REPRESENTATION OF PROCEDURAL KNOWLEDGE AND ITS USE TO COMPUTE A FORM OF SUBSUMPTION

Yamine AIT AMEUR
LISI/ENSMA-Université de Poitiers
BP 109. Site du Futuroscope. 86960 Futuroscope Cedex. France.
yamine@ensma.fr

Abstract This paper shows the possibility to describe language based representations by model based representations. This uniform representation is useful for model integrations and helps to compute a form of subsumption.

Keywords: Data modelling languages, Procedural knowledge, Derivation, Constraints.

1. Introduction

Nowadays, computer aided engineering activities like specification, design, maintenance, simulation, etc. involve the use of electronic data which require the use of complex data models. These developments led to a wide class of heterogeneous data models. As a consequence, the processes of sharing, exchanging and integrating data and data models become complex as well. The origin of the work outlined in this paper is CAD which involves several industrial parts descriptions defined by several part suppliers and supported by several CAD systems and LMS (Library Management Systems). Usually these parts need to be assembled (or composed) in order to incrementally build more complex parts which are themselves stored in such LMS in order to be shared and/or exchanged. Thus, part descriptions are heterogeneous.

Meta-modelling and model transformation techniques have been put into practice in order to promote model integration and reduce heterogeneity. We propose a meta model approach allowing to handle procedural knowledge in order to increase integration of data models which use procedural knowledge for derivation and constraint expressions and we give the benefits of this approach for computing a particular form of subsumption.

Before giving the overview of our approach, let us review the following toy example. Let us consider two descriptions of a screw. *Screw_R* defined by

the *Head_Ray* and *Length* properties and *Screw_Diameter* defined by the *Head_Diameter* and *Length* properties. These two descriptions of a common screw are different. However, the expression $Head\_Diameter = 2 * Head\_Ray$ (Resp. $Head\_Ray = Head\_Diameter/2$) may be used in order to relate the two descriptions and integrate them. Indeed, using the expression $Head\_Diameter = 2 * Head\_Ray$ (Resp. $Head\_Ray = Head\_Diameter/2$) instances of *Screw_R* (Resp. *Screw_D*) may become instances of *Screw_D* (Resp. *Screw_R*) encoding here a form of subsumption. Therefore, there is a need of representing in the data models to be integrated the expression $Head\_Diameter = 2 * Head\_Ray$ (Resp. $Head\_Ray = Head\_Diameter/2$). More generally, there is a need of representing the procedural knowledge either for encoding not only expressions like in this simple example, but also possible constraints (logical expressions) on data (e.g. $Head\_Diameter < 2 * Length$).

Continuing this example, two questions arise: (1) where are these expressions represented in the data model ? (2) How are they represented ?

At the ontological (dictionary) level is the answer to the first question. Indeed, the properties of *Head_Diameter* and *Head_ray* associated to the concept of a *Screw_D* or a *Screw_R* (classes of instances) shall be uniquely and universally defined in a shared ontology (or dictionary). For example, they can be represented as derived attributes. In this paper, we do not focus on the ontology representation aspect, we suppose that such an ontology exists. The reader may refer to (Pierra, 2004) available in these proceedings for ontology modelling.

Several different answers for the second question are possible (abstract syntax, languages, data-models, etc.). We have chosen to represent the procedural knowledge using data models. The main reason is uniformity of the representation. Indeed, both structural, descriptive and procedural knowledge are represented using the same technology, the same modelling language and the same data model independently of any programming language and platform. We do not separate the data models from the models for expressions. Indeed, expressions become properties (encoded in derived attributes or in constraints) of the elements they contribute to describe.

Finally, one can continue this toy example by introducing another level of heterogeneity that can be solved in the same manner by associating heterogeneous units to the properties (centimeters for the *Head_Ray* and inches for the *Head_Diameter*.

The rest of this paper gives an overview of our approach for representing procedural knowledge using data models and meta-modelling techniques issued from the work developed by (Bernstein, 2003,Ait-Ameur et al., 2000, Ait-Ameur et al., 1995b, Ait-Ameur et al., 1995a and Ait-Ameur and Wiedmer, 1998).

2. Modelling and language representation

In classical data engineering, data models are built in order to capture and to formalize the structure, the semantics, the representation, etc. of a set of data they intend to characterize. Different categories of knowledge are represented in these models (structural, descriptive and procedural). The problem of handling heterogeneous models and integrating them has been addressed by several researchers issued from different research areas (database systems, knowledge engineering, CAD systems etc.). In this context approaches promoting integration have been proposed (standards, exchange formats, meta-models, etc.).

Formal modelling languages are used to represent and to encode these data models. They offer the possibility to assert that given data, commonly named instances, fit with a given data model. These languages offer a set of basic modelling concepts (objects, attributes, derived attributes, constraints, etc.) allowing to describe data models which represent structural, descriptive and procedural knowledge. Again the use of these languages leads to a wide variety of heterogeneous data models that need to be integrated.

The integration data models requires to be able to formally talk about all the concepts described by these models. In practice, one can consider that models are represented as instances of meta models (themselves expressed in a modelling language) shared by all the parties involved during integration. The possibility to express complex data models depends on the richness of the meta-model which depends itself on the modelling power offered by the modelling language. According to (Bernstein, 2003), we define a model to be the set of objects, each of which has properties, and relationships between objects. Objects, properties and relationships have types. Three layers are required to describe the whole model technology.

1- *models* are defined by instances of objects, properties and relationships. Models definition is based on an explicit set extension description;

2- *meta-models* are the type definitions for the objects of the models. They describe the types the different objects belong to;

3- *meta-meta-models* is the language where models and meta-models are expressed.

The complexity of meta-model descriptions depends on the meta-meta-model language. The more this language is powerful, the more models are expressive. The richness of the models depends on the kind of knowledge that can be written within this language (availability of procedural, structural and descriptive knowledge). Model management systems are systems that implement models, meta-models and operators between models (Bernstein, 2003).

The models, we are dealing with in this paper, are sets of objects, with a root object, each of which has an identity and attributes (properties). Objects

can be related by the *is_a* relationship (generalization-specialization), *has_a* relationship (part-of). Objects, attributes and relationships can be typed by built-in types or can be constrained. The typing expressions depend on the typing power supplied by the meta-model and by the meta-meta-model.

Notice that the concepts introduced in the meta-model are present in almost all the available modelling languages, and depending on the meta-meta-model, it is possible to define complex types, constraints, derivation functions and so on, that will be used at the meta-model level.

Language representation. Procedural knowledge is described by a language of expressions involving operands and operators, allowing to express either derivations or constraints. Encoding this kind of knowledge requires representing the grammar to derive all the suited expressions. Let us consider a production rule which describes simple numeric expressions defined by $E ::= E + E|E * E|Cst|Var$. If we consider a modelling language with the *is_a* relationship then, figure 1 represents the previous production rule. A root concept (inside a box) represents an expression *E*. *Add_E* and *Mul_E* are concepts (represented inside boxes) for addition and multiplication with two attributes that are expressions, introducing recursive descriptions. Consider now,

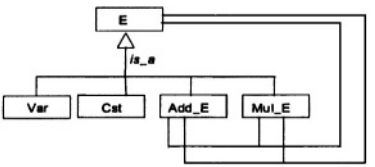

Figure 1. A simple example of a model for simple expressions with one non terminal and recursion.

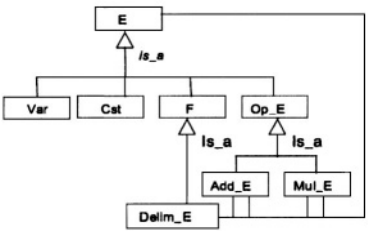

Figure 2. A simple example of a model for simple expressions with two non terminals and recursion.

the production rules $E ::= E\ op\ E|Cst|Var|F$, $op ::= +|*$ and $F ::= (E)$ with a case of indirect recursion. Using the same modelling language, figure 2 shows the translation of this set of production rules.

In both cases, a finite graph represents these production rules. The *is_a* relationship allows to encode the left and right hand sides of a production rule. Recursion is represented either by direct referencing or by indirect referencing through intermediate non terminals. These two simple examples show that it is possible to encode classical BNF grammars in modelling languages with a small set of basic concepts.

Formal grammars. The previous approach can be generalized to handle any kind of BNF grammar. A BNF grammar G is classically defined by a 4-tuple $G = (T, N, A, P)$. A production rule $F ::= E_1 | \cdots | E_n$, where $F \in N$ and $E_i \in (T \cup N)^*$, allows to rewrite the non terminal symbol F into one of the E_i. Using the same mechanism as described previously, it is possible to represent any production rule issued from such grammars and encode it by a data model with F as root.

Static semantics. In classical language processing, we denote by static semantics all the information that can be expressed (i.e. computed) on models in their static context. Usually, analyzers, type checkers, model checkers, theorem provers, abstract interpretation, are used to perform such a static analyses. In models static analysis is expressed by analyzing the relevant information embedded in the concepts. The set of instances of the model is not built nor provided.

Several features related to static semantics can be encoded for such grammars. Among them: enrichment of models by adding attributes using the *is_a* relationship for example, enrichment of models by adding types for constraining attributes domains, enrichment of models by adding constraints which express relevant static properties of the model (for example asserting that the underlying graph of an expression is a DAG). All these enrichments can be written although the set of all the instances is not known (static aspect).

Dynamic semantics and evaluation. In programming languages dynamic semantics is related to execution, interpretation and evaluation. In relational database systems, dynamic semantics is related to query executions. In model based representations, the expression of dynamic semantics is possible when instances of models are available and are evolving dynamically. In this case, adding, removing, modifying, querying of instances become possible. It is required that the meta-meta-modelling language is equipped with instance management operators. Depending on the power of the procedural knowledge encoded in the meta-meta-model language, it is possible to reach the level of a programming language or of a verification procedure, etc. If we come back to our example of figures 1 and 2, dynamic semantics can be expressed by writing an evaluator of an expression described in the concept *E.* To write this evaluator, it is necessary to have an instance of the model described by these figures and values associated to the variables.

Required characteristics for the meta modelling language. Naturally, the meta-meta-model language shall be powerful enough to support a logic allowing to encode derived attributes and to write and check constraints. The expression power and the resolution procedure of this logic express the kind

of constraints that can be described. We have experimented the EXPRESS language (Schenck and Wilson, 1994) which offers, in a common language, all these features and an operational available tool.

3. Conclusion

This paper showed the importance of modelling the procedural knowledge for data integration processes. It suggested a data model oriented approach allowing to encode expressions and more generally grammars by data models. The advantage of such a representation is the possibility to have all the data model characteristics encoded in a single and common framework based on data models and data modelling languages. When used in an ontology, it allows to describe expressions transformations for subsumption purposes. We have experimented this approach in several projects by encoding complex data models for representing procedural knowledge. Moreover, we have used this approach for encoding the procedural knowledge of the PLIB ontology (Ait-Ameur et al., 2000, Ait-Ameur et al., 1995b, Ait-Ameur et al., 1995a and Ait-Ameur and Wiedmer, 1998). In this work, generic expressions, numeric, boolean, string expressions, expressions for aggregates and relational database expressions were encoded using this approach.

References

Ait-Ameur, Y., Besnard, F., Girard, P., Pierra, G., and Potier, J. C. (1995a). Formal Specification and Metaprogramming in the EXPRESS Language. In *International Conference on Software Engineering and Knowledge Engineering,* pages 181–189. KSI Institute in Corporation with ACM Sigsoft and IEEE.

Ait-Ameur, Y., Pierra, G., and Sardet, E. (1995b). Using The EXPRESS Language For Metaptogramming. In *Express User Group Conference.* Grenoble, France.

Ait-Ameur, Y., Pierra, G., and Sardet, E. (2000). An Object Oriented Approach to Represent Procedural Knowledge in Heterogeneous Information Systems. In *International Conference on Object Oriented Information Systems, OOIS'2000,* pages 303–315. Choudhury-Patel-De Cesare, Editors, Springer Verlag.

Ait-Ameur, Y. and Wiedmer, H. (1998). *General resources.* ISO-IS 13584-20. ISO Geneve, 88 pages.

Bernstein, P. (2003). Applying Model Management to Classical Meta Data Problems. In *Proceedings of the CIDR conference.*

Pierra, G. (2004). The PLIB ontology-based approach to data integration. In *Topical day on Semantic Integration of Heterogeneous Data.* IFIP World Computer Congress, Toulouse - France.

Schenck, D. and Wilson, P. (1994). *Information Modelling The EXPRESS Way.* Oxford University Press.