CONTEXTS FOR ORGANIZATIONAL INFORMATION SYSTEM DESIGN AND IMPLEMENTATION

Salvador Abreu
Universidade de Évora and CENTRIA
Portugal

Daniel Diaz
Université de Paris I and INRIA
France

Keywords: Information Systems Applications, Modules, Object-oriented Programming, Contextual Logic Programming.

Abstract: In this article we sustain that Contextual Constraint Logic Programming (CxCLP for short) is a useful paradigm in which to specify and implement Organizational Information Systems, particularly when integrated with the ISCO mediator framework. We briefly introduce the language and its underlying paradigm, appraising it from the angle of both of its ancestries: Logic and Object-Oriented Programming. An initial implementation has been developed and is being actively used in a real-world setting – Universidade de vora's Academic Information System. We briefly describe both the prototype implementation and its first large-scale application. We conclude that the risk taken in adopting a developing technology such as the one presented herein for a mission-critical system has paid off, in terms of both development ease and flexibility as well as in maintenance requirements.

1 INTRODUCTION

In the process of devising a strategy for the gradual design and deployment of SIIUE, an organizational information system for Universidade de vora, we were faced with hard choices: as there are really no available ready-to-use solutions, some measure of in-house development was inevitable: Academic organizations have several specificities which are not well catered for by existing ERP products. The adoption of an existing methodology based on existing (commercial) tools was considered, but deemed to be rather inflexible and ultimately expensive in the long run, moreover, it would hardly build on the known-how acquired through the developments which had been locally initiated over the preceding years.

Our explicitly assumed option was to rely on open-source software as much as possible, in order to avoid vendor lock-in and to ensure that future developments could always be effected using in-house competence. In doing so, we had to be very careful in order to pick the most appropriate tool for each particular aspect of the project, always making sure that everything should fit harmoniously.

The overall coordination of the various software components of such a mixed system was critical to the success of the project: this is where Logic Programming appears to be a very interesting and promising choice. Early work on the design and development of SIIUE already indicated that it would benefit from Logic Programming tools playing an increasingly central part in the system. This observation, later confirmed by further developments (Abreu, 2001), led to incrementing the investment in the technology which was at the base of the entire project: Prolog as the foundation of an Organizational Information System specification *and implementation* language.

The benefits of Logic Programming are well known: the rapid prototyping ability and the relative simplicity of program development and maintenance, the declarative reading which facilitates both development and the understanding of existing code, the built-in solution-space search mechanism, the close semantic link with relational databases, just to name a few. The realization that Logic Programming is a promising tool with which to address this type of problem is not exclusive to Universidade de vora's project.

Our choice of GNU Prolog as the basic tool with which to develop our system was due to several factors, not the least of which is its inclusion of a complementary problem-solving paradigm: *constraint programming.* Constraints strengthen the declarative

Abreu S. and Diaz D. (2004).
CONTEXTS FOR ORGANIZATIONAL INFORMATION SYSTEM DESIGN AND IMPLEMENTATION.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 227-232
DOI: 10.5220/0002649202270232
Copyright © SciTePress

programming facet of Prolog, providing an a-priori search-space pruning model which complements the depth-first backtracking of classical Logic Programming.

Nevertheless, the Prolog language suffers from a serious scalability issue when addressing actual applications. There have been several efforts over the years to overcome this limitation: one which took many years to shape is the ISO standard formulation of *modules* for Prolog. This standard can hardly be considered satisfactory, being heavily influenced by previously existing and conflicting implementations: it can be argued that it essentially introduces the concept of "separate predicate spaces," which are reminiscent of ADA modules. Moreover, the standard is syntactically very verbose, which in itself is a very questionable departure from what we perceive to be one of Prolog's strengths: its syntactic simplicity. Moreover, it is our opinion that the standard completely misses the oportunity it had of assimilating closely related yet well established and vastly more powerful concepts, such as the notions of Object and Inheritance.

An interesting alternative solution to the same problem is that of *Contextual Logic Programming* (CxLP), a model introduced in the late 1980's. Informally, the main point of CxLP is that programs are structured as sets of predicates (*units*) which can be dynamically combined in an execution attribute called a *context.* Goals are seen just as in regular Prolog, except for the fact that the matching predicates are to be located in all the units which make up the current context.

We extended CxLP to attach *arguments* to units: these serve the dual purpose of acting as "unit-global" variables and as state placeholders in actual contexts.

CxLP clearly carries a higher overhead than regular Prolog, as the context must be searched at runtime for the unit that defines a goal's predicate, a process which requires at least one extra indirection compared to straight Prolog; this kind of situation has become more usual and less of a performance issue in recent systems, in Object-Oriented and even in procedural languages, for instance as a result of using dynamically-loaded shared libraries. We have built a prototype implementation of a Contextual Logic Programming language, GNU Prolog/CX, which has surprisingly good performance, considering we aren't performing any optimization.

Finally, we used GNU Prolog/CX to implement the newer components of SIIUE: the Academic Services subsystem. This article reports on the outcome of this initiative.

The rest of this paper is organized as follows: In section 2 we briefly present Universidade de vora's Academic Information System (SIIUE.sac) and how it benefits from GNU Prolog/CX. Section 3 explores the link between contexts and Object-Oriented Programming. Section 4 briefly describes our implementation on top of GNU Prolog. A short conclusion ends the paper.

2 UNIVERSIDADE DE ÉVORA'S ACADEMIC INFORMATION SYSTEM

GNU Prolog/CX has already seen actual use in a real-world application: Universidade de vora's second generation Academic Information System, which is a project that got under way in March 2003 and, at the time of this writing (October 2003), is already in production. This initiative was spurred by the University's decision to simultaneously reorganize all of its undergraduate offerings, to comply with the "Bologna principles," a goal which could not be met by the existing system without very significant and resource-consuming overhauls.

The Academic Information System (SIIUE.sac) is part of Universidade de vora's Integrated Information System (SIIUE), being its latest component and a useful and diverse testbed for the ISCO language.

2.1 ISTO: Evolving the ISCO Programming Language

ISCO (Abreu, 2001) is a Logic Programming language geared towards the development and maintenance of organizational information systems. ISCO is an evolution of the previous language DL and is based on a Constraint Logic Programming framework to define the schema, represent data, access heterogeneous data sources and perform arbitrary computations. In ISCO, processes and data are structured as *classes* which are represented as typed[1] Prolog predicates. An ISCO class may map to an external data source or sink, such as a table or view in a relational database, or be entirely implemented as a regular Prolog predicate. Operations pertaining to ISCO classes include a *query* which is similar to a Prolog call as well as three forms of *update*.

We are presently evolving ISCO to endow it with an expressive means of representing and implicitly using temporal information (Nogueira et al., 2003), the resulting language is called ISTO. In the course of designing ISTO, it was our goal to make further use of contexts, as an implementation became available.

[1]The type system applies to class members, which are viewed as Prolog predicate arguments.

2.2 The Academic Information System; SIIUE.sac

The ISCO architecture for the Academic Information System can be summarized as follows: three different layers correspond to actual physically different networks, interfacing each pair of layers which have contact. The physical separation is provided to ensure that access to higher-numbered layers is exclusively performed by hosts on the layer immediately below.

There is the requirement that, since all validation and authorization is performed by the ISCO layer, the layers above only access any application data via ISCO, hence the application must have three layers (web interface, ISCO and DBMS). The ISCO (GNU Prolog/CX executable) processes come from a pool where they perform initialization tasks before becoming available as query processors, thereby bypassing the overhead of some initialization chores, such as connecting to database servers. It should be noted here that GNU Prolog's architecture is very favourable to its usage as a Prolog implementation for this type of usage, because even complex programs[2] load very fast, as they're mostly native executable code by virtue of the compilation approach, therefore shared by all instances of the program.

ISCO programs may access relational data through ODBC using a GNU Prolog interface with unixODBC, which has been developed within the SIIUE project: this allows for accessing legacy data transparently. The executables are used from within PHP wrapper scripts in web-based interfaces: the PHP extensions have also been developed specifically for use with ISCO.

Although most of the relational databases used are built in PostgreSQL, other relational database engines where considered. This requires ISCO to be independent from the specific RDBMS engine being used. The ISCO compiler is aware of the differences between relational database engines, and generates SQL code appropriate to the specific back-end being used, through the use of different units for each known database back-end, building on similarities between some to exploit multi-level specialization schemes provided by Contextual Logic Programming, as for example in dealing with different versions of the sama RDBMS engine.

We fully integrated ISCO with the PiLLoW (Cabeza and Hermenegildo, 2001) library, a Prolog library for HTML/XML/SGML output and form handling, which is used for web-based development. PiLLoW has been ported to GNU Prolog.

[2]In this case, a typical SIIUE.sac user interface program has around a hunded thousand lines of code.

2.3 SIIUE.sac from a Software Engineering Perspective

Universidade de vora's commitment to develop the SIIUE.sac project was ascertained in early 2003 and the project itself got under way in March 2003 with a team of three experienced programmers. At that time, a complete rewrite of the ISCO tools and GNU Prolog/CX had just recently been rendered operational and the development team had no experience with either Contextual or Constraint Logic Programming, although they had done a few toy projects with Prolog. The project was then scheduled with quarterly milestones which targeted roughly:

1. The academic services internal use (e.g. graduation plans),
2. The student's use (e.g. course registrations) and
3. The faculty members' use (e.g. grading)

It was important that the schedule be met because this endeavour was considered mission-critical, as it involved unknowns at various levels: the technology and tools were very new and the development team was not familiar with the approach.

At the time of this writing, the first two phases had completed successfully and were in production, having been stress-tested with both the introduction of around 40 different graduation plans ranging from Visual Arts to Veterinary Medicine, some with very intricate structures, and the registration for individual courses by approximately 6000 students, averaging 10 courses per student. We only had very minor problems in the first day of the registration period, due to the load peak and which were resolved in under one hour. The third phase was progressing according to plan.

The experience we drew from the deployment of this first application can be summed up in a few points:

- The re-use, whenever appropriate, of existing well-established software components such as Apache, PHP, PiLLoW and LaTeX was essential as it saved us a lot of specification and implementation effort.
- Contextual Logic Programming played a key role in the overall incremental design and implementation process; a few aspects deserve explicit mention:
  - The representation of user sessions as contexts was a significant success, as the concept of session can very naturally be expressed as a context.
  - Role-based authorization and interface generation gained plenty of flexibility and reliability from the systematic use of contexts.
  - Coding the "business logic" as units that respond to standardized messages (e.g. the `item/1`

predicate) enabled us to design compositionally and made it relatively easy to rework implementations and restructure processes while preserving an unchanging interface.

- The choice of relegating the relational database to the role of persistency provider for ISCO seems to have been the correct one. This became particularly obvious at one stage, where an "SQL-like" design (structures represented as a collection of tuples or facts) was replaced with a more "Prolog-like" one (structures represented as a single large term): performance on a particular benchmark went up by a factor of 10 to 100 with that single change, mostly attributable to reduced database traffic.
- The relative ease with which programmers used to procedural languages and SQL adopted a little-documented paradigm and still very experimental development tools was surprising, as they became productive early in the development cycle.

3 CONTEXTS AS OBJECTS WITH STATE

The integration of the Object-Oriented and Logic Programming paradigms has long been an active research area since the late 1980's; take for example McCabe's work (McCabe, 1992). The similarities between Contextual Logic Programming and Object-Oriented Programming have been focused several times in the literature; see for instance the work by Monteiro and Porto (Monteiro and Porto, 1993) or Bugliesi (Bugliesi, 1992).

Other than the implementation-centered reports, previous work on Contextual Logic Programming focuses largely on issues such as the policy for context traversal, what the context becomes once a unit satisfying the calling goal is found, what to do when multiple units provide clauses for the same predicate, how to automatically tie several units together or how to provide encapsulation and concealment mechanisms.

To the best of our knowledge, no published work earlier than (Abreu and Diaz, 2003) builds on the notion of context arguments and their widespread use, even though Miller's initial work (Miller, 1989) already mentions the possibility of using module variables. This feature was present as a "hack" in the first C-Prolog based implementation of Contextual Logic Programming but was a little let down, possibly for lack of an adequate formalization and the nonexistence of convincing examples.

Instead of viewing a context as an opaque execution attribute, as happens in CSM (Natali and Omicini, 1993) for instance, we choose to regard it as a first-class entity, i.e. as a Prolog term. Not only is the context accessible from the program, but *it is intended* that it be explicitly manipulated in the course of a program's regular computation. The performance impact of this option will be succinctly analyzed in section 4: at this point we shall concentrate on the possibilities it allows from an expressiveness point of view, relating Contextual Logic Programming examples to other paradigms whenever appropriate.

3.1 Contexts and Object-Oriented Languages

The following table establishes some parallels between Contextual Logic Programming (CxLP) and Object-Oriented Programming (OOP) terminology, pointing out how units, contexts and context arguments can relate to OOP concepts. The most notable

OOP (Class)	OOP (Object)	CxLP
Object	Object	Context
Message	Message	Goal
Instance variable	Named slot	Unit argument
Method	Method	Predicate

difference between the CxLP and the OOP paradigms has to do with the concept of *inheritance*: instead of being statically defined as in the Class-based Object-Oriented languages, it is completely dynamic for each context (i.e. "object"), as it defines its own structure and, implicitly, its behaviour wrt. messages.

CxLP enables design approaches stemming from both class-based and prototype-based languages (an early example of which is Self (Ungar and Smith, 1987)) in that a unit can be seen as akin to a class as it defines partial state and behaviour and a context, as a self-sufficient object can serve as the basis for the creation of further contexts, either via the extension mechanism or by explicit manipulation of the context term (e.g. a copy).

3.2 Encapsulation and Concealment

These issues are central in Object-Oriented Programming and critical from the Software Engineering point of view. Earlier approaches in Contextual Logic Programming languages proposed several distinct mechanisms, along the lines of having an annotation of some sort to indicate that a given predicate was to be considered "visible" or "hidden", in the sense that a context traversal would see it or not.

Our approach of relying on deep contexts and unit arguments, made possible by the relative efficiency of the prototype implementation, as described in (Abreu and Diaz, 2003), allows us to shun the introduction of yet another set of predicate annotations, because simpler constructions are effectively available, through

the use of unit arguments and the context switch operation: all that is necessary is that the context arguments supply sufficient information for a *new* context to be built, in order to implement the requested method without disclosing the details to the invoking context.

Another situation is where we wish to implement a unit with a particular interface, similar to that of another unit but with some predicates omitted. This can be achieved via a definition for `context/1` as previously suggested, to ensure that the remaining goals will execute in a controlled context.

3.3 Contexts as Implicit Computations

ISTO is a development of ISCO (Abreu, 2001), a Prolog-based mediator language which can transparently access several kinds of information sources, namely relational databases. ISTO relies on GNU Prolog/CX as its compiler's target language and is further described by means of an application in section 2.

Consider a unit `person(ID, NAME, BIRTH_DATE)` which defines the following predicates:

- `item/0` which returns, through backtracking, all instances of the `person/3` database relation by instantiating unit arguments,
- `delete/0` which nondeterministically removes instances of the `person/3` database relation, as restricted by the unit arguments,
- `insert/0` which inserts new instances into the `person/3` database relation, taking the values from the unit argument.

Accessing an "object" specified by a context is always done via one of these predicates, which are to be evaluated in a context which specifies the relation (in this case `person/3`). Assume that there are also predicates with the same name and one argument, which represents the relevant unit with bound arguments, i.e. `item/1`, `delete/1` and `insert/1`. An actual implementation of these predicates could rely on the standard Prolog built-ins `clause/1`, `retract/1` and `assertz/1` or access an external database, as is done in the ISCO compiler (Abreu, 2001).

4 OVERVIEW OF THE PROTOTYPE IMPLEMENTATION

In order to experiment programming with contexts we have developed a first prototype inside GNU Prolog (Diaz and Codognet, 2001). Our main goal was to have a light implementation modifying the current system as little as possible. Due to space restrictions we only give here an overview, the interested reader can consult (Abreu and Diaz, 2003) for more details.

The main change concerns a call to a given predicate `P/N`. If there is no definition for `P/N` in the global predicate table (containing all built-in predicates and predicates not defined inside a unit) then the context must be scanned until a definition is found.

To evaluate the context implementation, we followed a methodology similar to that of Denti et al. (Denti et al., 1993): a goal is evaluated in a context which is made up of a unit which implements the goal predicate, below a variable number of "dummy" units which serve to test the overhead introduced by the context search. We used the exact same methodology as in (Denti et al., 1993). The observed relative performance is much better in GNU Prolog/CX: even in CSM's most favorable situation (the modified WAM), there is a 50% performance hit as soon as there are 5 "dummy" units in the context. Finally note that the "50% performance degradation" threshold is reached when the context comprises about 40 dummy units. This demonstrates the effective ability to extensively use *deep contexts* in actual applications, and is a *sine qua non* requirement for the practical use of such a language feature.

5 CONCLUSIONS AND FUTURE DEVELOPMENTS

The conclusions we can draw from the experience we've had so far include:

- A large application such as SIIUE.sac can bring out fragilities in the implementation of the tools it uses: such was the case, for instance, with GNU Prolog/CX in which a few hitherto unnoticeable bugs became manifest (and were fixed.)
- The in-development status of some of the tools, most notably GNU Prolog/CX, turned out *not* to be a very serious hindrance, as the design discipline made up for the lack of features such as an effective debugger.
- GNU Prolog/CX is well suited to o incremental OO design, as a system can become operational even while still incomplete and underspecified.
- Optimal complex SQL code generation is not as important a goal as we thought it would as it can largely be compensated by the judicious use of the result of simple queries.
- The gradual adoption of Contextual Logic Programming as a design and programming paradigm

has exhibited not too steep a learning curve and is allowing us to further our sensitivity to its different applicability situations and program patterns. Some of these reflect back onto the language itself.

- The developers did have to rid themselves from SQL and procedural language habits, namely in what concerns the manipulation of more complex data structures, in order to extract acceptable performance from the system.

Some directions for future work:

- The removal of some implementation-specific limits (e.g. area sizes).
- The dynamic loading of compiled Prolog code, which will allow for on-the-fly extension of compiled applications.
- Multi-thread execution.
- The development of a Graphical interface (Gnome-Prolog).
- The generic web-based relation browser, as this will greatly decrease interface development time.
- Automatic caching of external database relations, and
- A more extensive performance analysis and tuning under load.

ACKNOWLEDGEMENTS

The work described herein was partly made possible by the bilateral INRIA/GRICES project "Extensions au Logiciel Libre GNU Prolog." Universidade de vora is acknowledged for supporting and funding the SIIUE.sac project.

REFERENCES

Abreu, S. (2001). Isco: A practical language for heterogeneous information system construction. In *Proceedings of INAP'01*, Tokyo, Japan. INAP.

Abreu, S. and Diaz, D. (2003). Objective: in Minimum Context. In Palamidessi, C., editor, *Proceedings of the Eighteenth International Conference on Logic Programming*, volume 2916 of *LNCS*, Mumbai, India. Springer-Verlag. (forthcoming).

Bugliesi, M. (1992). A declarative view of inheritance in logic programming. In Apt, K., editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 113–127, Washington, USA. The MIT Press.

Cabeza, D. and Hermenegildo, M. (2001). Distributed WWW programming using (Ciao-)Prolog and the PiLLoW library. *Theory and Practice of Logic Programming*, 1(3):251–282.

Denti, E., Lamma, E., Mello, P., Natali, A., and Omicini, A. (1993). Techniques for implementing contexts in Logic Programming. In Lamma, E. and Mello, P., editors, *Extensions of Logic Programming*, volume 660 of *LNAI*, pages 339–358. Springer-Verlag. 3rd International Workshop (ELP'92), 26–28 February 1992, Bologna, Italy, Proceedings.

Diaz, D. and Codognet, P. (2001). Design and implementation of the gnu prolog system. *Journal of Functional and Logic Programming*, 2001(6).

McCabe, F. G. (1992). *Logic and Objects*. Prentice Hall.

Miller, D. (1989). A logical analysis of modules in logic programming. *The Journal of Logic Programming*, 6(1 and 2):79–108.

Monteiro, L. and Porto, A. (1993). A Language for Contextual Logic Programming. In Apt, K., de Bakker, J., and Rutten, J., editors, *Logic Programming Languages: Constraints, Functions and Objects*, pages 115–147. MIT Press.

Natali, A. and Omicini, A. (1993). Objects with State in Contextual Logic Programming. In Bruynooghe, M. and Penjam, J., editors, *Programming Language Implementation and Logic Programming*, volume 714 of *LNCS*, pages 220–234. Springer-Verlag. 5th International Symposium (PLILP'93), 25–27 August 1993, Tallinn, Estonia, Proceedings.

Nogueira, V. B., Abreu, S., and David, G. (2003). Using Contextual Logic Programming for Temporal Reasoning. In Pimentel, E. and Brisaboa, N. R., editors, *VIII Conference on Software Engineering and Databases (JISBD 2003)*, Alicante, Spain.

Ungar, D. and Smith, R. B. (1987). Self: The Power of Simplicity. In Meyrowitz, N. K., editor, *Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'87), October 4-8, 1987, Orlando, Florida, Proceedings*, volume 22 of *SIGPLAN Notices*, pages 227–242.