Components for Rule-Based Constraint Programming in the Large

Pierre Deransart, François Fages[1], **Jacques Robin** [2]

[1]Institut de Recherche en Informatique et Automatique (INRIA)
CR de Paris Rocquencourt

[2]Universidade Federal de Pernambuco (UFPe)
Campus da UFPe

{pierre.deransart,francois.fages}@inria.fr, robin.jacques@gmail.com

Abstract. *In this paper we present how constraint based applications may benefit from software engineering modeling approaches. Extension of constraint handling rules are currently developed as a general rule based constraint programming environment including constraint solving, deduction, default reasoning, abduction, belief revision, and planning tasks and methods. But it still lacks general application modeling methods in order to be used in large scale software products. We show how the current collaborative project C4RBCP between UFPe and INRIA brings a contribution to this objective.*

1. Introduction

In this paper we show how Component-Based Software Engineering (CBSE) may give a boost to the use of Rule Based Constraints Programming (RBCP) for large software engineering applications. These two areas bring complementary principles to the general issue of software reuse for Automated Reasoning (AR).

Constraint Logic Programming supports a great ambition for programming: the one of making of programming essentially a modeling task, with equations, constraints and logical formulas. Rule-Based Constraint Programming (RBCP) is the combination of rule systems and constraint resolution. There are huge repositories of global constraints [Beldiceanu et al. 2008], including constraints with rules. [Fages and Martin 2008] introduce a general purpose rule-based modeling language for constraint programming, named Rules2CP which allows in particular to express search strategies and heuristics as preference orderings on variables, values, and *and/or* formulae.

Another paradigm is Constraint Handling Rules (CHR) originally developed by [Fruehwirth and Abdennadher 2003]. Several extensions of constraint handling rules are currently developed as a general rule based constraint programming environment including constraint solving, deduction, default reasoning, abduction, belief revision, and planning tasks and methods. Despite of these many approaches and improvements of modeling with constraints, constraints programming is not used as much as it could be. It still lacks of general modeling methods in order be effectively used for modeling very large software applications.

Component-Based Software Engineering (CBSE) [Atkinson et al. 2002] brings the idea of developing any software in two largely orthogonal steps. The first, design

for reuse step identifies services recurrently needed in a variety of contexts and encapsulate them into components that clearly separate the hidden services realizations from their public specifications as a provided interface. Other components can use the services provided by this interfaces by connecting to their associated ports. This offers them the possibility to outsource by design part of their own provided services realizations. This part must be publicly specified as required interfaces with associated ports. The second, design with reuse step of CBSE can then assemble at low-cost a variety of complex services by just connecting component ports with matching required and provided interfaces.

These CBSE concepts of components, interfaces, ports and assembly are currently only supported by imperative programming platforms (whether object-oriented or not) such as .Net or EJB. They are not supported by RBPC platforms, making constraint solving rule bases monolithic. Our project is to investigate how these CBSE concepts can be adapted to the RBPC language CHR^V (Constraint Handling Rules with Disjunctive bodies), a simple yet powerful extension of CHR, and implemented in the adaptive CHR^V engine CHROME (CHR Online Model-driven Engine) currently under development at CIn-UFPE.

After presenting some aspects of constraint programming and of CBSE, we present some results and perspectives resulting from a collaboration between the INRIA research team “Constraints” and the UFPe-CIN “SE” team.

2. Constraints Programming

Constraint Programming is a field born during the mid 70s from Logic Programming [Kowalski 1974, Colmerauer and Roussel 1996], Linear Programming coming from Operations Research [Schrijver 1998], and Constraint Propagation techniques coming from Artificial Intelligence [Marriot and Stuckey 1998, Rossi et al. 2006]. Its foundation is the use of relations on mathematical variables to compute with partial information. The successes of Constraint Programming for solving combinatorial optimization problems, from pure problems to real problems in industry or commerce, owe much to the bringing of, on the one hand, new local consistency techniques, and, on the other hand, declarative languages which allow control on the mixing of heterogeneous resolution techniques: numerical, symbolic, deductive and heuristic.

Several constraint solvers paradigms have been developed, in particular the Constraint Handling Rule (CHR) approach [Fruehwirth and Abdennadher 2003] which combines in an elegant and effective way the declarativeness of logic programming and the flexibility and empowerment of combining hybrid solvers. In particular CHR offers a good base to specify and implement Rule-Based Constraint Programming (RBCP). CHR has been proven successful in over 100 applications (see `WebCHR: http://www.cs.kuleuven.be/ dtai/projects/CHR/`).

The INRIA “Constraints” team is developing SiLCC (Linear Concurrent Constraint programming) an imperative and concurrent constraint programming language based on a single paradigm: the one of Vijay Saraswat’s concurrent constraint programming extended with constraint systems based on Jean-Yves Girard’s Linear Logic. In the late 90’s he developed the theory of this extension and is now working on its implementation. The LCC paradigm offers at the same time a theoretical framework for analysis, and a valuable guide for practical language design and implementation.

The UFPe group worked on an extension of CHR which shares many similarities with SiLCC, called CHR^V. It is a versatile RBCP language that can serve as a unifying basis to represent constraint solving, deduction, default reasoning, abduction, belief revision, and planning tasks and methods.

In parallel, INRIA contributed to develop tools for constraint solvers behavior analysis, in particular by the OADymPPaC project [Deransart & al 2004] (2001-2004). The recent progresses made on the theory and implementation of traces in CP (definition of the generic trace format GenTra4CP) put it into a firm ground for developing tracers for rule based systems. The figure 1 illustrates variable domain reduction during a 40 queens puzzle resolution.

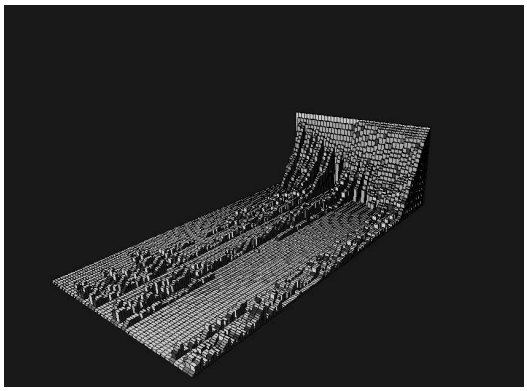

Figure 1. An example of visualisation of constraint propagation (40 queens problem, OADymPPaC project)

3. Component-Based Software Engineering

Components are the leading cost-cutting software reuse technology. A component encapsulates the realization of a service set and makes them accessible from outside through a provided interface with its access port. It may delegate part of this realization to external components by specifying them as required interfaces, each one with its access port. Applications and larger component realizations can be quickly assembled from smaller components by connecting the required ports of client components to the provided ports of server components with matching associated interfaces. In this regard, type systems [Russell and Norvig 2003] provide a first way to ensure some level of consistency in the assembling. Furthermore, each operation of a component's provided and required interfaces can be specified in detail using pre and post conditions. By providing detail service specifications while completely hiding their realizations, components with matching specifications but distinct realizations can be plugged out and in without affecting the rest of the assembly.

A component can possess a publicly visible state lifecycle: only a subset of its provided interface operations can be called in a given state and these calls can in turn alter this state. Each component in an assembly can follow its own independent thread, using its ports as asynchronous communication channels with the other components. The KobrA method [Gross 2004] prescribes how to leverage UML to develop component-based artifacts throughout the entire software lifecycle from requirements, to design and testing.

The .Net and J2EE platforms provide sophisticated component assembly implementation, deployment facilities.

The long term topic of our project is thus the introduction of components to the most practical and challenging variety of RBCP: adaptive solving that integrates constraint simplification, propagation and search, and provides an insightful visual solving trace to ease debugging and evolution of constraint programs.

4. Recent results

CHR^V has emerged as a versatile knowledge representation language, usable for an unparalleled variety of automated reasoning tasks: constraint solving, optimization, classification, subsumption, classical deduction, abduction, truth-maintenance, belief revision, belief update and planning. [da Silva et al. 2008] add default reasoning to this list, by showing how to represent default logic theories in CHR^V. It is also discuss how to leverage this representation together with the well-know correspondence between default logic and Negation As Failure (NAF) in logic programming, to propose an extension CHR$^{V;naf}$ of CHR^V allowing NAF in the rule heads.

[Fages et al. 2008] introduce a modular version of the Constraint Handling Rules language CHR, called CHRat for "modular CHR with ask and tell". Any constraint defined in a CHRat component can be reused both in rules and guards in another CHRat component to define new constraint solvers. Unlike previous work on modular CHR, this approach is completely general as it does not rely on an automatic derivation of conditions for checking entailment in guards, but on a programming discipline for defining both satisfiability (tell) and entailment (ask) checks by CHRat rules for each constraint. The operational and declarative semantics of CHRat are defined, a transformation of CHRat components to flat CHR programs is provided, and the preservation of the semantics is proven. They give examples of the modularization of classical CHR constraint solvers and of the definition of complex constraint solvers in a modular fashion.

5. Conclusion

INRIA and UFPe teams have started a regular cooperation since 3 years in particular through the C4RBCP project, founded by FACEPE, INRIA and MAEE French Ministry. Our cooperation proved to be extremely fruitful supported by complementary objectives.

Both teams share a common interest and previous researches in RBCP and CHR. However, they bring complementary expertise to different aspects of the issues to investigate in the proposed project. The INRIA Rocquencourt team specific expertise fields in constraint programming are theoretical foundations, trace generation and visualization, as well as modules, and type systems, two precursors of software components. The CIn-UFPE specific expertise fields are model-driven engineering , software architecture, component-based modeling, and object-oriented implementation of RBCP engines. This motivates common works.

Furthermore cross cooperation with other European country [Robin et al. 2007] suggests to look for European cooperation. This motivated the authors to elaborate a STREP proposal submitted in the FP7 ICT 2007-3.2.2 framework.

References

Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Paech, B., Wust, J., and Zettel, J. (2002). *Component-based Product Line Engineering with UML*. Addison-Wesley.

Beldiceanu, N., Carlsson, M., and Rampon, J.-X. (2008). Global constraint catalog.

Colmerauer, A. and Roussel, P. (1996). The birth of Prolog. In Bergin, T. J. and Gibson, R. G., editors, *History of Programming Languages*. ACM Press/Addison-Wesley. 1992 draft: http://alain.colmerauer.free.fr/ArchivesPublications/HistoireProlog/19november92.pdf.

da Silva, M. A. A., Fages, F., and Robin, J. (2008). Default reasoning in chr^v. In *Proceedings of the 5th Workshop on Constraint Handling Rules (CHR'08)*, Hagenberg, Austria.

Deransart & al, P. (2004). Outils d'Analyse Dynamique Pour la Programmation Par Contraintes (OADymPPaC). Technical report, Inria Rocquencourt and École des Mines de Nantes and INSA de Rennes and Université d'Orléans and Cosytec and ILOG. Projet RNTL. `http://contraintes.inria.fr/OADymPPaC`.

Fages, F., de Oliveira Rodrigues, C. M., and Martinez, T. (2008). Modular chr with ask and tell. In *Proceedings of the 5th Workshop on Constraint Handling Rules (CHR'08)*, pages 95–110, Hagenberg, Austria.

Fages, F. and Martin, J. (2008). From rules to constraint programs with the Rules2CP modelling language. INRIA Research Report RR-6495, Institut National de Recherche en Informatique.

Fruehwirth, T. and Abdennadher, S. (2003). *Essentials of constraint programming*. Springer Verlag.

Gross, H. (2004). *Component-based Product Line Engineering with UML*. Springer Verlag.

Kowalski, R. A. (1974). Predicate logic as programming language. In *IFIP Congress*, pages 569–574.

Marriot, K. and Stuckey, P. J. (1998). *Programming with constraints: An introduction*. The MIT Press.

Robin, J., Vitorino, J., and Wolf, A. (2007). Constraint programming architectures: Review and a new proposal. In *11th Brazilian Symposium on Programming Languages (SBLP'07)*, Natal, RN, Brazil.

Rossi, F., Beek, P. V., and Hardbound, T. W. (2006). *Handbook of constraint programming*. Elsevier.

Russell, S. and Norvig (2003). *Artificial Intelligence: A Modern Approach (2nd Ed.)*. Prentice-Hall.

Schrijver, A. (1998). *Theory of Linear and Integer Programming*. John Wiley & sons.