Exploiting ODB-Tools as a tool for the design of electrical systems

Sonia Bergamaschi[1,2], Maurizio Vincini[1]
(1) University of Modena, DSI (2) University of Bologna - CSITE-CNR
Via Campi 213/B - 41100 Modena Viale Risorgimento, 2 - 40136 Bologna
e_mail: {sonia,vincini}@dsi.unimo.it

Abstract

This paper presents the use of the OLCD (Object Languages with Complements allowing Descriptive cycles) Description Logic in the modelling and design phases of industrial electrical systems.
We propose the use of a DL-based system, ODB-Tools (available in internet at http://sparc20.dsi.unimo.it), to aid the developer in the design and consistency check activity of the system.

ODB-Tools is an integrated environment for object oriented database (OODB) schema validation and semantic query optimization, preserving taxonomy coherence and performing taxonomic inferences.
The approach of the tool is based on the OLCD description logic proposed as a common formalism to express class descriptions, a relevant set of *integrity constraints* (IC rules) and queries. Description Logic inference techniques are exploited to evaluate the logical implications expressed by IC rules and thus to produce the minimal taxonomy of the schema w.r.t. inheritance. ODB-Tools is a ODMG 93 [1] compliant tool, both for the schema definition (ODL language) and for the query language (OQL), and supports an on-line graphical interface developed in Java language.
The effectiveness of ODB-Tools for electrical system design is shown by means of a real application modelling example.

1 Motivation

Description Logics languages - DLs [1], derived from the KL-ONE model [2], have been proposed in the 80's in the Artificial Intelligence research area. DLs are fragments of the first order logic: they enable *concepts* to be expressed, so that they can be viewed as logical formulas built using unary and binary predicates, and contain one free variable (to be filled with instances of the concept).

By exploiting defined concepts semantics, and, given a *type as set* semantics to concept descriptions, it is possible to provide reasoning techniques: computing *subsumption* relations (i.e. "isa" relationships implied by concept descriptions) and detecting *incoherent* (i.e. always empty) concepts. In this paper we describe the OLCD DL [3, 4], extended with *integrity constraints* and *method* definition, to improve the DLs reasoning techniques applied (incoherence and subsumption).

These techniques are profitable for database design activities: if we map a database schema including only classes (no views) into one of the DLs supported by a system, we are able to automatically detect incoherent classes. Incoherence of a schema can thus be avoided by preventing the introduction of incoherent classes. A more active role can be performed with the introduction of views. Given a new view, it can be automatically *classified* (i.e., its right place in an already existing taxonomy can be found) by determining the set of its most specific subsumer views (*subsumers*) and the set of its most generalized specialization views (*subsumees*). Therefore, beside a passive consistency check, *minimality* of the schema with respect to inheritance can easily be computed. We have developed a prototype system based on OLCD, called ODB-Tools [5, 6] to aid the developer in the design and consistency check activity of the system.

The paper shows the effectiveness of ODB-Tools in the modelling and design phases of a real indus-

[1] *DLs* are also known as *Concept Languages* or *Terminological Logics*.

trial electrical systems. It will be cleared out how the consistency check techniques could be be easily used for the system data model design, thanks to the ODB-Tools ODMG-93 [1] compliant interface.

The paper is organized as follows: Section 2 briefly introduces the electrical systems design phases, while Section 3 describes OLCD and the consistency check techniques. In Section 4 ODB-Tools prototype is illustrated and in Section 5 we report some remarks about the use of ODB-Tools in real application modelling domain. For a complete report of the design of the application see [7] and http://sparc20.dsi.unimo.it/prelet. Finally, in Section 6, you can find the conclusions.

2 The Elect-Designer Project

Nowadays, the core of industrial manufacture is represented by high technology and automation. Therefore, Elect-Designer Project's main aim is to aid the designer in the design of electrical systems.
The modern manufacturing automation systems are made up by:

- Decision making center, composed of a series of PLC (Programmable Logic Controller) connected together;
- Peripherals, divided into actuators and sensors. Actuators are the electrical motors that translate the incoming signal (produced by the PLCs) into mechanical action. Sensors are able to translate into electrical signals the physical and mechanical quantity measured on field;
- Electrical system: the connection wires among decision making center and peripherals.

Briefly, an industrial automated system design is organized in the following fundamental phases:

1. System layout definition (mechanical specification);
2. Choice of Primary Components (sensors and actuators) by the designer;
3. Choice of Subcomponents useful for primary component (automated phase);
4. Choice of Auxiliary Components: control box, electrical panel, console, button panel, ...;
5. System layout definition (electrical specification);
6. Component specification and cash budget;
7. PLC software programming.

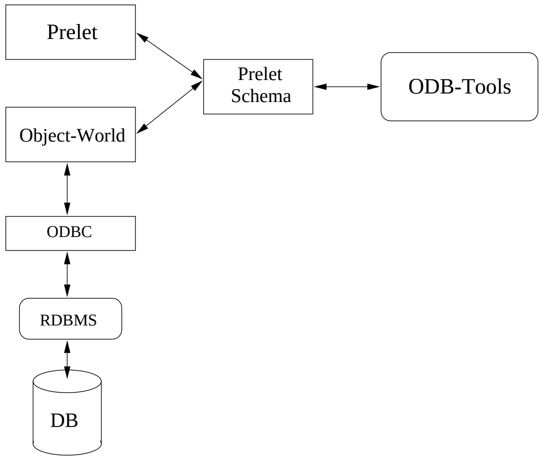

Figure 1: Elet-Designer System

The goal of the Elect-Designer Project is to cover the entire electrical project design cycle, from the system's mechanical layout to the component's specification, via primary component and subcomponent choice and electrical layout definition (PLC software programming specification is not considered).

2.1 Elect-Designer System

Fig. 1 shows the system realized in the Elect-Designer Project. The application, developed by Microsoft C++, uses ODB-Tools to represent the information regarding the electrical systems. For the data storage a relational DBMS is used: the data transfer procedures between the application and the RDBMS are executed by a software module (Object Wrapper) developed during the project.
Each system component is now described in detail:

- Prelet: the application developed within the Elect-Designer Project. Starting from the mechanical layout of the system, the application leads the designer to make the choice of primary and auxiliary components and subcomponents, automatically providing the list of suitable components for the requirements. Finally, it (automatically) produces the component specification and cash budget. In the next development of the project, the system will be integrated with CAD application to produce a global layout;

- Object Wrapper: software module for storing and retrieving objects in a RDBMS supporting a OO interface. Having these capabilities in a separate component, it helps to isolate data management system dependencies and hence contributes to portable applications [8].
- ODB-Tools: thanks to the interaction with ODB-Tools (described in Section 4), the object data schema is validated by the coherence and variance check.

Since the focus of the paper is on the usage of DLs as a support for engineering applications, next sections of the article will describe the reasoning techniques on the data schema and their implications, leaving out the project's applications (Prelet and Object Wrapper software modules).

3 OLCD: A Formalism for Complex Objects including Integrity Constraints

OLCD is an extension of the *object description language* ODL, introduced in [3, 9] and is in the tradition of complex object data models [10]. OLCD, as its ancestor ODL, provides a *system of base types*: string, boolean, integer, real; the type constructors *tuple*, *set* and *class* allow the construction of complex value types and class types. Class types (also briefly called *classes*) denote sets of *objects with an identity and a value*, while *value types* denote sets of *complex, finitely nested values without object identity*. Additionally, an intersection operator can be used to create intersections of previously introduced types allowing simple and multiple inheritance. Finally, types can be given names. Named types come in two flavours: a named type may be *primitive* that means the user has to specify an *element*'s membership in the interpretation of the name or *virtual* and in such a case its interpretation is computed.

The extensions to ODL introduced in OLCD are: *quantified path types*, *integrity constraint rules* and *methods* definition. The first extension has been introduced to deal easily and powerfully with nested structures. Paths, which are essentially sequences of attributes, represent the central ingredient of O-O query languages to navigate through the aggregation hierarchies of classes and types of a schema [11]. In particular, following [11], we provide *quantified* paths to navigate through set types. The allowed quantifications are existential and universal and they can appear more than once in the same path. A path type is a type associating with a path to a type of the formalism. Therefore, by means of path types, we can express a class of integrity constraints.

The second extension allows the expression of integrity constraints represented as a *if then rule* universally quantified over the elements of the domain with an antecedent and a consequent which are types of the formalism. These rules give the possibility to represent a relevant piece of knowledge in a declarative style.

The third extension permits the definition of methods as used in most of the existing object-oriented data model [12]. Each method is related to a class and can be inherited along the classes hierarchy. Method names can be reused in different parts of the hierarchy (overloading).

A *generalized database schema* definition can be thus introduced since it perfectly fits the usual database viewpoint.

3.1 Semantic expansion of a type and coherence check

The semantic expansion of a type allows to incorporate any possible restriction which is not present in the original type but is *logically implied* by the type and by the schema. We propose a method to compute the semantic expansion of a type [4] which is based on two ingredients: iteration of the transformation "if a type implies the antecedent of a rule then the consequent of that rule can be added"; evaluation of logical implications by means of *subsumption computation* [2, 3] among types. Following the approach of [13] for semantic query optimization and exploiting at the same time subsumption computation to evaluate logical implications, we perform the semantic expansion of the types included at each nesting level in the type description.

Moreover, we say that a type is *incoherent iff* for all domains the (*semantic expansion* of the) type extension is always empty. A database schema is *incoherent iff* for all domains has at least an *incoherent* type. In [3], the algorithms to compute the *subsumption relationship* and *coherence check* of a

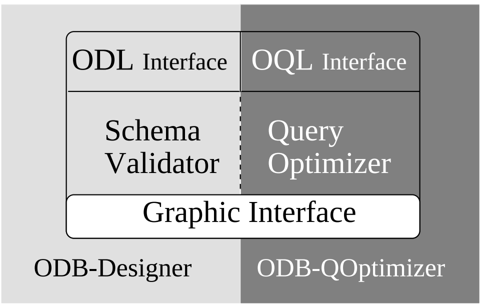

Figure 2: ODB-Tools

schema are given.

Finally, we consider the method consistency problem and, in particular, we want to check whether a given method can produce an inconsistency (i.e. when a method is called with arguments for which this method is undefined). Following the approach used in O_2 data model [12], we define the *covariant* specialization for a method. This means that for a method specification, the sufficient condition to ensure a safe type is that each type (i.e. signature field of the method) can be specialized only by terms whose types are "subtypes" of the fields type (we adopt the notion of "subtypes" defined in [14]). As described in [15], the covariant specialization guarantees the statically well-type method but is no safe with respect to run-time type error. To avoid runtime error it is necessary to use the *contraviariant rule* [16] for the method signature parameters, so that the parameter types of the ancestor are subtypes of the inherited method types. In the OLCD extension we have defined both a covariant and contravariant safe schema, so that it is possible to require, alternatevely, both coherence checks, since covariance and contravariance are not opposing views, but distinct concepts with their own place in object oriented systems. The independence of the two mechanisms, shown in [15], should be integrated in a type-safe checking of the object oriented language.

4 ODB-Tools Architecture

ODB-Tools, whose architecture is shown in Fig. 2, provide a *user-friendly* integrated environment based on the ODMG-93 standard, with the following features:

Schema validation and classification: The user inserts a DB schema, exploiting ODL language, and the system performs the coherence validation and the classification, i.e., for each class, the system determines the right place of the class in the inheritance hierarchy between its most specific generalizations and its most generalized specializations.

Regarding the coherence check w.r.t. methods, the tool performs the covariance check on return parameter (for the method defined in the integrity constraint), while the covariance check on the call parameter types will be implemented in the future. The result is shown by a graphic representation of the schema inheritance and aggregation hierarchies.

Semantic Query Optimization: the user can insert a query, exploiting OQL language, related to the given schema and the system executes the semantic query optimization. The result is the semantic expansion of the query shown by a new OQL description and by a graphic representation of the query's classification with respect to the schema.

ODB-Tools is composed of five main modules:

- **ODL Interface**: the schema acquisition module that accepts a schema description in ODL language and translates it into a OLCD schema. ODL syntax has been extended to provide the IC Rules descriptive capability.
- **OQL Interface**: the query input (output) interface module that receives a query in OQL language and translates it into OLCD syntax (and vice-versa).
- **Schema Validator**: the (covariance) schema validation module which automatically builds up the class taxonomy and preserves the coherence with respect to the inheritance and aggregation hierarchies.
- **Query Optimizer**: the module which executes the semantic expansion of the query.
- **Graphic Interface**: the module which visualizes the schema inheritance and the aggregation hierarchies.

ODB-Tools is available on Internet at http://sparc20.dsi.unimo.it. The interfaces, validation and optimization modules are realized using the C language (gcc 2.7.2 compiler, flex 2.5 and bison 1.24 generator), while the graphic module is developed by Java language (JDK 1.1 compiler).

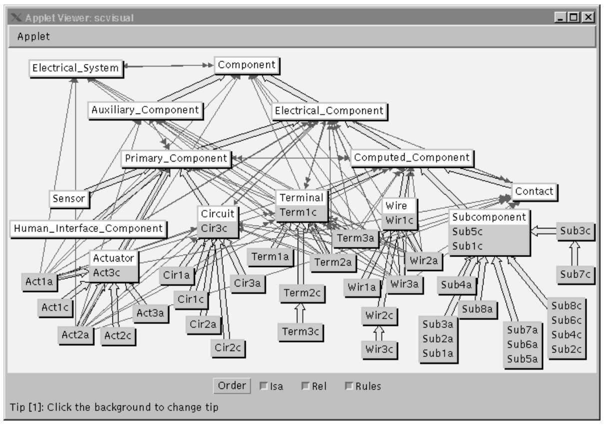

Figure 3: The Elect-Designer Schema

5 The DB schema of Elect-Designer: coherence check

This section will illustrate the application data schema created by exploiting the reasoning techniques for the design.

The DB schema designed for the Elect-Designer Project is shown in fig. 3, where a light box represents a class while a grey one is an Integrity Constraint (IC), divided in antecedent (*if A*) and consequent (*then B*). The use of ODB-Tools during the design ensures the avoiding of coherence problems in classes, ICs and methods definition. It is important to underline that in the Elect-Designer Project, caused by the nature of the problem, ICs and methods constitute more than half of the schema information knowledge: almost all the information are electrical quantities whose relationships are known a-priori and fixed by the UE document DIN VDE 0660, N. 102,107,200. Thanks to the ICs and methods declaration we may insert at the conceptual level (directly into the schema) the consistence rules belonging to the electrical normative. The subsequent covariant-schema check permits the automatic verification of classes and methods definition. This feature turns out to be very useful in the reduction of the project's time development and in obtaining the schema correctness. In such a way designers can focus their attention on both the semantic aspects of the schema and the IC rules, without necessarily stopping for the manual coherence check after each minimal schema modification.

At the end of the design, some further remarks on the obtained schema can be carried out :

1. ODB-Tools is able to show directly classes (and ICs) relationships (even the inherited ones) and gives the chance to make a deep analysis of the implications among classes and ICs (for example rule Act1A is related to the classes Electrical_System, Contact and Terminal);
2. equivalent classes are promptly detected (and shown in the same box). In the example the Sub5C is equivalent to Subcomponent class, so that the consequent of Sott5 IC is redundant and replaceable by the class itself;
3. an inheritance hierarchy regarding the ICs may be detected (by subsumption computation), so that the designer has the opportunity to enrich and hence modify the knowledge base. In the example, Term3 rule is a specialization of Term2 rule.

6 Conclusions

Starting from ODB-Tools, a DL-based system used in the design of engineering application to aid the developer in the consistency check activity, we have extended OLCD including methods definition and check analysis.

The effectiveness of the approach for electrical system design is shown by means of a real application modelling example. Therefore, the usage of this inferential instrument is an element of the schema's automatic check which turns out to be a relevant aid for the critical analysis of the schema itself.

References

[1] R. G. G. Cattel. *The Object Database Standard - ODGM93.* Morgan Kaufmann, 1996.

[2] R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[3] S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Journal of Applied Intelligence*, 4:185–203, 1994.

[4] Domenico Beneventano and Sonia Bergamaschi. Incoherence and subsumption for recursive views and queries in object-oriented data models. *Data and Knowledge Engineering*, 21(3):217–252, February 1997.

[5] Domenico Beneventano, Sonia Bergamaschi, Claudio Sartori, and Maurizio Vincini. ODB-QOPTIMIZER: A tool for semantic query optimization in oodb. In *Int. Conference on Data Engineering - ICDE97*, 1997. http://sparc20.dsi.unimo.it.

[6] D. Beneventano, S. Bergamaschi, C. Sartori, and M. Vincini. ODL-tools: a description logics based tool for schema validation and semantic query optimization in object oriented databases. In *Sesto Convegno AIIA - Roma*, 1997.

[7] S. Riccio. Elet-designer: uno strumento intelligente orientato agli oggetti per la progettazione di impianti elettrici industriali. Tesi di Laurea, DSI, Università di Modena, 1998.

[8] S. Bergamaschi, A. Garuti, C. Sartori, and A. Venuta. The object wrapper: an object-oriented interface for relational databases. In *Euromicro - 97*, 1997.

[9] D. Beneventano, S. Bergamaschi, S. Lodi, and C. Sartori. Consistency checking in complex object database schemata with integrity constraints. *IEEE Transactions on Knowledge and Data Engineering*, 10:576–598, July/August 1998.

[10] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *SIGMOD*, pages 159–173. ACM Press, 1989.

[11] E. Bertino, M. Negri, G. Pelagatti, and L. Sbattella. Object-oriented query languages: The notion and the issues. *IEEE Trans. Knowl. and Data Engineering*, 4(3):223–236, June 1992.

[12] François Bancilhon, Claude Delobel, and Paris Kanellakis (eds.). *Implementing an Object-Oriented database system: The story of* O_2. Morgan Kaufmann, 1992.

[13] M. Siegel, E. Sciore, and S. Salveter. A method for automatic rule derivation to support semantic query optimization. *ACM Trans. on Database Systems*, 17(4):563–600, December 1992.

[14] Luca Cardelli. A semantics of multiple inheritance. *Information and Computation*, 76(2/3):138–164, February/March 1988.

[15] Giuseppe Castagna. Covariance and contravariance: Conflict without a cause. *ACM Transactions on Programming Languages and Systems*, 17(3):431–447, May 1995.

[16] Pierre America and Frank van der Linden. A parallel object-oriented language with inheritance and subtyping. *ACM SIGPLAN Notices*, 25(10):161–168, October 1990. *OOPSLA ECOOP '90 Proceedings*, N. Meyrowitz (editor).