Configuration Knowledge Representation using UML/OCL

Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker

Institut für Wirtschaftsinformatik und Anwendungssysteme, Produktionsinformatik,
Universitätsstrasse 65-67, A-9020 Klagenfurt, Austria,
email: {felfernig, friedrich, jannach, zanker}@ifit.uni-klu.ac.at.

Abstract. Today's economy is exhibiting a growing trend towards highly specialized solution providers cooperatively offering configurable products and services to their customers. In this context, knowledge based configurators which support the configuration of complex products and services, must be enhanced with capabilities of knowledge sharing and distributed configuration problem solving. In this paper we demonstrate how UML/OCL can be used as knowledge representation language supporting standardized knowledge interchange thus enabling cooperative problem solving by different configuration environments. We show the representation of configuration domain specific types of constraints in OCL and present an OCL based knowledge acquisition workbench which enables configuration knowledge base development, maintenance and interchange.

1 Introduction

Knowledge based configuration has a long history as a successful application area of Artificial Intelligence [2,9,13,14,17]. Informally, configuration can be seen as a special kind of design activity [22], where the configured product is built of a predefined set of component types and attributes, which can be composed conform to a set of corresponding constraints. Starting with rule-based systems such as R1/XCON [2], higher level representation formalisms were developed to exploit the advantages of more concise representation, faster application development, higher maintainability, and more flexible reasoning. Although these representations have proven their applicability in various real-world applications, there exist a number of obstacles for a far reaching acceptance of this technology.

Integration with industrial software development processes. The integration of development and maintenance of knowledge based systems with industrial software development processes is an important prerequisite for a broader application of AI technologies (such as knowledge based configuration). The so-called knowledge acquisition bottleneck is obvious since configuration knowledge base development and maintenance is only feasible with the support of a knowledge engineer who can handle the formal representation language of the underlying configuration system. In the following we show how to tackle this challenge by

employing UML/OCL[1] [21,26] as configuration knowledge representation language - the language is widely adopted in industrial software development and provides the necessary expressiveness for building configuration knowledge bases.

Knowledge sharing. Available state-of-the-art e-marketplace environments support the integration of configuration knowledge bases by forcing suppliers to provide their product/service descriptions conforming to a given proprietary representation format. A consequence of this is that suppliers offering their products in different e-marketplace environments are forced to provide a separate description of their products and services for each of those environments. In this context UML/OCL can be used as basic mechanism for standardized configuration knowledge representation - products/services provided by different suppliers can be represented as refinements of well-defined domain models based on UML/OCL.

Distributed problem solving. The integration into e-marketplaces must be seen as a first step towards a stronger coupling of knowledge based configuration systems. Today's economy is exhibiting a growing trend towards highly specialized solution providers cooperatively offering configurable products and services to their customers. This paradigm shift requires the provision of distributed configuration problem solving capabilities which are not available in state-of-the-art configuration environments. All the more a standard configuration knowledge representation formalism is needed providing the basis for a clear and common understanding of a given configuration task. Using UML/OCL, configurator components participating in a distributed configuration process are enabled to interchange instance data corresponding to well defined domain models. In this context configurator specific refinements of domain models can be interpreted as *service descriptions*, i.e. the capabilities of a configurator to configure different kinds of products or services.

The Model Driven Architecture (MDA) [19] is the result of ongoing standardization efforts of the Object Management Group (OMG)[2]. The goal is to provide an integrated architecture supporting system interoperability on the application level based on the sharing of metadata. The overall strategy for sharing and understanding metadata consists of the automated development, publishing, management, and interpretation of models [19]. The long term vision for MDA includes applications capable of automatic discovering capabilities/properties of other applications. This is a step from metadata based application integration towards a knowledge-based application integration. In the following we show how UML/OCL can be used as basic description mechanisms for configuration knowledge representation thus providing the necessary preconditions for intelligent configurator integration. Figure 1 sketches the integration of different configuration systems, e-marketplaces, and product catalogs using an MDA based channel for intelligent application integration. The task of the integrated solution provider is the integration of different configurators, e-marketplaces, and

[1] We use a subset of UML/OCL, namely UML class diagrams and OCL expressions of type boolean.

[2] For further details see *www.omg.org*.

product catalogs in order to coordinate the calculation of a solution for a given configuration problem. Note that the integrated solution provider can also dispose of a configuration system (in most cases in the role of the coordinating main configuration system).

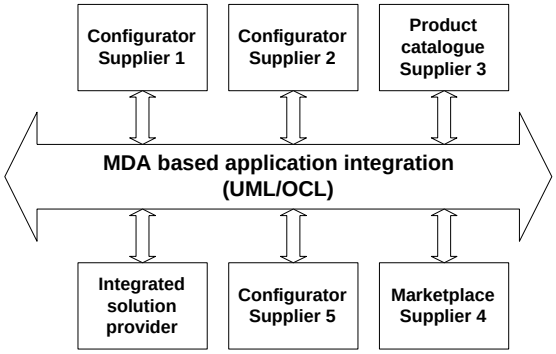

Fig. 1. UML/OCL based configurator integration

The paper is organized as follows. In Section 2 we show the OCL representation of fundamental configuration domain specific constraint patterns. For the presented constraint patterns we show their implementation in our knowledge acquisition workbench (Section 3). In Section 4 we discuss related work and open issues with respect to the application of UML/OCL in the configuration domain.

2 Representing configuration knowledge in UML/OCL

Knowledge sharing and distributed configuration problem solving requires that the configurators share a clear common understanding of the problem definition and its solution. Consequently, it is necessary to agree on the definition of a configuration task and its solution[3].

Definition (configuration task): *In general we assume a configuration task is described by a triple* $(DD, SRS, CLANG)$. DD *represents the domain description of the configurable product (i.e. the product structure and additional constraints on the allowed combinations of instances and attribute settings) and* SRS *specifies the particular system requirements (e.g. customer requirements) defining an individual configuration task instance.* $CLANG$ *comprises a set of concepts for the description of configuration solutions.* □

The Meta-Modelling Language (MML) [4] is a static object-oriented modeling language which serves as a basis for the definition of modeling notations such as

[3] For a detailed discussion on the definition of configuration tasks see [6,17].

UML. MML contains a basic expression language based on OCL. The domain description (DD) of a configurable computer can be defined in MML as follows[4].

```
package computer
    class Software
        capacity: Integer
    end;
    class DTPSoftware extends computer.Software
        inv
        "capacityDTPSoftware"
        capacity = 15
    end;
    class Textedit extends computer.Software
        inv
        "capacityTextedit"
        capacity = 10
    end;
    class HDUnit
        capacity: Integer
    end;
    class IDEUnit extends computer.HDUnit
        inv
        "capacityIDEUnit"
        capacity=25
    end;
    class SCSIUnit extends computer.HDUnit
        inv
        "capacitySCSIUnit"
        capacity=35
    end;
    class MB end;
    class MB1 extends computer.MB end;
    class MB2 extends computer.MB end;      //... further definitions
    class Screen end;
    class Computer
            Software = Set{computer.Software};
            HDUnit = Set{computer.HDUnit};
            MB = Set{computer.MB};
            Screen = Set{computer.Screen}
            // inv
            // constraints for DD ...
    end
end
```

[4] The corresponding UML class diagram is shown in Figure 2.

Based on a given domain description (DD), additional customer requirements (SRS) complete the definition of a concrete configuration task. Customer requirements represent additional constraints on the generic product structure defined in DD, i.e. restrict the set of possible configuration solutions. An example for SRS could be "a motherboard of type $MB1$ and a harddisk unit of type $IDEUnit$". Using MML, these requirements can be defined as follows.

HDUnit = Set{computer.IDEUnit};
MB = Set{computer.MB1};

Based on the above definition of a configuration task, a configuration result can be defined as follows.

Definition (Consistent configuration): *Given a configuration task* (DD, $SRS, CLANG$), *a configuration* (*result*) $CONF$ *is consistent, iff* $DD \cup SRS \cup CONF$ *is satisfiable.* □

Using MML, a concrete configuration $CONF$ can be represented as follows. Note that the objects $software1, hdunit1, mb1, screen1$, and $computer1$ have been created as instances of the corresponding classes $Textedit, IDEUnit, MB1$, and $Computer$ (e.g. $software1$ = @$computer.Textedit$). Those language elements which allow the description of a concrete configuration constitute $CLANG$ (i.e. MML elements used for the description of a concrete configuration).

```
package result extends computer
    software1 = @computer.Textedit
        capacity=10
    end;
    hdunit1 = @computer.IDEUnit
        capacity=25
    end;
    mb1 = @computer.MB1 end;
    screen1 = @computer.Screen end;
    computer1 = @computer.Computer
            software = Set{result.software1};
            hdunit = Set{result.hdunit1};
            mb = Set{result.mb1};
            screen = Set{result.screen1}
    end
   // ...
end
```

Figure 2 shows the product structure of a configurable computer. This configuration model will serve as a working example throughout the following sections. The basic structure is modeled using classes (e.g. the class *Screen*), associations

(e.g. a motherboard(MB) is part of a $Computer$), and generalization hierarchies (e.g. a CPU can be either a $CPU1$ or a $CPU2$). In the following we discuss the OCL representation of three different classes of constraints which are frequently used in the configuration domain[5].

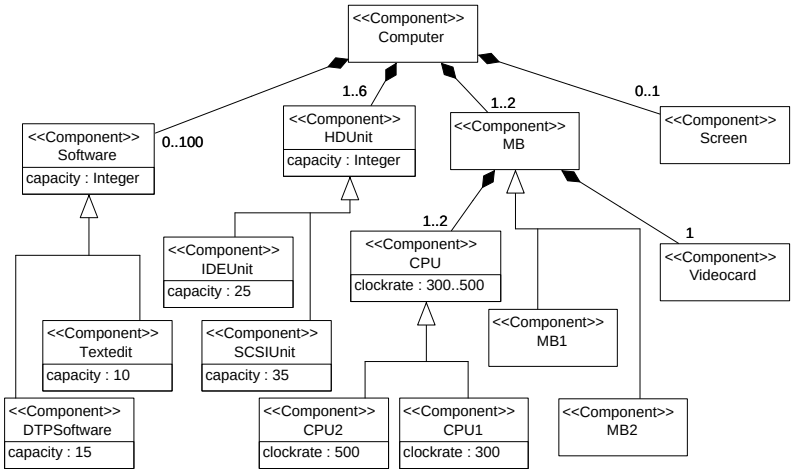

Fig. 2. Example PC configuration model

2.1 Requirements constraints

In some cases, the existence of an instance of a specific type requires the existence of another specific instance in the configuration result. Such a *requires* constraint imposes restrictions of the form $p(R, C_i) \wedge ... \wedge p(R, C_k) \rightarrow p(R, C_l) \wedge ... \wedge p(R, C_n)$, where the expression $p(R, C_x)$ denotes a navigation path from class R to class C_x. Requirements constraints can be imposed on the attribute level as well, i.e. $p(R, C_i.a_i) = va_i \wedge ... \wedge p(R, C_k.a_k) = va_k \rightarrow p(R, C_l.a_l) = va_l \wedge ... \wedge p(R, C_n.a_n) = va_n$ or using any combination of attribute level and class level representation. The expression $p(R, C_x.a_x) = va_x$ denotes a navigation path from class R to class C_x with a corresponding attribute setting $C_x.a_x = va_x$.

Using OCL, requirements constraints on the class level can be expressed as follows[6]:

[5] Note that our OCL parser supports a case insensitive formulation of OCL constraints.

[6] Note that A_{ij} denotes the j^{th} class in the navigation path to class C_i, where we assume that no names for the associations between the classes are provided, i.e. we use class names to construct a navigation expression.

context R inv:
$((self.A_{i1}.A_{i2}.\ ...\ .C_i\text{->size} > 0)$and ... and$(self.A_{k1}.A_{k2}.C_k\text{->size} > 0))$
implies
$((self.A_{l1}.A_{l2}.C_l\text{->size} > 0)$and ... and$(self.A_{n1}.A_{n2}.C_n\text{->size} > 0))$

Requirements constraints on the attribute level can be expressed as follows:

context R inv:
$((self.A_{i1}.A_{i2}.C_i.a_i = va_i)$and ... and$(self.A_{k1}.A_{k2}.C_k.a_k = va_k))$
implies
$((self.A_{l1}.A_{l2}.C_l.a_l = va_l)$and ... and$(self.A_{n1}.A_{n2}.C_n.a_n = va_n))$

In cases, where a subclass S of C within a generalization hierarchy is accessed via a navigation path, additionally the expression C->select(S) must be added to the navigation path - this additional selection is used in the following examples.

Example (IDEUnit requires MB1):
context $Computer$ inv:
(self.$HDUnit$->select(oclIsTypeOf($IDEUnit$))->size>0) implies
(self.MB->select(oclIsTypeOf($MB1$))->size>0)

When translating this constraint into the representation of a concrete configuration system, the output could be the following[7]:

```
oModel.add(oModel.forAll(Computer,
    oModel.imply(
        oModel.gt(oModel.cardinality(oModel.setOf(
            PC.getObjectSetField("HDUnit"),IDEUnit)),0),
        oModel.gt(oModel.cardinality(oModel.setOf(
            PC.getObjectSetField("MB"),MB1)),0))));
```

The method *imply* represents the OCL *implies*, the method *getObjectSetField* realizes OCL *navigation* expressions, the method *cardinality* corresponds to the OCL *size*, the method *setOf* corresponds to the OCL *isOclTypeOf*, and the method *gt* corresponds to the OCL operator ">".

Note that the above translations refer to situations where *0..n* associations are concerned (constraints on class level) and *0..1* associations are concerned (constraints on attribute level). As already mentioned, hybrid variants of these basic constellations are possible - for reasons of readability we omit those variants.

[7] This is the JAVA based representation of the ILOG JConfigurator [12].

2.2 Compatibility constraints

Certain types of instances must not be part of the same final configuration, they are incompatible. Such an incompatibility constraint imposes restrictions of the form $(p(R, C_i) \wedge ... \wedge p(R, C_k)) \rightarrow false$. Incompatibility constraints can be imposed on the attribute level as well, i.e. $(p(R, C_i.a_i) = va_i \wedge ... \wedge p(R, C_k.a_k) = va_k) \rightarrow false$, or using any combination of attribute level and class level representation.

Using OCL, compatibility constraints on the class level can be expressed as follows:

context R inv:
$(((self.A_{i1}.A_{i2}.C_i)$->size $> 0)$and ... and$((self.A_{k1}.A_{k2}.C_k)$->size $> 0)))$
implies false

Compatibility constraints on the attribute level can be expressed as follows:

context R inv:
$((self.A_{i1}.A_{i2}.C_i.a_i = va_i)$and ... and$(self.A_{k1}.A_{k2}.C_k.a_k = va_k))$
implies false

Example (SCSIUnit incompatible with MB1):
context $Computer$ inv:
((self.$HDUnit$->select(oclIsTypeOf($SCSIUnit$))->size>0) and
(self.MB->select(oclIsTypeOf($MB1$))->size>0)) implies false

2.3 Resource constraints

Parts of a configuration task can be seen as a resource balancing task, where some of the classes produce some resources and others are consumers (e.g. the hard-disk capacity needed by the installed software must not exceed the provided hard-disk capacity).

Using OCL, resource constraints can be expressed as follows assuming that $\{C_i, ..., C_k\}$ denotes a set of consumers and $\{C_l, ..., C_n\}$ denotes a set of producers and $\{a_i, ..., a_k\}$, $\{a_l, ..., a_n\}$ denote the resource attributes of those classes. The following constraint represents the case where all the accessed classes are reachable via at least one navigation over an $1..n$ association[8].

context R inv:
$((self.A_{i1}.A_{i2}...C_i.a_i)$->sum + $(self.A_{k1}.A_{k2}...C_k.a_k)$->sum) <=
$((self.A_{l1}.A_{l2}...C_l.a_l)$->sum + $(self.A_{l1}.A_{l2}...C_l.a_l)$->sum)

[8] Note that the *sum* operator is an additionally introduced aggregation function.

Example (Consumed Software capacity $\leq$ HDUnit capacity):
context *Computer* inv:
(self.*Software.capacity*)->sum <= (self.*HDUnit.capacity*)->sum

2.4 Additional constraints

The constraints discussed in the previous subsections represent frequently modeled restrictions on configuration models. For these types of constraints we provide a set of constraint schemes with the corresponding input masks (see Section 3). When defining more complex constraints on a configurable product, we also allow a text-based formulation of OCL constraints (see also Section 3).

3 Knowledge acquisition workbench

CAWICOMS[9] is an EU-funded project which aims at the provision of new technologies allowing the application and integration of available configuration systems in Web-based environments. One of the central parts of the CAWICOMS environment is the knowledge acquisition workbench supporting standardized configuration knowledge acquisition, interchange, and maintenance in distributed environments. The main component of the knowledge acquisition workbench is an Rational Rose add-in especially tailored for modeling the structure of configurable products and services. A constraint editor supports the design of constraints on the product/service structure. Within the knowledge acquisition workbench OCL is used as standard constraint representation language. For frequently used configuration domain specific constraints we provide a set of constraint schemes with a corresponding graphical interface.

Figure 3 shows the representation of compatibility constraints between different classes of the configuration model, where the product structure is represented in the left-hand-side window. The user can select classes and corresponding attributes to formulate constraints conform to the structure discussed in Section 2 - these constraints are automatically translated into an OCL based representation which is further translated into representation of the underlying configuration system, i.e. a configuration task (see Section 2) is defined by automatically generating a configuration knowledge base from a given UML/OCL specification - satisfiability of the configuration task can now be checked by the underlying configuration system [12].

Figure 4 shows the representation of resource constraints between different producer and consumer classes of the configuration model. In this case the product model is represented in the left-hand-side as well as in the right-hand-side window in order to provide different input areas for producers and consumers. The structure of the input mask for *requires* constraints is very similar to that

[9] CAWICOMS is the acronym for Customer-Adaptive Web Interface for the Configuration of products and services with Multiple Suppliers (EU-funded project IST-1999-10688).

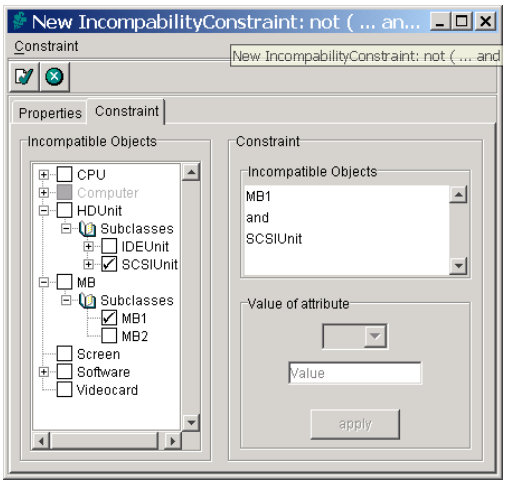

Fig. 3. Interface for compatibility constraints

for resource constraints - therefore we omit the presentation of the input mask for *requires* constraints here. Finally, constraints can be directly entered as OCL constraints - for this purpose a simple text editor is provided (see Figure 5).

Knowledge interchange between different configuration environments is supported by the exchange of XML representations of product structures and constraints (see Section 2). On the basis of an OCL parser, configuration knowledge bases (e.g. for the ILOG JConfigurator [12]) can be automatically generated using configurator specific translation routines. Based on those knowledge bases, a distributed problem solving process is triggered which is based on state-of-the-art distributed constraint satisfaction algorithms [23,27]. The communication protocol between the systems participating in the problem solving process is realized via SOAP [25], i.e. configuration services are represented as Web services [7]. The CAWICOMS runtime environment provides mechanisms for integrating supplier product catalogs stemming from ERP systems or e-marketplace environments. The integration of product catalogs is supported by the possibility of modeling abstract references (catalog entries) to those catalogs using the Rational Rose add-in - the entries are uploaded by the configurator at runtime.

4 Related Work

Up to now we have presented the basic concepts behind the CAWICOMS knowledge acquisition workbench. The underlying knowledge representation language OCL includes the concepts needed for the construction of configuration knowledge bases, i.e. it is an excellent basis for a common configuration knowledge representation language. The availability of such a standardized language is a crucial success factor for integrating configuration technologies into industrial

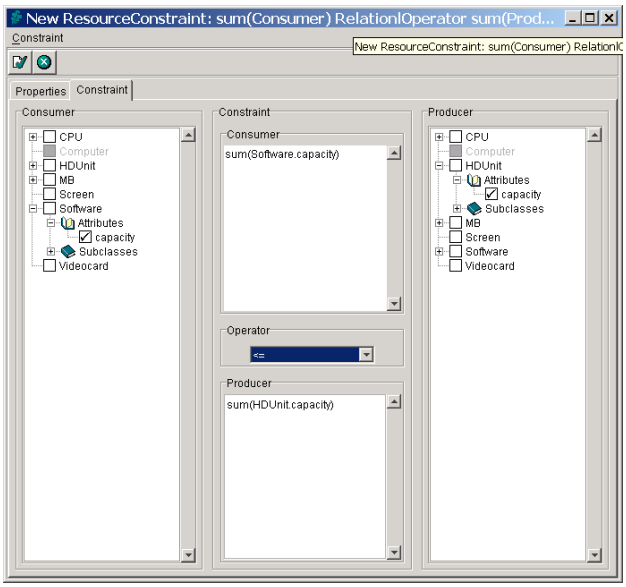

Fig. 4. Interface for resource constraints

software development processes and for configuration knowledge base integration. Current implementations based on OCL focus on different facets of model checking (e.g. [5,20]). In [20] an approach is presented for validating UML models and corresponding OCL constraints, which is based on animation. The user of the system can provide a set of snapshots representing possible states of the considered system. These snapshots can be furthermore validated against the built UML model and the corresponding OCL constraints. If a snapshot (positive example) does not fulfill the model constraints, there could be constraints in the model which are too restrictive and consequently must be eliminated or relaxed. Constraints on the system may be too weak, i.e. there are inconsistent constellations in the snapshot (negative example), which are not considered by the actually formulated model constraints.

The Meta-Modelling Language (MML) [4] is a static object-oriented modeling language which serves as a basis for the definition of modeling notations such as UML. MML contains a basic expression language based on OCL - the underlying implementation (MMT - Meta-Modelling Tool) provides basic model generation facilities (e.g. for generating instances of a given relationship or for instantiating variables in expressions). The work of [4] is the first step towards the provision of model generation mechanisms which are crucial for the effective application of OCL in configuration domain specific problem settings. There is a large spectrum of approaches to model generation already existing in the AI community [9,14,16,17] which are also implemented in state-of-the-art configuration systems. The implementation of model generation mechanisms especially

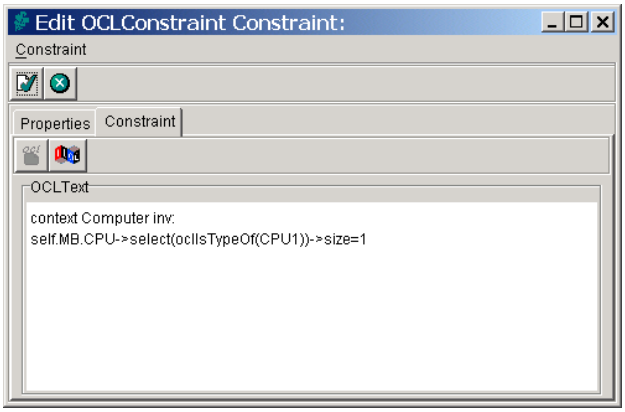

Fig. 5. Interface for basic OCL constraints

in the context of building a configurator environment on the basis of OCL is the subject of future work.

In [1] experiences from the application of OCL in industrial software development processes are discussed. In principle, OCL seems to be quite useful and software engineers and even domain experts with a technical background are able to apply OCL for stating formal constraints on a given object model. Especially software engineers accepted OCL because of the similarities of its syntax to object-oriented programming languages. However, [1] point out that additional, more intuitive concepts are needed in order to support effective introduction of OCL constraints. They made the observation that software engineers tried to change an objectt's state, what is prohibited by the declarative semantics of OCL. In order to tackle this challenge, [1] introduce the notion of constraint schemes. These schemes represent parameterizable constraints, which can be differently instantiated depending on the actual situation. For example, a constraint schema could restrict the occurrence of objects of a class to an upper bound. In this case the upper bound is represented by a variable which must be instantiated in order to instantiate the corresponding OCL constraint. The input masks for *requires*, *incompatible*, and *resource* constraints provided by the CAWICOMS knowledge acquisition component can be interpreted as a facet of such constraint schemes.

OIL [8] and DAML+OIL [24] are ontology representation languages developed within the context of the Semantic Web [3]. These languages should serve as a basis for enabling Web technologies providing access to Web resources not only to humans but as well to applications supporting Web services such as information brokering, information filtering, intelligent search or service synthesis [18]. Triggered by the requirement for more flexibility and expressiveness of those languages there are ongoing efforts to increase the expressiveness of Web ontology languages. DAML-L [15] is a language which builds upon the basic concepts of DAML. XML Rules [11] and CIF (Constraint Interchange Format) [10] are similar approaches with the goal to provide rule languages for the Se-

mantic Web. As a result of our investigations [7], Semantic Web representation languages are suitable for configuration knowledge representation, however an additional language is needed supporting an intuitive formulation of constraints on product/service structures. An interesting question in this context is, how the concepts of OCL can be integrated into languages such as OIL or DAML+OIL in order to exploit the advantages of both paradigms.

5 Conclusions

We have presented the CAWICOMS knowledge acquisition workbench which supports UML/OCL based construction of configuration knowledge bases. Using UML/OCL as standard representation formalism for defining configuration tasks facilitates knowledge sharing and cooperative configuration problem solving. The development of configuration systems is integrated into industrial software development processes which is the major precondition for a broader and more effective application of configuration technologies. Finally, the approach shows how UML/OCL can be applied for knowledge-based application integration which is one the goals of the OMG standardization efforts.

References

1. T. Baar, R. Hähnle, T. Sattler, and T.H. Schmitt. Entwurfsmustergesteuerte Erzeugung von OCL Constraints. In K. Mehrhorn and G. Snelting, editors, *Informatik 2000, 30. Jahrestagung der Gesellschaft für Informatik*, pages 389–404. Springer-Verlag, 2000.
2. V.E. Barker, D.E. O'Connor, J.D. Bachant, and E. Soloway. Expert systems for configuration at Digital: XCON and beyond. *Communications of the ACM*, 32(3):298–318, 1989.
3. T. Berners-Lee. *Weaving the Web.* Harper Business, 2000.
4. T. Clark, E. Evans, St. Kent, and P. Sammut. The MMF Approach to Engineering Object-Oriented Design Languages. In *Workshop on Language Descriptions, Tools and Applications*, 2001.
5. A. Evans. Reasoning with UML class diagrams. In *Proceedings of the Workshop on Industrial Strength Formal Methods(WIFT'98)*, Florida, USA, 1998. IEEE Press.
6. A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner. Consistency-Based Diagnosis of Configuration Knowledge Bases. In *Proceedings of the* 14^{th} *European Conference on Artificial Intelligence (ECAI 2000)*, pages 146–150, Berlin, Germany, 2000.
7. A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker. Semantic Configuration Web Services in the CAWICOMS project. *Proceedings of* 1^{st} *International Semantic Web Conference*, pages 192–205, 2002.
8. D. Fensel, F. vanHarmelen, I. Horrocks, D. McGuinness, and P.F. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.
9. G. Fleischanderl, G. Friedrich, A. Haselböck, H. Schreiner, and M. Stumptner. Configuring Large Systems Using Generative Constraint Satisfaction. *IEEE Intelligent Systems*, 13(4):59–68, 1998.

10. P. Gray, K. Hui, and A. Preece. An Expressive Constraint Language for Semantic Web Applications. In *Proceedings of the IJCAI 2001 Workshop on E-Business and the Intelligent Web*, pages 46–53, Seattle, WA, 2001.
11. B.N. Grosof. Standardizing XML Rules. In *Proceedings of the IJCAI 2001 Workshop on E-Business and the Intelligent Web*, pages 2–3, Seattle, WA, 2001.
12. U. Junker. Preference programming for configuration. In *Proceedings of Workshop on Configuration (IJCAI'01)*, Seattle, WA, USA, 2001.
13. E.W. Jüngst M. Heinrich. A resource-based paradigm for the configuring of technical systems from modular components. In *Proceedings of the* 7^{th} *IEEE Conference on AI applciations (CAIA)*, pages 257–264, Miami, FL, USA, 1991.
14. D. Mailharro. A classification and constraint-based framework for configuration. *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing Journal, Special Issue: Configuration Design*, 12(4):383–397, 1998.
15. Sh. McIlraith, T.C. Son, and H. Zeng. Mobilizing the Semantic Web with DAML-Enabled Web Services. In *Proceedings of the IJCAI 2001 Workshop on E-Business and the Intelligent Web*, pages 29–39, Seattle, WA, 2001.
16. S. Mittal and B. Falkenhainer. Dynamic Constraint Satisfaction Problems. In *Proceedings of the National Conference on Artificial Intelligence (AAAI 90)*, pages 25–32, Boston, MA, 1990.
17. S. Mittal and F. Frayman. Towards a Generic Model of Configuration Tasks. In *Proceedings* 11^{th} *International Joint Conf. on Artificial Intelligence*, pages 1395–1401, Detroit, MI, 1989.
18. E. Motta, D. Fensel, M. Gaspari, and V.R. Benjamins. Specifications of Knowledge Components for Reuse. In *Proceedings of* 11^{th} *International Conference on Software Engineering and Knowledge Engineering*, pages 36–43, Kaiserslautern, Germany, 1999.
19. J.D. Poole. Model-Driven Architecture: Vision, Standards And Emerging Technologies. In *Workshop on Metamodeling and Adaptive Object Models, ECOOP 2001*, 2001.
20. M. Richters and M. Gogolla. Validating UML Models and OCL Constraints. In *Proceedings of the* 3^{rd} *International Conference on the Unified Modeling Language (UML'2000)*, volume 1939, pages 265–277, York, UK, 2000. Springer Lecture Notes in Computer Science.
21. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.
22. D. Sabin and R. Weigel. Product Configuration Frameworks - A Survey. In B. Faltings and E. Freuder, editors, *IEEE Intelligent Systems, Special Issue on Configuration*, volume 13,4, pages 50–58. IEEE, 1998.
23. M.C. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous search with aggregations. In *Proceedings of* 17^th *National Conference on Artificial Intelligence (AAAI)*, pages 917–922, Austin, TX, USA, 2000.
24. F. vanHarmelen, P.F. Patel-Schneider, and I. Horrocks. A Model-Theoretic Semantics for DAML+OIL. *www.daml.org*, March 2001.
25. W3C. Simple Object Access Protocol (SOAP), ver. 1.2. *www.w3c.org*, 2001.
26. J. Warmer and A. Kleppe. *The Object Constraint Language - Precise Modeling with UML*. Addison Wesley Object Technology Series, 1999.
27. M. Yokoo, E.H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, 1998.