Joint NCC & IIS Bull., Comp. Science, 12 (1999), 29–31

Database processing in constraint programming paradigm based on subdefinite models

Sergey Lipski, Vladimir Sidorov, Vitaly Telerman, Dmitry Ushakov

A technology for solving problems with large data sets in an object-oriented constraint programming environment *NeMo+* is presented. We describe facilities for database access and interaction of data with *NeMo+* objects.

1. Introduction

The paradigm of constraint programming is becoming a popular tool for creation of systems supporting the solution of complex problems with large sets of weakly structured data. The idea is to use a partially defined description of an object rather than an exactly defined object. In this case, the problem is described as a collection of variables and constraints. The process of solutions consists in finding some values of the variables which satisfy the given set of constraints.

There is no distinction between a program and data in constraint programming. Consequently, programs for large applications that require considerable storage space are also becoming too large to allow efficient processing in the computer's main store. Therefore, the program needs to interact with a database. Moreover, data persistence provided by databases is needed when the program is frequently used with different values of parameters or when the intermediate results should be saved for further use.

In this paper we consider a problem of merging of the object-oriented constraint programming system *NeMo+* with database processing. *NeMo+* solves constraint satisfaction problems using the apparatus of subdefinite models. The theory of subdefinite models was proposed by A. Narin'yani in the early 80s [1]. The connection with relational databases is implemented in *NeMo+* via the ODBC interface.

2. Brief overview

There are different approaches to the organization of interaction of CLP programs with databases. One of the most promising paradigms is that of constraint databases [2]. A key idea of constraint databases is to consider data as constraints and constraints as data. Information is stored in constraint databases in the form of the so-called generalized tuples. A generalized tuple is the conjunction of a set of constraints of the form $A*C$, where A is an attribute of a relation, C is a constant, and $* \in \{=, >, \geq\}$. In other words, a generalized tuple is a formula of the CLP language. This results in a homogeneous structure of the program data and the database. Due to highly abstract representation of data, constraint databases have greater expressive power as compared to other data models, and potentially can be applied to arbitrary domains. However, many issues related to efficiency of constraint databases, such as transactions, optimization, and indexing remain unresolved, and so the technology is not yet available for commercial applications.

Another approach is to use the data of the problem in traditional relational databases. It is this approach which is most frequently used in existing industrial systems of application development, the best known of which is the CHIP system [3]. This approach is called loose coupling to an external database, in contrast to the tight connection which exists between a constraint logic programming system and a database in the case of constraint databases. This approach also has some disadvantages. Since repeated database accesses are required throughout program execution, it may incur a runtime

overhead. The second disadvantage is that constraints in general cannot be stored in the database in a form that reflects their meaning. Thus, after retrieval from the database, the data must be converted from the database representation into a form which can be used by a constraint logic programming system. Nevertheless, this approach has been used to create various complex industrial applications [4].

3. The *NeMo+* Programming Environment

The *NeMo+* system is an object-oriented constraint programming environment based on the subdefinite model approach [5]. It has a rich set of predefined data types and constraints, as well as a universal (i.e. applicable to any data types) constraint propagation algorithm. The user may define his own data types and constraints over the predefined ones. A new data type is described as a class of objects whose components (or *slots*) are linked by a certain set of constraints. The ability to specify problems in terms of their subject domains increases significantly the efficiency of application of the *NeMo+* technology. In fact, the knowledge representation language of *NeMo+* represents a combination of the object-oriented approach and the paradigm of constraint programming. The high-level specification of the problem is transformed by the compiler into a network containing the predefined data types and constraints. The library of predefined data types and constraints is implemented as a system of C++ classes.

In the *NeMo+* language, both the internal semantics of objects and all aspects of interaction between them are described by relations and constraints. Only those properties of the object-oriented approach are used which support knowledge structuring, including construction of a hierarchy of notions, inheritance and incapsulation. In addition to ordinary expressions, the language has conditional expressions and alternative expressions.

At present *NeMo+* uses one algorithm of constraint propagation which has two peculiarities. First, it is applied to an object-structured network of constraints. Second, this network may contain variables of different types.

4. Database support in *NeMo+*

To solve real-world problems with a great number of data, it is necessary to provide *NeMo+* with some database support. Database management in *NeMo+* is carried out through the ODBC interface. Thus, *NeMo+* can use any external data sources. A database access in a *NeMo+* program is translated into calls to ODBC API functions. To support the database access, *NeMo+* implements an instrumental generic class `DBR` which is instantiated by the name of a user-defined class. Objects of the class `DBR(C)` are tables of a relational database. The slots of the parameter class of `DBR` should have the names and types equal to the names and types of some attributes from the database table. The constraint

```
function Connect (DSrceName:string) : bool;
end;
```

serves as connection with a specific relational database named `DSrceName`. The constraint

```
function Open (SQL_query:string) : DBR;
end;
```

links an object of type `DBR` with the table returned as a result of submitting an SQL query to the current database. If the number of columns of the table is greater than the number of slots in the parameter class `DBR`, then the table is projected to the slots. If the slots of a class contain constraints, then only those rows of the table will be selected which satisfy the constraints. Finally, the constraint

```
function Contains (rel:DBR, tuple:any) : bool;
end;
```

is used to write the information from the table `rel` into a generalized tuple `tuple`. In the mathematical notation this is expressed as `tuple` $\in$ `rel`. Since the slots of the object `tuple` carry subdefinite information on their values, this information can be refined as a result of interpreting the constraint according to the algorithm of constraint satisfaction in *NeMo+*. For example, the fragment

```
Connect("Sample");
rel : DBR(Point) with This = Open("select * from Map");
tuple : Point with x = [100,150]; y = [200,250]; end;
Contains( rel, tuple );
```

allows us to retrieve, from the table `rel`, all the points whose coordinates lie within the region $[100, 150] \times [200, 250]$.

The subdefinite values of the slots of a generalized tuple may be changed (refined) while solving the problem. Therefore, these values are checked for each database access, and only those records which conform to them are retrieved. Thus, we are able to reduce significantly the run-time overhead for repeated database access during calculations.

5. Conclusion

The technology of working with *NeMo+* has several applications in various subject domains related to intensive processing of large data sets.

For example, in a time scheduling problem we can store schedules in a database. We have also a set of constraints that should be satisfied by the values from the schedules. The problem can be solved differently, depending on how we organize the interaction between the constraint solver and databases. One of possible approaches is to consider schedules not as a collection of records in the database, but rather as additional constraints specified in a tabular form. This approach assures the semantic integrity of the program data and the database and makes it possible to work with the database by using standard constraint satisfaction algorithms.

Other significant applications of our approach are the areas of geographic information systems and CAD/CAMs. Their peculiarities are a very large number of objects and very complicated computations. *NeMo+* makes it possible to maximally restrain the search region and to carry out the necessary computations automatically.

NeMo+ is implemented in C++ under *Microsoft Windows* and can run on 386-compatible computers. In order to work with databases, it is necessary to install ODBC drivers version 2.0 or later. A number of benchmarks solved by *NeMo+* have been presented in paper [6].

References

[1] A.S. Narin'yani, *Subdefiniteness and Basic Means of Knowledge Representation*, Computers and Artificial Intelligence, Bratislawa, **2**, No 5, 1983, 443–452.

[2] V. Gaede, M. Wallace, *An Informal Introduction to Constraint Database Systems*, Lect. Notes Comput. Sci., **1191**, 1997, 7–52.

[3] P. Kay, H. Simonis, *Building industrial CHIP applications from reusable software components*, Proc. Conf. on Practical Applications of Prolog (PAP'95), 1995.

[4] H. Simonis, *Application Development with the CHIP System*, Lect. Notes Comput. Sci., **1034**, 1995, 1–21.

[5] V. Telerman, D. Ushakov, *Subdefinite Models as a Variety of Constraint Programming*, Proc. of the Intern. Conf. Tools of Artificial Intelligence (ICTAI'96), Toulouse, 1996.

[6] I.Shvetsov, V. Telerman, D. Ushakov, *NeMo+: Object-Oriented Constraint Programming Environment Based on Subdefinite Models*, Lect. Notes Comput. Sci., **1330**, 1997.