Accessing Configuration-Databases by means of Description Logics

T. Kessel and **M. Schlick** and **O. Stern**
ERIC – ENSAIS, Strasbourg, France
email: ⟨kessel, schlick, stern⟩@steinway.u-strasbg.fr

1 Introduction

Description Logics (DL) has become one of the most interesting formalisms in the Knowledge Representation field [Woods and Schmolze, 1992]. There exists now a wide range of implemented systems (for example C3L, CLASSIC, KRIS, LOOM, BACK), a well studied theory with respect to expressive power (for instance the representation of time or uncertainty), inferential services and enhancements based on other paradigms (rules, constraints).

Until now, less attention has been paid to an integration of Knowledge Representation and *Data Base Management Systems* (DBMS) technology. The research in the first area was focused on the interpretation of the semantic links between data which transform them into knowledge, whereas the latter was in charge of handling large amount of data. In our point of view, both fields can be considered as complementary.

The interfacing between DBMS and Knowledge Representation systems may cover two topics:

1. The interface is exclusively focused on the exchange and transformation of data. For instance, information about a mechanical piece is retrieved from a CAD data base system and converted into the internal format of the DL system.
2. The DL system constitues a kind of application layer that is built on top of a DBMS. This top layer provides a maximum of inferential services and expressive power. Furthermore it allows the conception, optimisation or distribution of queries that are mapped to the underlying DBMS.

We propose to elaborate the second scenario because it combines the best of both worlds. Thanks to the embedded DL system, the application layer may use domain knowledge to optimise queries or to send it to one of the distributed DBMS, whereas the DBMS itself handles the storage, retrieval and recovery of data. For instance, our major application will be the configuration of modular bus systems from various, available modules for the electronic bus system of a vehicle, as described in [Keith *et al.*, 1995]. This research project requires a semantic integration of heterogeneous knowledge sources. They denote for instance, constraints which specify possible combinations of bus modules, rules which guide the configuration process by means of heuristics, concepts which describe the functional composition of electromechanical components within a motor and individuals that represent variants of the bus modules. The DL system we would like to employ in this context is the C3L system [Kessel *et al.*, 1995] which is implemented within a frame-based structure and smoothly integrated in an object-oriented programming and development environment.

At the moment we have no experience in the specific research field of combining DL and DBMS yet, although it seems to be very promising with respect to the above mentioned research project . The inferential services of a DL system, in particular the classification of concepts and the realization of individuals, are indispensable to structure such a large model as the description of a modular bus system in a vehicle. Open world reasoning seems appropriate to us, because we have to deal with incomplete knowledge which may evolve continually during the configuration process. Furthermore, an important feature of DL is the deduction of implicit information by means of propagation of recently added knowledge in the ABox. The implementation of a multilayered typology of user-programmed inferences, attached to concepts, which reason about individuals, like in classical knowledge-based system shells, for instance KAPPA or Nexpert Object, would be helpful to include procedural knowledge. A substantial contribution of DL comes from the power of its intelligent retrieval queries that allow, first to normalize completely different retrieval descriptions (which need not necessairly make use of the inheritance information), and second the generation of abstract concepts which are deduced from role resp. attribute values.

What may be the impact of such a project on our research about DL systems ? We suggest to focus on the study of retrieval functionalities provided by DL systems. This issue concerns essentially all kinds of ABox services, for instance the retrieval or realization of instances. Within this framework it is worth to examine the ABox's performance with respect to its architecture in order to reduce the average retrieval costs. What are the possible drawbacks of our approach ? The coupling between the DL system and the DBMS may induce some performance losses, due to the intensive communication exchange between two different systems. Another aspect concerns the mapping of the complex concept or individual structures to a relational or object-oriented database scheme which has been undertaken yet.

The rest of the paper is structured as follows. A brief C3L system description is presented in the second section. The following chapter tackles the integration of procedural knowledge in a DL system, notably C3L. How to integrate an object-oriented DBMS within such a system, exemplified at C3L, is discussed in the fourth section. Afterwards the significance of the retrieval inference and related topics are studied. Applying a DL system for configuration purposes constitutes the principal issue of the sixth chapter. Last, but not least, we conclude our work.

2 C3L - a system description

At the moment, two different C3L versions exist: one academic research prototype, built in Common Lisp, and another version including an object-oriented data base management system and written in C++, but which is at the current state limited to the TBox. The later system is called C3L++ (for obvious reasons) and will serve as the implementation base for future development and enhancements. The porting of C3L from Common Lisp to C++ is motivated by the idea to scale small knowledge bases up to large ones, which require more sophisticated means for handling huge amounts of knowledge.

The C3L system may be characterized in some terms as follows:

- it is a descendant of the description logics (or KL-ONE) family
- it provides reasonable expressive power (e.g. conjunction, all, one-of, fills, at-least, at-most)
- it incorporates useful inferential services (e.g. subsumption, classification, recognition)
- it is implemented in a frame-based based system
- it contains declarative as well as procedural knowledge
- it is concieved in the perspective to serve as a knowledge representation module for a hybrid development environment (the term IKME, intelligent knowledge management environment, describes it best)

Special features of C3L which distinguish it from other description logics systems are:

- a reflexive object-oriented, frame-based architecture
- the integration of methods

The initial design philosophy of C3L was to combine ideas coming from the communities of frame languages and description logics [Carré *et al.*, 1995]. Having (partially) achieved this goal, we realized that it seems necessary to include the database community as well, in order to be able to handle large quantities of data without significant performance losses and keeping powerful reasoning mechanisms. Anyway, the description logics system C3L can be considered as a kind of application layer which hides the underlying database system and allows a completely transparent management of data for the user.

Two inhouse research projects constitute the testbed for the C3L system. The first is the domain modeling of a configuration system for a modular electronic bus system in vehicles, whereas the second addresses the representation of features in CAD and their links to technological information which are needed e.g. for production purposes. Both projects are rather small-scale research projects and still in progress, but they already provided us with very helpful feedback.

After having tested our system in the above mentioned two application scenarios, we identified two major requirements for the improved successor C3L++ of the academic research prototype C3L:

- high performance: the currently used data structures are not optimized for high performance, because it is an exploratory implementation
- large scale knowledge bases: due to memory restrictions the number of defined concepts, roles and individuals has been quite restricted so far

C3L enabled us to acquire a lot of valuable experience about description logics system design and building such architectures for information systems. A complete redesign of the C3L system is now in progress, a port of the TBox to C++ is the very first result of this effort. We think of testing it by means of a huge random knowledge base which will be automatically generated. Such an approach might allow to obtain reliable, empirical data of the systems performance and behaviour. A pre-version of C3L++ has showed a considerable increase of performance which will be studied in more detail in the near future.

3 Integrating procedural knowledge

Motivated by the requirements of studied applications and the need for an efficient manipulation of knowledge, we are actually working on the integration of C3L in an object-oriented programming environment. Obviously it is not satisfactory to provide simply methods, but to offer a formalism which enables the experienced user on the one hand to fulfill his particular needs and on the other hand to control the side effects of the methods. The objective is to support a multi-layered typology of methods whose consequences can be easily supervised and tested.

Suggestions for formalizing the notion of methods come from the area of specification languages for the development of knowledge-based systems in the field of knowledge engineering [Fensel and Harmelen, 1994]. The SCARP system [Willamowski, 1994], set on top of SHIRKA, influences as well our decision to include methods in form of *tasks*. Anyway the employed approach cannot deny the impact of classical ideas coming from the field of hierarchical and sceletal planning.

Tasks are on the one hand sufficiently complex, declarative means to abstract from simple methods and on the other hand they are close to the implementation level by incorporating source code of the underlying programming language or by calling other subtasks. A task is defined by numerous properties or attributes which allow better validation and coherence tests of the task.

At the moment, three complementary categories of procedural attachments which are listed below are in work:

- methods: they represent general purpose attachments and are linked to concepts
- demons: they are triggered by value changes of roles, for instance if-added, if-deleted, if-changed, if-needed, are classical demons
- events: they survey the concepts instances and are launched if one instance fulfills the events conditional part

One major problem of procedural attachments are the consequences on the recognition and retraction inferences provided by the ABox. One possible solution is to execute demons first, list the concerned objects and pass them to the recognition service afterwards, in order to avoid costly recomputation of the individuals status during the chained manipulation of data. A more interactive approach may suppose that the user has to demand explicitly the (re-)recognition of an individual which was modified in the past.

Summarizing the benefits of embedding the underlying programming language in a description logics system may result in the following advantages:

- increased flexibility and maintenance of the complete system
- high performance for complex (mathematical) computations
- use of the programming languages high expressive power and performance

Offering an almost exhaustive library of primitive tasks (or methods) is the consequent next step of the evolution of our development environment. In most cases the user may only select the appropriate existing task or use pre-defined tasks which have to be instantiated. The programming effort would be reduced to calling tasks or profiting of the hierarchical task structure. Basic tasks are elementary retrieval or manipulation methods which constitute the principal task layer and which can be completed step by step.

One important feature we are working on is the automatic classification of tasks with respect to certain properties, for instance parameters or agents. The principal underlying idea is to map a task scheme to a normalized concept and to call the usual classification service. Afterwards you may reason about tasks like about normal individuals. This may be particular useful to support programmers who look for special properties of an incompletely specified task.

4 Integrating an object-oriented DBMS

The current C3L++ system which is ported to C++ is particulary optimized for performance issues and synthesizes the design experience we obtained in building the former Common Lisp version. One major system requirement is to get an almost platform independent description logics system.

After having evaluated the possibility to concieve a special DBMS interface for C3L++ within the systems architecture, which provides its own storage and caching strategies, we opted for a more commercial solution by building C3L++ on top of the POET [POE, 1995] database. POET is in fact a pre-compiler which generates (documented) C++ code and it takes the complete memory handling in charge. Therefore the system programmer can concieve the system as whether he had a large, but finite (virtual) memory space, persistent objects only have to be marked in their class definitions.

The chosen solution implies several benefits:

- the object-oriented DBMS system does what it can the best, e.g. memory caching strategies, even if they are not optimized for a description logics system and its specific inferences and data structures
- using such a DBMS simplifies the design, maintenance and documentation of source code with respect for data storage, enabling the system designer to focus on the essential system properties
- the solution is easy to implement and fast thanks to the employment of available classes and methods; we hope also to obtain a real gain of programming efficiency

The principal disadvantage of the solution is that there does not exist anymore a clear distinction between source code of the POET DBMS and of the description logics system. Both parts become very closely intertwined and inseparable.

Nevertheless we are convinced that the above mentioned solution lets us enough space for improving and adapting the C3L++ system for specific applications. Another aspect that was not discussed so far is the mapping from queries of the retrieval language to the underlying DBMS. All topics concerning this issue are studied in the appropriate retrieval section.

5 Significance of retrieval

In industrial applications databases are more often queried than updated. To fulfill this condition, efficient retrieval mechanisms have to be provided. In this context SQL-like query languages are often too difficult to learn for the average employee. As is known from different investigations, three out of four query attempts are non-successful, resulting in an immense loss of time and money. A possible solution to this problem is the usage of DL as a front-end to the database system. Here the user can communicate with the knowledge base by means of simple operators and intuitively understandable object descriptions. The DL system can then optimize the queries and transform them into terms of the underlying database language. This step is completely transparent for the employee, resulting in an increasing acceptance of database applications.

To meet all requirements of database users, we have to provide two different classes of query operators. First, it must be possible to access the descriptions of database objects, for example to receive information on a specific mechanical component. Second, we need dedicated retrieval facilities to find objects by means of arbitrary descriptions. For example an engineer could be interested in finding all bus components transferring high data rates on a specific bus segment. In these cases an intuitive description of such a component is easily constructed, in comparison to the joining of several relations in a relational database system employing SQL.

In C3L the first class is represented by the operators **showall**, **show** and **ask**. Each of them occurs in three different contexts: for roles, concepts and individuals. A **showall**-operation returns a list of the elements of the specified type which are known in the database. The **show**-operator provides the most important properties of the object in question, for example the dependencies from other objects or the values of all its attributes. The **ask**-operator finally allows to specify the properties and attributes of interest for an object, for example if we are curious to know the data transfer rates of a special bus component. All these operations can be easily mapped to database queries without the need for dedicated reasoning mechanisms. In contrast, the operator for the second class, **search**, uses the inferential capabilities of the DL system. It is tightly connected to the retrieval inferences for roles, concepts and individuals. This operator takes an arbitrary object description and returns all database elements matching it.

The retrieval algorithms for roles and concepts can be constructed from the basic TBox inferences for subsumption and classification. The retrieval of ABox individuals is far more complex. In the following we will therefore concentrate on this mechanism. The processing of a query involves five steps [Stern, 1995]:

1. Analysis of the query.
2. Optimization of the query.
3. Choice of the appropriate resolution strategy.
4. Execution of a number of retrieval primitives following the chosen strategy.
5. Verification of the results.

Step four involves real database access, for example by means of SQL. But this is absolutely transparent for the user, the DL system is in charge of the whole transformation process.

The optimization phase first detects inconsistencies in the query. It then tries to simplify the description to accelerate the further processing. Numerical intervals, for example, are normalized and subsuming roles are eliminated.

The choice of the resolution strategy depends on a rather large number of properties of the query. If we are confronted with distributed databases, for example, we have to decide which ones are relevant for the query and when and how to access these knowledge sources. The strategy is also different for various formats of the query. Short ones are processed in another way then queries with lots of roles and attributes, and the treatment is distinct for queries comprising a conceptual description or lacking this. Furthermore, various types of conceptual parts are processed differently. By means of such distinctions we are able to reuse a maximum of already derived facts (recognition inference) to guarantee high performance of the retrieval inference.

To gain a maximum of speed there exists a huge number of retrieval primitives, implemented as independent methods. They can be freely combined or used as single mini-inferences. We have also provided primitives for the most frequent combinations of these basic methods. There are methods to access the different precomputed facts of the ABox and TBox, for example by evaluating the semantic indexing structure or the information inside the role hierarchy. We can classify the conceptual parts of a query or even generate appropriate concept descriptions from role lists if the given description seems not clear enough. And, what is self-evident, there are primitives to access the database interface of C3L which performs the transformation of basic queries into the database language.

As a result of step four we receive a set of individuals that possibly match the query. Due to limitations in the optimization and calculation steps, mainly to minimize the number of real database accesses, this set may contain elements that do not exactly match all role restrictions of the query. This makes a final verification step necessary. There we match the candidate instances with the possibly offended role restrictions by means of a dedicated subsumption algorithm for individuals. The verified objects are finally returned to the user.

The entire retrieval algorithm is correct and nearly complete. It is even more complete than the recognition inference of C3L. This could be achieved by deducing further implicit facts during step four of the algorithm. In all test cases so far, the retrieval inference was capable of calculating all the instances matching a query. Hopefully, there will be only very few cases where some implicit dependencies can not be detected.

The coupling between C3L and the database system can be described by employing a meta-model. This model comprises all the different aspects of the integration of a DL system with a database, like the construction of queries in terms of the database language, the distribution of queries in distributed environments, and the access methods of the database interface of C3L. By means of this model the interface to the database management system can be easily adopted to any commercial product. We can entirely avoid changements to the inference mechanisms. Only the mapping of basic queries of C3L to the database language has to be modified. In the future we will also try to make use of already existing databases. To perform this difficult task, it will be necessary to extract generic concept descriptions and individual definitions from the database contents to use them for the queries. An automatic transformation seems, at the current state of our research, rather difficult if not impossible. But even an extraction by hand could be worth the trouble, compared to the benefits of using DL as a query component.

When we have a closer look on the meta-model, we can distinguish the different tasks of the DL system and the DBMS in our application scenario. The database system is only concerned with the storage of large amounts of data, whereas C3L is in charge of all problems involving some reasoning:

- Construction and verification of queries.
- Detection of inconsistencies.
- Optimization and distribution of queries.
- Generalization of queries.

The last point in this list is worth some more explanations. In the analysis step of the retrieval inference we can detect sub-queries that occur very often. A considerable speed-up for such queries can now be achieved be generalizing the sub-query, resolving it and caching the results. In subsequent queries these parts have not to be processed, it is sufficient to use the stored answers.

Concerned with industrial applications, for example in the domain of the configuration of bus systems for vehicles, we have learned that it is not sufficient to provide only system defined retrieval capabilities. Most applications show a need for dedicated facilities specially adopted to the domain in question. To meet this requirement, C3L can be extended by procedural knowledge. Within a syntactically and semantically regulated framework, the user can add retrieval inferences implemented in the host language. For this purpose, most of the retrieval primitives and optimization methods are accessible through a programmers interface. They can now be used to implement domain-specific retrieval functions and strategies. A little drawback of this approach is that these user programmed methods can lead to inconsistencies in the user defined query answering process. But the necessity of a careful implementation style seems to be a little inconvenience compared to the possibility to adopt the system to the special requirements of a real world application. Furthermore, this is a feature heavily missing in database-only systems that use, for example, SQL.

As already mentioned, the actual coupling of C3L with a database system is performed by means of a dedicated interface. The only purpose of this module is to perform the transformation between basic DL queries and queries in terms of the database language. We could identify a small number of such basic queries that are sufficient to provide C3L with all necessary information from the underlying database. A realization of this interface exists for relational databases that use SQL as their query language. Actual work is in progress for the integration of the object oriented database system POET. This OODBMS will function as the back-end data store in the C++ version of C3L. In our experience so far, the presented approach is well suited for relational DBMS as well as OODBMS.

6 Configuration as an application

Configuration can be defined as the design of a technical system, according to a specification, by choosing and assembling different modules taken from a module catalogue. If this is done by hand, especially for more complex problems, it often results in errors like inconsistency or missing parts. Therefore the aim is to develop a system to support the configuration process or to do configuration automatically. We propose a system based on Description Logics.

The problem of solving the configuration task by means of DL was already studied by the AT&T research group in the framework of the PROSE project [Wright *et al.*, 1993]. We want to focus on the advantages that Description Logics offers for the treatment of large amounts of data needed for the configuration process.

Databases are necessary to store the large module catalogues. It is important for the economic success of a configuration system that module descriptions of newly developed modules can be integrated in the DBMS as fast as possible. Using a normal DBMS, a domain specialist and a DBMS specialist are only together capable to formalize the information about a module and to add the resulting description to the DBMS. This results in a loss of time and money. Furthermore, consistency checking between the module descriptions is indispensable for the configuration process. Ordinary databases are not capable of performing this task.

The use of a Description Logics system as an application layer that is built on top of the DBMS can solve these problems. Domain and knowledge engineering experts have to work together to build the terminological part of the knowledge. In this part, the domain vocabulary and principles are described. Once done, this part only has to be changed if the description is no longer sufficient. In contrast to rare updates of the terminological knowledge the assertional knowledge has to be modified rather frequently, because all descriptions of the new modules are integrated as individuals. Because of the easier access methods of DL and the possibility to use the domain vocabulary, we expect that this could be done directly by the domain expert. This would make it possible to integrate the updating of the DB in the module development process. Additionally, the DL system automatically guarantees a maximum of consistency of the knowledge base.

A DL system does not only improve the management of a knowledge base. As described above, the use of retrieval and classification offers possibilities to accelerate the access to the different facts. For example a common problem during the configuration process is, to find a module which combines the properties of two or more other different modules. We could for example be interested in finding an integrated automobile motor management unit which integrates the ignition and injection management. The problem is to retrieve a module description that fits to a list of various properties. With a normal DBMS such a search would be very expensive. With a DL system, the query strategy can be individually optimized which results in a higher performance and a better acceptance of the configuration system.

The use of a DL system also poses one problem: Is it possible to change between an open and a closed world assumption? The open world is convenient for the knowledge acquisition step, to enable the user to integrate new facts easily into the knowledge base. During the configuration process, a closed world assumption seems to be more adequate. If, for example, the configuration system excludes the first of two possible modules it can choose the second. An open world assumption would not allow this conclusion.

We have started to model the communication flow among different components linked to the electronical bus system of a vehicle. Modeling the domain using an object-oriented approach, like Description Logics, is more appropriate than conceptual modeling for DBMS. Domain experts have less problems to intuitively understand the resulting models.

7 Conclusion

In this paper we tried to motivate the benefits of coupling a DBMS system with a knowledge representation system, in particular the description logics system C3L++. The most important requirements for such a synergetic combination are:

- the need for large scale knoledge bases
- the potential performance gains

The implementation of C3L++ which incorporate such features is still in progress. Starting with a brief presentation of the academic research prototype C3L, we familiarize the reader with its idiosyncrasies, for instance the integration of procedural knowledge by means of methods, demons and events. Some decision criteria, for choosing an object-oriented DBMS (in our case: POET) and the reasons for setting C3L++ on top of it, are studied in the following section by emphasizing the system development aspect. Discussing the impact of retrieval for our configuration application and its consequences for the DBMS coupling form the major topics of the successive part. Finally, some problems posed by applying description logics to the configuration of electronic bus systems are elaborated.

8 Acknowledgments

We would like to thank the anonymous referees for their valuable hints on earlier versions of this paper, and our colleagues for very fruitful discussions on the subject. In particular we owe a lot to the Robert Bosch company, Germany, with who we cooperate in the research project on configuration of modular bus systems for vehicles.

References

[Carré *et al.*, 1995] B. Carré, R. Ducournau, et al. Classification et objets: programmation ou représentation. In *PRC-GDR Intelligence Artificielle*. TEKNEA, 1995. In French.

[Fensel and Harmelen, 1994] D. Fensel and F. Harmelen. A comparison of languages which operationalize and formalize KADS models of expertise. *The Knowledge Engineering Review*, 9(2):105–146, 1994.

[Keith *et al.*, 1995] B. Keith, T. Kessel, M. Schlick, and O. Stern. A description logics based approach to the configuration of diagnostic systems. In *Proceedings of the IAR conference on Automatic Control and Signal Processing*, 1995. Forthcoming.

[Kessel *et al.*, 1995] T. Kessel, F. Rousselot, and O. Stern. From frames to concept: Building a concept language within a frame-based system. In *Proceedings of the International Description Logics Workshop at Rome*, 1995.

[POE, 1995] POET Software Corporation, San Mateo. *POET Release 3.0*, 1995.

[Stern, 1995] O. Stern. Entwicklung der assertionalen Komponente ERICA für das terminologische Wissensrepräsentationssystem C3L. Master's thesis, Universität Karlsruhe (TH), 1995. In German.

[Willamowski, 1994] J. Willamowski. *Modélisation de tâches pour la résolution de problèmes en coopération système-utilisateur*. PhD thesis, université Joseph Fourier - Grenoble 1, 1994. In French.

[Woods and Schmolze, 1992] W. Woods and J. Schmolze. The kl-one family. In *Semantic Networks in Artificial Intelligence*, pages 133–177. Pergamon Press, 1992.

[Wright *et al.*, 1993] J. Wright, E. Weixelbaum, et al. A knowledge-based configurator that supports sales, engineering and manufacturing at AT&T Network Systems. In *Proceedings of the Innovative Applications of Artificial Intelligence Conference*, 1993.