \text{id}_i \) are output parameter names with their type names, \( [\text{id}_1, \ldots, \text{id}_m] \) represents the list of attributes modified by the operation, and \( P \) a predicate representing the pre/post conditions.

\[ \text{Get-name} (n!: \text{String}) [\cdot ] : n! = \text{name} \]
\[ \text{Hire} (e?: \text{Employee}) [\text{employee}] : e? \in \text{employee} \land \# \text{employee} < 10 \land \text{employee'} = \text{employee} \cup \{e?\} \]

V. OCL AND OBJECT-Z SPECIFICATIONS OF THE REFINED VERSION OF THE HUMAN RESOURCES' EXAMPLE

The previous example is now refined by replacing the “works for” association by an association class Job whose intent is to give more details on this association. The Job class includes the attributes position and salary, and a query operation called Get-income.

Employees, in this example, may work for more than one company.

A. The OCL Institution

In this refined example, the association class Job, through its (implicit) association end named job, ensures the navigability between the Company and Employee classes.

1. The OCL Specification

The OCL expressions corresponding to this modified version of the example are given below:

Invariant:
\[ \text{context Employee} \]
\[ \text{inv: self.job}\rightarrow \text{size()} > 0 \]
Init:
\[ \text{context Company:: job: Job} \]
\[ \text{init: self.job}\rightarrow \text{size()} = 1 \]
Pre/Post Invariants:
\[ \text{context Job:: Get-income (bonus: Real): Real} \]
\[ \text{post: result = Tuple (bonus = self.salary \times 10/100, result = self.salary + bonus)} \]

OCL does not allow more than one return result. The OCL type constructor Tuple permits to overcome this issue.

2. OCL Institution: Illustration Through the Refined Example

Signature
\[ C= \{\text{Employee, Company, Job}\} \]
\[ T= \{\text{String, Real, Void}\} \]
\[ Q= \{\text{ssn, name, cid, salary, position, Set of Job, Set of Employer, Set of Company}\} \]
\[ M= \{\text{Hire, Get-income}\} \]

Sentences
Here we focus only on the new added sentences.
Post invariants:
\[ \text{context Job:: Get-income (bonus: Real): Real} \]
\[ \text{post : result} = \text{Tuple (bonus} = 10*\text{salary}/100, \text{salary} = \text{salary@pre + bonus)} \]

Sentences for association multiplicities and navigability sentences
\[ \text{context Job} \]
\[ \text{inv: self.employer}\rightarrow \text{count()} = 1 \]
\[ \text{context Job} \]
\[ \text{inv: self. employee}\rightarrow \text{count()} = 1 \]
\[ \text{context Employee} \]
\[ \text{inv: self.job} \rightarrow \text{count()} \geq 1 \]
\[ \text{context Company} \]
\[ \text{inv: self.job} \rightarrow \text{count()} \leq 10 \]
\[ \text{context Employee} \]
\[ \text{inv: self.job.employer} \rightarrow \text{forall (x: Company| self.job.employee} \rightarrow \text{contains (self)}) \]

B. The Object-Z Institution: Illustration through the Refined Example

1. The Object-Z Specification

\[ \text{Employee} \]
\[ \text{ssn} : \text{SSN} \]
\[ \text{name} : \text{String} \]
\[ \text{jb} : \text{P-Job} \]
\[ \# jb \geq 0 \]
\[ \# jb = 0 \]
\[ \text{Get}\rightarrow \text{ssn} \]
\[ \text{sl} : \text{SSN} \]
\[ \text{sl} = \text{ssn} \]

Fig. 3 Object-Z Specification of the Modified Employee Class

The following shows the modified specifications of the Company and Employee classes (Fig. 3, Fig. 4) and the “works-for” association. The employee and employer attributes of the previous Company and Employee classes are now respectively replaced by the attributes jbe and jb. These
last two attributes refer to collections of instances of the new introduced Job class (Fig. 5). Specific user defined base types (SSN, and CID) are introduced to address the types of the ssn and cid attributes in a more appropriate way. OCL does not support the concept of user-defined base types.

Fig. 4 Object-Z Specification of the Modified Company Class

Fig. 5 Object-Z Specification of the Job Class

Beside the position and the salary attributes, the Job class includes two extra attributes (employee and employer) completing the specification of jobs. Each of these extra attributes is constrained by an invariant (a given job relates one employer with one employee). The operation Get-income has two output parameters and both are separately returned (predicate part of the operation schema). The specification of the modified example over the Object-Z institution is given by the following.

Signature

\begin{align*}
C &= \{ \text{Employee}, \text{Company}, \text{Job} \} \\
T &= \{ \text{String}, \text{Real}, \text{SSN}, \text{CID} \} \\
B &= \{ B_{\text{Employee}} \rightarrow \text{String}, B_{\text{Employee}} \rightarrow \text{SSN}, B_{\text{Company}} \rightarrow \text{CID}, \\
& \quad B_{\text{Company}} \rightarrow \text{String}, B_{\text{Job}} \rightarrow \text{String}, B_{\text{Job}} \rightarrow \text{Real} \} \\
B_{\text{Employee}} &= \{ \text{name} \}; \quad B_{\text{Employee}} \rightarrow \text{SSN} = \{ \text{ssn} \} \\
B_{\text{Company}} &= \{ \text{activity} \}; \quad B_{\text{Company}} \rightarrow \text{String} = \{ \text{activity} \} \\
B_{\text{Job}} &= \{ \text{position} \}; \quad B_{\text{Job}} \rightarrow \text{Real} = \{ \text{salary} \} \\
R &= \{ \text{R}_{\text{Employee}} \rightarrow \text{P}_{\text{Job}}, \text{R}_{\text{Company}} \rightarrow \text{P}_{\text{Job}}, \\
& \quad \text{R}_{\text{Job}} \rightarrow \text{P}_{\text{Employee}}, \text{R}_{\text{Job}} \rightarrow \text{P}_{\text{Company}} \} \\
R_{\text{Employee}} &= \{ \text{jb} \}; \quad R_{\text{Company}} = \{ \text{jbe} \} \\
R_{\text{Job}} &= \{ \text{employee} \}; \quad R_{\text{Job}} \rightarrow \text{P}_{\text{Company}} = \{ \text{employer} \} \\
O &= \{ O_{\text{Employee}}, \text{<}, \text{SSN}> \}; \quad O_{\text{Company}}, \text{<}, \text{CID}> \\
& \quad O_{\text{Company}}, \text{<}, \text{employee} \} = \{ \text{Get-ssn} \} \\
& \quad O_{\text{Company}}, \text{<}, \text{<CID>}, \text{<Real>} = \{ \text{Get-cid} \} \\
& \quad O_{\text{Company}}, \text{<}, \text{<Real>} = \{ \text{Hire} \} \\
& \quad O_{\text{Job}}, \text{<}, \text{<Real>} = \{ \text{Get-income} \} \\
\end{align*}

Sentences

- Init sentences:
  \begin{align*}
  \text{Init}_{\text{Employee}} & : \forall e: \text{Employee}. \# e.jb = 0 \\
  \text{Init}_{\text{Company}} & : \forall c: \text{Company}. \# e.jbe = 1 \\
  \end{align*}

- Sentences for state schema invariant:
  \begin{align*}
  \text{Inv}_{\text{Employee}} & : \forall e: \text{Employee}. \# e.jb \geq 0 \\
  \text{Inv}_{\text{Company}} & : \forall c: \text{Company}. \# e.jbe \leq 10 \\
  \text{Inv}_{\text{Job}} & : \forall j: \text{Job}. \# j.employer = 1 \\
  \text{Inv}_{\text{Job}} & : \forall j: \text{Job}. \# j.employer = 1 \\
  \end{align*}

- Pre/Post sentences
  \begin{align*}
  \text{Get-ssn} & : \text{ssn} \text{! : SSN} \rightarrow \text{ssn} \\
  \text{Get-cid} & : \text{cid} \text{! : CID} \rightarrow \text{cid} \\
  \text{Hire} & : \text{employee}! : \text{Employee} \rightarrow \text{job} \notin \text{job} \\
  \text{Hire} & : \text{employee}! : \text{Employee} \rightarrow \text{job} \notin \text{job} \\
  \text{Get-income} & : \text{bonus}! : \text{Real}, \text{income}! : \text{Real} \rightarrow \text{bonus}! = 10*\text{salary}/100 \land \text{income}! = \text{salary} + \text{bonus}! \\
  \end{align*}

VI. DISCUSSION

A slight examination of the UML/OCL and Object-Z institutions illustrated through examples show similarities between them. In fact these institutions capture the essence of some (common) core features of both languages. Indeed both languages support, in a native or in a non-native way, the notion of class, attribute, operation, data type, operations' pre/post conditions, in/out operation parameters and return result, associations (in their diversity), and invariants (in their diversity). The similarity or proximity between these core features makes the mapping between these institutions more or less straightforward.

However a deep insight inside these two languages reveals some divergent features, divergent in the sense that features supported by one of these languages have not their "equivalent" in the other language, and vice-versa.
On one side Object-Z specifically supports the notion of user defined base type, the ability to combine operations (conjunction, sequential, parallel), the notion of system class, the polymorphism, the union data type, and others. On another side UML/OCL specifically supports the notion of stereotype class, message/signal, undefined values, and others. According to the upper elicitated divergence in some of the identified features, we suggest a solution to the raised mapping issues based on the use of the concept of co-span of institutions.

VII. RELATED WORKS

In [13], the authors define a formal mapping between UML and Object-Z and show the correctness of their mapping using metamodeling. The authors in [14] present a mapping of UML/OCL into Object-Z. The mapping between UML class diagrams into Object-Z is based on the results in [13]. The authors identify features where mapping of UML/OCL into Object-Z appears to be "straightforward" for basic features, whilst it needs more elaboration or other advanced features because of a "divergence" in the syntaxes used in both languages. Out of these findings, little attention is paid to the kind of mapping corresponding to both kinds of situations. However to the best of our knowledge no work has so far addressed the mapping of UML/OCL into Object-Z from the institutional point of view.

VIII. CONCLUSION

In this paper we first showed how to express in a consistent way the relation between UML/OCL and Object-Z (in their basic versions) by a co-morphism of institutions backing both of these technologies. Semantic consistency is demonstrated through the use of the language of institutions. The language of institutions allows indeed to provide each technology with its "natural" semantics and to relate formally both semantics at the level of the vocabulary specific to each technology, at the constraints' level, and at the semantic level independently of any change in the used vocabulary. Then we identified a subset of diverging features in order to characterize closely the type of mapping between both languages when used in their "full" versions. According to the identified features it appears that some of the features of UML/OCL are not supported by Object-Z and vice-versa. From this we concluded that this issue might be solved through the use of the concept of co-span of institutions. Our findings were demonstrated through a simple example related to human resource management in companies.

As future work we project to build a "richer" institution "enclosing" both UML/OCL and Object-Z in their full version, and then to construct the appropriate co-span. The objective is to take advantage of both technologies by offering a "broadband" language [16] allowing to tackle the software development process in an evolutionary way.

ACKNOWLEDGEMENT

This work is supported by the Deanship of Scientific Research, Philadelphia University.

REFERENCES


83