Construction of Configuration Models

Lothar Hotz

HITeC e.V., Universität Hamburg

Abstract

In this paper, a novel approach for creating configuration models is supplied y introducing a meta-knowledge base that enables the construction of configuration models. The meta-knowledge base represents all knowledge bases that can be expressed with a given configuration language, in the case of this paper, with the Component Description Language CDL. The meta-knowledge base itself is again represented with CDL and thus, at the metalevel it can use configuration tools that relay on CDL. With this approach inference techniques that are normally used for configuration of technical systems can be applied for the construction of configuration model, i.e. during knowledge acquisition and evolution.

1 Introduction

Knowledge-based configuration has its origin in the task of configuring physical components like drive systems (Ranze et al., 2002) or elevators (Marcus et al., 1988). For example in (Günter, 1995) configuration is defined as "the composition of technical systems from parameterisable objects to a configuration, that fulfills a certain task" or Stefik defines in (Stefik, 1995) configuration tasks as tasks that ``select and arrange instance of parts from a set". The focus is set on the composition of parts to aggregates and thus, on the compositional relation `has-parts`.

Naturally, in all approaches *descriptions* of objects are composed, not the physical objects themselves. By doing so, configuration can be understood as *model construction* (Buchheit et al., 1995; Hotz and Neumann, 2005; Hotz, 2009). From the configuration point of view, model construction deals with the composition of arbitrary artifacts on the basis of a logical theory. Hereby, a strict separation of the logical theory, i.e. the knowledge base or *configuration model*, and the logical model, i.e. the *configuration* or better *construction*, is issued. Starting from a knowledge base a configuration system composes a construction that is consistent with the knowledge base, i.e. a logical model of the knowledge base is created.
Following this understanding of the configuration task the above mentioned `has-parts` relation is more a `has` relation, which is applied to various domains e.g.

services (Tiihonen et al., 2006), where e.g. a client *has* a certain insurance demand, or to software (Hotz et al., 2004), where a software component *has* a certain feature, to scene interpretation (Hotz, 2006), where a certain scene description *has* observed or hypothized objects or actions. The `has` relation determines what part descriptions are to be integrated in a resulting construction (i.e. a system description). By taking up such a perspective sometimes considered conceptual mismatches (Tiihonen et al., 2006), which may come up, when using configuration systems in non-physical domains, are avoided.

Taking a further step, one may look at configuration models as a type of software that is constructed during a knowledge-acquisition process. Thus, the questions arise: "Can the construction of configuration models be supported by configuration tools?" or "What are the parts that are composed in such an approach?" or "How does a configuration model that enables the configuration of configuration models (i.e. a *meta-configuration model*) look like?".

An application of such a meta-configuration model is naturally to support the knowledge-acquisition process needed for knowledge-based configuration systems. In a first phase of a knowledge-acquisition process, the typically tacit knowledge about a domain is extracted by applying knowledge-elicitation methods and high interaction between a knowledge engineer and the domain expert (*knowledge-elicitation phase*). A model sketch is the result, which in turn is formalized during the *domain-representation phase*. During this phase a configuration model is created. The configuration model has to be expressed with the facilities of a configuration language. The meta-configuration model can be used to check such configuration models for being consistent with the configuration language. Thus, by using the meta-configuration model as a knowledge base of a configuration system, the domain-representation phase can be supported similarly to a configuration process.

In this paper, we will elaborate answers to the mentioned questions by first presenting a construction language, i.e. the *Component Description Language* CDL, which enables the description of domain objects (see Section 2). We than investigate in a concept for a configurator that enables the configuration of arbitrary configuration models, i.e. a meta configurator and its *meta-configuration model* (Section 3). We partly implement such a meta configurator by using the configuration system KONWERK (Günter and Hotz, 1999). A discussion and a summary are provided in Section 4 and Section 5 respectively.

2 The Component Description Language

2.1 A Sketch of CDL

The Component Description Language CDL introduced here is similar to existing other configuration languages as they are described in (Soininen et al., 1998;

Stumptner, 1997; Felfernig et al., 2002; Cunis et al., 1991; Günter, 1995). The language mainly consists of two modeling facilities:

Concept Hierarchy Domain objects are described using *concepts*, a specialization hierarchy (based on the `is-a` relation), and structural relations. Concepts gather all properties, a certain set of domain objects has, under a unique name. A specialization relation relates a *super-concept* to a *sub-concept*, where the later inherits the properties of the first. The structural relation is given between a concept *c* and several other concepts *r*, which are called *relative concepts*. With structural relations a compositional hierarchy based on the `has-parts` relation can be modeled as well as structural relationships like `has-feature` or `has-concept`. Parameters specify domain-object attributes with value intervals, sets of values (enumerations), or primitive values. Parameters and structural relations of a concept are also referred to as *properties* of the concept. *Instances* are instantiations of the concepts and represent concrete domain objects. When instantiated, the properties of an instance are initialized by the values or value ranges specified in the concepts.

Constraints *Constraints* summarize conceptual constraints, constraint relations, and constraint instance. *Conceptual constraints* consist of a condition and an action part. The condition part specifies a *structural situation* of instantiated concepts. If this structural situation is fulfilled by some instances (the instances *match* the structural situation), *constraint relations* that are formulated in the action part are instantiated to *constraint instances*.[1] Constraint relations can represent restrictions between properties like `all-isp` or `create-instance`. Figure 1 shows the definition of the predefined constraint relations used in the following. The constraint relations `create-instance` and `integrate-instance` are later used for constructing structural relations and thus, provide main facilities for creating resulting constructions.

Knowledge processing is done by the *inference techniques* taxonomical reasoning, value-related computations like interval arithmetic, establishing structural relations, and constraint propagation. The structural relation as the main machinery causes the constructive notion of the language: if such a relation is given between a concept *c* and several relative concepts *r*, depending on what exists first as instances in

`integrate-instance <set1 instance1 instance2 set2>` Integrate `instance1` into `set2` and `instance2` into `set1`. `instance1` and `instance2` than have established structural relations among them.
`all-isp <set type>` Ensures that all objects in `set` are subtype of `type`.

Figure 1: Some predefined relations of CDL.

[1] Thus, conceptual constraints are similar to rules, except the action part yields to instantiations of constraint relations not to changes in objects like rules do.

the construction (*c* or one or more of the relative concepts *r*), instances for the other part of the relation are created and the construction increases.

```
(define-relation :name has-elements
  :inverse element-of
  :mapping m-n)

(define-concept :name Door
  :specialization-of Opening
  :element-of
   ((:type Scene-Aggregate :min 0 :max 2)
     :=
    (:type Entrance :min 0 :max 1)
    (:type Balcony :min 0 :max 1)))

(define-concept :name Entrance
  :specialization-of Opening
  :has-elements
   ((:type Scene-Object :min 1 :max 3)
     :=
    (:type Door :min 1 :max 1)
    (:type Wall :min 0 :max 1)
    (:type Roof :min 0 :max 1)
    (:type Stairs :min 0 :max 1)))

(define-concept :name Balcony
  :specialization-of Scene-Aggregate
  :has-elements
   ((:type Scene-Object :min 1 :max 3)
     :=
    (:type Railing :min 1 :max 1)
    (:type Window :min 0 :max 1)
    (:type Door :min 0 :max 1)))
```

Figure 2: Example of a concept definition in CDL. The structural relation `has elements` is defined, which relates one aggregate with several parts and one part with several aggregates. Furthermore, several concepts are defined with number restricted structural relations. The right side of the operator `:=` consists of the super-concept of all relative concepts and the total minimal and maximal number of those concepts. The left side restricts the number of each type.

A configuration process (or better *model-construction process*) applies these inference techniques in a certain way and constructs step-by-step a construction. At each step a *current partial construction* is issued. The knowledge needed for this processing is modeled by further modeling facilities, i.e. a *task description* and *procedural knowledge*. The task description is given in terms of an aggregate, which must be configured (the goal), and possibly additional restrictions such as choices of parts, prescribed properties, etc. Furthermore, the configuration process provides a stepwise composition of a construction. Each step is one of the following kinds of construction steps: *top-down structuring* (e.g. *aggregate instantiation*), *bottom-up structuring* (e.g. *part integration*), *instance specialization*, and *parameterization*. A step reduces a property value of an instance to a subset or finally to a constant. Procedural knowledge declaratively describes the selection of those steps and the inference techniques to be used.

Thus, adding facilities for task descriptions and procedural knowledge to CDL one gets a complete configuration language like the Configuration Knowledge Modeling Language CKML described in (Hotz et al., 2006). However, in this paperw we concentrate on the first mentioned modeling facilities of concepts and constraints and try to express them with CDL again. CDL is fully described in (Hotz, 2009).

2.2 Parts of the Metamodel of CDL

For expressing the goals of this paper, we give more details for the definition of structural relations in CDL.

Concepts and constraints of CDL are given by an abstract syntax (see Figure 3), a concrete syntax (see Figure 2 for an example[2]), and several consistency rules.

For describing CDL with an abstract syntax, we introduce three facilities: a *knowledge element*, a *taxonomical relation* between knowledge elements, and a *compositional relation* between knowledge elements. However, these facilities are not to be mixed up with the above mentioned CDL facilities: *concepts*, a *specialization relation*, and *structural relations*. See Figure 3, a CDL concept is represented with a knowledge element of name *concept*, a CDL structural relation is represented with the knowledge element *relation-descriptor*. The fact that CDL concepts can have several structural relations is represented with a compositional relation with name *has-relations*. Similarly parameters are represented with a compositional relation with name *has-parameters*. Thus, the above mentioned modeling facilities of CDL are represented with these metalevel facilities.

In Figure 4 further parts of the metamodel are given for representing structural relations. The fact that a concept is related by a structural relation of other concepts (the relative concepts) is represented with three knowledge elements and three compositional relations in a cyclic manner.

Several consistency rules define the meaning of the syntactic constructs. Looking at the structural relation, one rule defines that the types of the relative concepts of a structural relation have to be sub-concepts of the concept on the left side of

[2]For the examples, the facade domain is used where the domain objects are parts of houses like balcony, door, and stories. The purpose is to construct scene interpretations from facade images (Hotz, 2008).

the operator := (*rule-5*). Additionally consistency rules are given that check CDL instances, e.g. one rule defines when instances match a conceptual constraint (*rule-6*).

3 A Concept for a Meta Configurator

3.1 What CDL provides

The main feature of CDL is given by the use of its inference techniques like constraint propagation (see Section 2). By representing the knowledge of a domain with modeling facilities of CDL (like concepts with specialization, structural, and constraint relations) those inference techniques can be applied for model construction. This representation is basically a generic description of domain objects of a domain at hand. For the representation of concrete domain objects this description is instantiated. Such instances are related to each other through the relations. Furthermore, instances can be checked for concept membership.

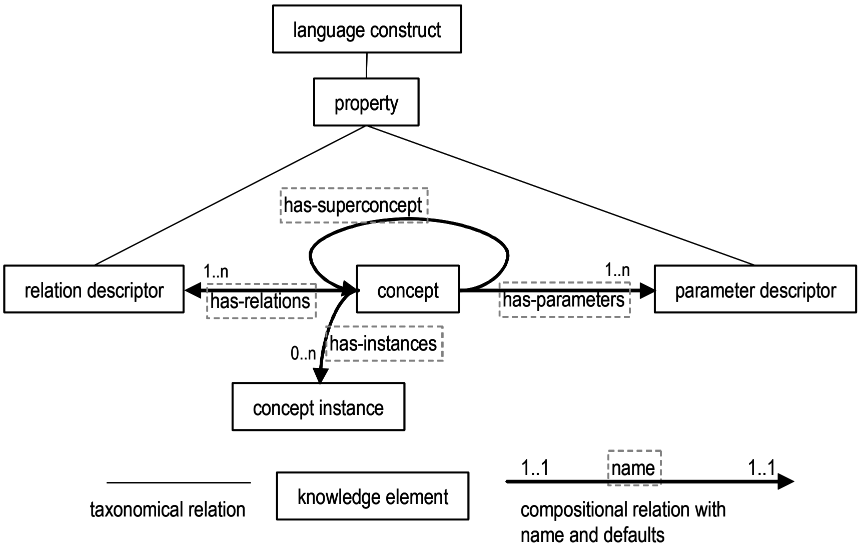

Figure 3: Metamodel for a concept of CDL

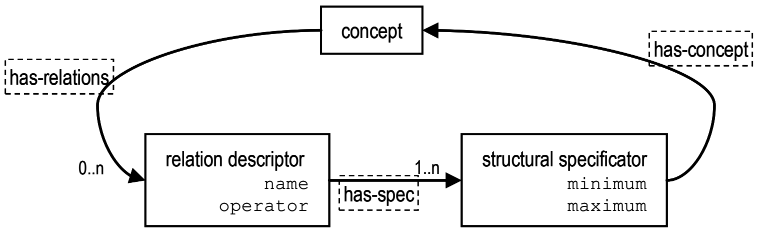

Figure 4: Metamodel for a structural relation of CDL

What does this mean for the representation of CDL in CDL? In this case, the domain consists of CDL knowledge bases. A Meta-CDL knowledge base (Meta-CDL-KB) generically represents all knowledge bases that can be expressed with CDL (see Section 3.2). Doing so, the above mentioned inference techniques can be used for CDL knowledge bases. For example, a knowledge base *G* for a certain domain *D* (like the facade domain) can be created through instances of concepts of the Meta-CDL-KB. Examples are *concept-mm* for representing concepts and *parameter-mm* for representing parameters (see Figure 5). These concepts are related to each other e.g. *concept-mm has-parameters parameter-mm*. Through a configuration process, which applies the inference techniques of CDL in a certain way, a knowledge base *G* of a domain *D* can be created. Furthermore, a given knowledge base can be checked, if it can be constructed in principal with the Meta-CDL-KB, i.e. if it is a CDL knowledge base. An architecture that supports these tasks is given in Section 3.3.

3.2 CDL in CDL

For the presentation of CDL, in Section 2 three facilities are used, i.e. *knowledge elements*, *taxonomical relation*, and *compositional relation*. Those are mapped to the CDL constructs *concept*, *specialization relation*, and *structural relation* respectively. For example, the knowledge elements for describing the CDL facilities in Figure 3 concept, relation, and parameters are represented with the metaconcepts *concept-mm*, *relations-descriptor-mm*, and *parameter-mm* (see Figure 5).

Furthermore, the consistency rules of CDL have to be represented. This is achieved by defining appropriate constraints, which in turn use value-related computations (Section 2) for computing appropriate values. In Figure 6 a conceptual constraint is represented, which checks the types of a structural relation.[3]

Also instances can be represented on the metalevel by including a metaconcept *instance-mm* for them. Through these instances also conceptual constraints and their matching instances can be represented (see Figure 7). Furthermore, the fact that instances fulfill a certain conceptual constraint is represented through establishing

[3] For a complete mapping of the CDL consistency rules to conceptual constraints see (Hotz, 2009).

appropriate relations using the constraint relation `integrate-instance`. Note that also self references can be described, e.g. a *concept-mm* is related to itself via the *has-superconcept-mm* relation (see also the loop in Figure 3).

```
(define-concept :name concept-mm
  :specialization-of named-domain-object-mm
  :concept-of-dom-mm (:type domain-mm)
  :superconcept-of-mm
   (:type concept-mm :min 0 :max inf)
  :in-some-mm (:type some-mm :min 0 :max inf)
  :has-superconcept-mm
   (:type concept-mm :min 0 :max 1)
  :has-relations-mm
   (:type relation-descriptor-mm :min 0 :max inf)
  :has-parameters-mm
   (:type parameter-mm :min 0 :max inf)
  :has-instances-mm
   (:type instance-mm :min 0 :max inf))

(define-concept :name relation-descriptor-mm
  :specialization-of named-domain-object-mm
  :relation-of-mm (:type concept-mm)
  :has-left-side-mm (:type some-mm :min 1:max 1)
  :has-right-side-mm (:type some-mm :min 0:max inf)
  :has-relation-definition-mm
   (:type relation-definition-mm :min 1:max 1))

(define-concept :name some-mm
  :specialization-of domain-object-descriptor-mm
  :parameters ( (lower-bound [0 inf])
                (upper-bound [0 inf]))
  :in-relation-left-mm
   (:type relation-descriptor-mm)
  :in-relation-right-mm
   (:type relation-descriptor-mm)
  :some-of (:type concept-mm))

(define-concept :name instance-mm
  :specialization-of named-domain-object-mm
  :instance-of-dom-mm (:type domain-mm)
  :instance-of-mm (:type concept-mm)
  :matching-instance-of-mm
   (:type conceptual-constraint-mm)
  :has-relations-mm
    (:type relation-descriptor- mm :min 0 :max inf)
  :has-parameters-mm
    (:type parameter-mm :min 0 :max inf))
```

Figure 5: Formalizing the knowledge elements shown in Figure 3 with CDL concepts.

```
(define-conceptual-constraint :name consistency-rule-5
  :structural-situation
   ((?c    :name concept-mm)
    (?rd   :name relation-descriptor-mm
           :relation-of-mm ?c)
    (?svt  :name some-mm
           :in-relation-left-mm ?rd)
    (?stdi :all :name some-mm
           :in-relation-right-mm ?rd))
  :constraint-calls
   ((all-isp ?stdi ?svt)))
```

Figure 6: A conceptual constraint representing consistency rule 5. The concepts of the right side of a relation descriptor has to be sub-concepts of the left side.

```
(define-concept :name conceptual-constraint-mm
  :specialization-of named-domain-object-mm
  :structural-situation
   (:type concept-expression-mm :min 1 :max inf)
  :constraint-calls
   (:type constraint-call-mm :min 1 :max inf)
  :matching-instances
   (:type instance-mm :min 0 :max inf))

(define-conceptual-constraint
  :name instance-consistency-rule-6
  :structural-situation
   ((?cc :name conceptual-constraint-mm)
    (?i  :name instance-mm
         :self (:condition
                 (instance-matches-cc-p *it* ?cc))))
  :constraint-calls
   ((integrate-instance-relation ?i
      (matching-instance-of ?i) ?cc
      (matching-instances ?cc))))
```

Figure 7: Describing conceptual constraints with there matching instances on the metalevel.

3.3 A Meta-Knowledge Server

In this section, we describe the use of the Meta-CDL-KB for the construction of a CDL knowledge base for arbitrary domains. This use is realized by introducing a Meta-Knowledge Server (MKS) for supervising the construction of the CDL knowledge base. The MKS handles the current status of the evolving CDL knowledge base during the knowledge acquisition process as well as the current status of CDL instances during a configuration process.

In Figure 8, we sketch the first case. the MKS uses the Meta-CDL-KB as configuration model *M*. Furthermore, MKS uses the model-construction process for supervising the construction of a configuration model *G* of a given domain. If e.g. a concept *c* of the domain is defined with `define-concept` the MKS is informed. The MKS observes the activities during the construction of the CDL knowledge base, i.e. during the domain-representation phase. The MKS

- supplies services like *check-knowledge-base*, *add-conceptual-constraint*,
- creates appropriate instances of Meta-KB-CDL metaconcepts (e.g. *concept-mm* or *conceptual-constraint-mm*),
- applies the typical model-construction process by using procedural knowledge,
- uses constraint propagation for checking the consistency rules,
- completes the CDL knowledge base by including mandatory parts, and
- checks consistency of created parts of *G*.

The MKS integrates concepts of *G* as instances of metaconcepts of *M* in the current partial construction, which represents *G* at the metalevel. Changes in already defined concepts are represented by backtracking on the metalevel model-construction process. For example, first a parameter of a concept is some how modeled, e.g. a value for that parameter is modeled. This is constructed in the metalevel model-construction process as a configuration step. If later this parameter value is changed, the MKS performs a backtracking step to the previously performed configuration step for that parameter value. Thus, the previous parameter value is taken back and the new parameter value is configured. By using already developed backtracking approaches for knowledge-based configuration, especially dependency-based backtracking (see (Hotz et al., 2004; Ferber et al., 2002)), dependencies of modeling decisions can be automatically managed. Thus, changes of a CDL configuration model (i.e. during the knowledge-acquisition process or during evolution (Männistö and Sulonen, 1999)) are basically changes of *G*, i.e. changes of the currently constructed model on the metalevel. In other word, *Evolution is backtracking on the metalevel.*

Besides this construction of CDL knowledge bases the MKS can scrutinize CDL instances, which are created during a model-construction process. For this task, MKS is supplied with such instances (see Figure 9) and creates instances of the metaconcept *instance-mm*. By doing so consistency rules for instances represented as conceptual constraints on the metalevel can be checked.

Looking from the MKS perspective the construction of a CDL knowledge base can be seen as the interpretation of an external system similar to the interpretation of an outside scene. MKS observes the construction of the CDL knowledge base and tries to integrate the observations by using the Meta-CDL-KB. This task is similar to scene interpretation where evidence in a scene is interpreted by constructing an interpretation on the basis of a model for anticipated scenes. Thus, similar implementations can be applied for the MKS like top-down and bottom-up structuring, spontaneous instantiation, and merging (see also (Hotz and Neumann, 2005; Hotz, 2006)).

We implemented parts of the meta configurator with the configuration system KONWERK (Günter and Hotz, 1999). The Meta-CDL-KB could be used for constructing knowledge bases for a PC-domain. However, first experiments demand the need of highly interactive facilities for visualizing the complex relational structures of meta-level instances, e.g. visualizing which `some-mm` instance belongs to the which `concept-mm` during the configuration process.

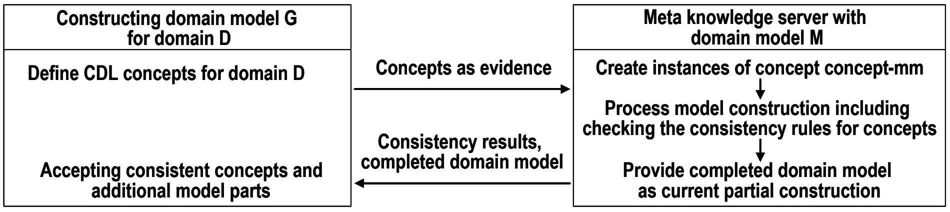
Constructing domain model G for domain D
Define CDL concepts for domain D
Accepting consistent concepts and additional model parts
Concepts as evidence
Consistency results, completed domain model
Meta knowledge server with domain model M
Create instances of concept concept-mm
Process model construction including checking the consistency rules for concepts
Provide completed domain model as current partial construction

Figure 8: Meta-knowledge server applied to constructing a configuration model.

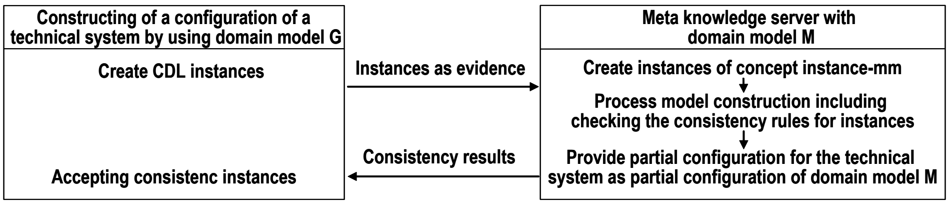
Constructing of a configuration of a technical system by using domain model G
Create CDL instances
Accepting consistenc instances
Instances as evidence
Consistency results
Meta knowledge server with domain model M
Create instances of concept instance-mm
Process model construction including checking the consistency rules for instances
Provide partial configuration for the technical system as partial configuration of domain model M

Figure 9: Meta-knowledge server applied to constructing a configuration.

4 Discussion

The model-construction view as it is emphasized in this work is a systematic generalization of structure-oriented configuration like it is provided by (Günter, 1995; Soininen et al., 1998) and others. This is mainly achieved by focusing on the structural relation, which ensures existence of instances in the resulting construction. These instances build the constructed model. This model is a description of the desired technical system, which is used e.g. for the production process. In this sense, also other configuration approaches like connection-based (Mittal and Frayman, 1989), resource-based (Heinrich and Jüngst, 1991), or function-based (Najman and Stein, 1992) can be seen as model-construction approaches. This view to configuration enables the concise application of configuration tools in environments like services, software, or like in this paper on metalevels. However, for model construction seldom supported facilities are needed as there are:

- The representation and processing of cyclic relational structures. Those techniques are sometimes avoided like in [Magro et al., 2002; Arlt et al., 1999].
- Sophisticated control mechanisms like bottom-up and top-down construction. Typically only top-down is emphasized in configuration systems.
- Connecting model construction with other external systems or the real world *during* the model-construction process demands spontaneous instantiation of concepts. In configuration systems only for creating the knowledge base external data like databases are used and the resulting configuration is exported for producing the configured system.

The creation of a metamodel for CDL with the aid of CDL has its tradition in self-referencing approaches like Lisp-in-Lisp (Brooks et al., 1983) or the metaobject protocol, which implements CLOS (the Common Lisp Object System) with CLOS (Kiczales et al., 1991). Such approaches demonstrated the use of the respective language. In case of CDL the metaknowledge server is enabled. It makes strong use of the implemented inference techniques of CDL like constraint propagation.

The meta-knowledge server is basically an implementation of a configuration tool on the basis of the Meta-CDL-KB, i.e. of a configuration model. A typical configuration tool is implemented with a programming language and an object model implemented with it. During this implementation one has to ensure correct behavior of model construction and the inference techniques. By using CDL this behavior (e.g. the consistency rules) is declaratively modeled, not implemented. The bases for this realization are of course the implementation of value-computation methods and constraint mechanisms.

The here introduced meta-knowledge base has some relations to metamodeling approaches like described in (Hesse, 2006; Kühne, 2006; OMG, 2006;). Thus, in the following, we take a first glance to some aspects of metamodeling (see also (Asikainen and Männistö, 2009) for a deeper analysis of metamodeling). The main task of metamodeling is to specify modeling facilities that can be used for defining models, see for example (OMG, 2007): "A metamodel is a model that defines the language for expressing a model". Or compiled to terms used here: "The Meta-CDL-KB is a configuration model that defines CDL, which in turn is used for expressing a configuration model" (see Figure 10). However, the notion of modeling is still not finally fixed (see (Hesse, 2006; Kühne, 2006)), or as (Hesse, 2006) says: "A complete and unanimously accepted theory of modeling is still emerging.". Besides these theoretical issues, in our approach a more pragmatical and operational view is taken, i.e. how to apply a metamodel for supporting the use of the language the metamodel defines. From this perspective, let us examine the Requirements Specification Language RSL (Kaindl et al., 2007; Smiałek et al., 2007; Hotz et al., 2009). A metamodel defines elements typically used for specifying requirements as their are use-cases, scenarios etc. A tool (*RSL-Tool*) enables a requirements engineer to express her use-cases etc. through a user interface and the tool constructs a requirements specification expressed in RSL. Thus, the metamodel of RSL is used by the implementor of the RSL-Tool, which in turn ensures a requirements specification that is compliant to the metamodel of RSL (see Figure 10). However, the implementation is done manually and specific for the RSL-metamodel. The creation of metamodel compliant models can be supported by a configuration tool.

A configuration tool supplies mainly three tasks:

1. It enables the expression of a configuration model that is consistent with the configuration language, which the tool implements. For this task, it performs consistency checks of given configuration models (or parts of it) with the language specification.
2. On the basis of the configuration model, the configuration tool supports the creation of constructions that are consistent with the configuration model. For this task, the tool interprets the logical expressions of the configuration model and creates constructions according to these definitions.
3. The configuration tool supplies user interfaces for expressing the configuration model and for guiding the construction process. The configuration model can be typically given in textual forms or with graphical user interfaces that enable the creation of concepts and constraints.

Thus, a configuration tool contains means for supporting the step from a domain model to a system specific model (see Figure 10). By introducing configuration models in the model chain as presented in Figure 10, an additional level is introduced, i.e. the domain-model level. This level represents all systems of a domain. The model for a system is an instantiation of the domain model. This instantiation is computed by a configuration tool. In our metamodeling approach based on the Meta-CDL-KB this instantiation facility is used for supporting the step from the configuration language to the domain model, i.e. the domain-representation phase. By applying the configuration tool to a domain model that contains every model of a language, i.e. by applying it to the Meta-CDL-KB, the construction of a domain model of an arbitrary domain is supported. This is achieved because of the general applicability of the language constructs of CDL, which are based on logic (see Section 2). Furthermore, other advantages of configuration tools, like a declarative representation of the configuration model, or the use of the inference techniques can thus applied to the Meta-CDL-KB.

A similar approach as supplied by the meta-knowledge server is provided by (Kienzler, 2000) who uses meta planning. A primary construction process is supported by a secondary analysation process on the metalevel. The configuration process is controlled by a meta planner. The meta planner is strongly coupled with the configuration process. However, it is realized by a further external implementation not in the configuration language itself.

Other approaches like (Dietrich et al., 2004) also use a metamodel approach for supporting the configuration process. However, by using a configuration language for expressing the metamodel, in our approach a configuration tool can directly applied for making use of the metalevel.

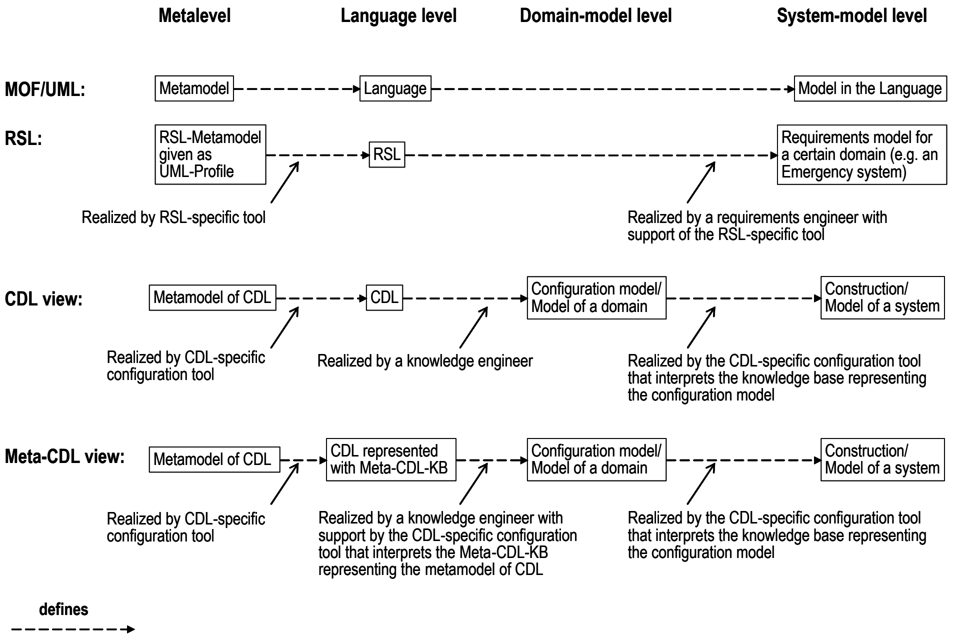
Metalevel
Language level
Domain-model level
System-model level
MOF/UML:
Metamodel
Language
Model in the Language
RSL:
RSL-Metamodel given as UML-Profile
RSL
Requirements model for a certain domain (e.g. an Emergency system)
Realized by RSL-specific tool
Realized by a requirements engineer with support of the RSL-specific tool
CDL view:
Metamodel of CDL
CDL
Configuration model/ Model of a domain
Construction/ Model of a system
Realized by CDL-specific configuration tool
Realized by a knowledge engineer
Realized by the CDL-specific configuration tool that interprets the knowledge base representing the configuration model
Meta-CDL view:
Metamodel of CDL
CDL represented with Meta-CDL-KB
Configuration model/ Model of a domain
Construction/ Model of a system
Realized by CDL-specific configuration tool
Realized by a knowledge engineer with support by the CDL-specific configuration tool that interprets the Meta-CDL-KB representing the metamodel of CDL
Realized by the CDL-specific configuration tool that interprets the knowledge base representing the configuration model
defines

Figure 10: Relations between metamodels and domain models and their tool support.

5 Summary

The paper shows how a configuration language can be expressed with its own representation facilities. Thus, the parts that are composed in such a case are the modeling facilities the configuration language supplies, i.e. concepts, parameters, constraints etc. The configuration model contains concepts, parameters, constraints that again represent concepts, parameters etc. By doing so, inference techniques that are provided by the language can be used for constructing configuration models and thus, support the knowledge-acquisition process. In this case, the configuration tool is mainly used for checking the consistency of the constructed configuration models. Thus, the use of the inference techniques supports the formal basis of such processes. Further work will emphasize user-interface tools that support the visualization and manipulation of highly structured relationships including cyclic structures that occur on the metalevel.

References

Arlt V, Günter A, Hollmann O, Wagner T, and Hotz l (1999). EngCon - Engineering & Configuration. In Proc. of AAAI-99 Workshop on Configuration, Orlando, Florida

Asikainen T and Männistö T (2009). A metamodelling approach to configuration knowledge representation. In Proc. of the Configuration Workshop on 22th European Conference on Artificial Intelligence (IJCAI-2009), Pasadena, California

Brooks RA, Gabriel RP, and Steele L (1983) Jr. Lisp-in-Lisp: High Performance and Portability. In Proc. of Fifth Int. Joint Conf. on AI IJCAI-83

Buchheit M., Klein R, and Nutt.W (1995) Constructive Problem Solving: A Model Construction Approach towards Configuration. Technical Report TM-95-01, Deutsches Forschungszentrum für Künstliche Intelligenz, Saarbrücken

Cunis R, Günter A, and Strecker H(Hrsg.) (1991) Das PLAKON-Buch. Springer Verlag Berlin Heidelberg

Dietrich AJ, Hümmer W, and Meiler C (2004) Meta model based Configuration Approach for mass-customizable Products and Services. In Proceedings of the 4thWorkshop on Information Systems for Mass Customization (ISMC 2004), Madeira Island, Portugal

A. Felfernig, G. Friedrich, D. Jannach, M. Stumptner, and M. Zanker. A Joint foundation for Configuration in the Semantic Web. In Proc. of the Configuration Workshop on 15th European Conference on Artificial Intelligence (ECAI-2002), pages 89–94, Lyon, France, July 21-26 2002.

Ferber A, Haag J, and Savolainen J (2002) Feature Interaction and Dependencies: Modeling Features for Re-engineering a Legacy Product Line. In Proc. of 2nd Software Product Line Conference (SPLC-2), Lecture Notes in Computer Science, pages 235–256, San Diego, CA, USA

Günter A and Hotz L.(1999) KONWERK - A Domain Independent Configuration Tool. Configuration Papers from the AAAI Workshop, pages 10–19

Günter A. (1995) Wissensbasiertes Konfigurieren. Infix, St. Augustin

Heinrich M and Jüngst E (1991). A Resource-based Paradigm for the Configuring of Technical Systems from Modular Components. In Proc. of 7th IEEE Conf. on Artificial Intelligence for Applications (CAIA'91), pages 257–264, Miami Beach, Florida, USA

Hesse W (2006). More matters on (meta-)modelling: remarks on Thomas Kühne's matters". Journal on Software and Systems Modeling, 5(4):369–385

Hotz L and Neumann B. (2005) Scene Interpretation as a Configuration Task. Künstliche Intelligenz, 3:59–65

Hotz L, Krebs T, and Wolter K.(2004) Dependency Analysis and its Use for Evolution Task. In 18th Workshop, New Results in Planning, Scheduling and Design (PuK2004). University of Oldenburg

Hotz L, Wolter K, Krebs T, Deelstra S, Sinnema M, Nijhuis J, and MacGregor J.(2006) Configuration in Industrial Product Families - The ConIPF Methodology. IOS Press, Berlin

Hotz L, Wolter K, Knab S, and Solth A. (2009) Ontology-based Similarity of Software Cases. International Conference on Knowledge Engineering and Ontology Development, KEOD, Madeira

Hotz L (2006). Configuring from Observed Parts. In C. Sinz and A. Haag, editors, Configuration Workshop, 2006,Workshop Proceedings ECAI, Riva del Garda

Hotz L (2008) Modeling, Representing, and Configuring Restricted Part-Whole Relations. In J. Tiihonen, editor, Configuration Workshop, 2008, Workshop Proceedings ECAI, Patras

Hotz L (2009). Frame-based Knowledge Representation for Configuration, Analysis, and Diagnoses of technical Systems (in German), volume 325 of DISKI. Infix

Kaindl H, Smiałek M, Svetinovic D, Ambroziewicz A, Bojarski J, Nowakowski W, Straszak T, Schwarz H, Bildhauer D, Brogan JP, Ssamula Mukasa K, Wolter K, and Krebs T (2007). Requirements specification language definition. Project Deliverable D2.4.1, ReDSeeDS Project

Kiczales G, Rivieres J des, and Bobrow DG (1991). The Art of the Metaobject Protocol. The MIT Press, Cambridge, MA, CA

Kienzler.F (2000) Synthesis versus Analysis in model-based AI-Planning Systems? PIAKON - a autoadaptive diagnostic Solution Approach for Action Planing and Configuration Problems (in German). PhD thesis, University of Ulm

Kühne T (2006). Matters of (Meta-)Modeling. Journal on Software and Systems Modeling, 5(4):369–385

Magro D, Torasso P, and Anselma L (2002). Problem Decomposition in Configuration. In Configuration Workshop, 2002,Workshop Proceedings ECAI, Lyon, France

Männistö T and Sulonen R. (1999) Evolution of Schema and Individuals of Configurable Products. In Proc. of ECDM'99 - Workshop on Evolution and Change in Data Management, Versailles, France

Marcus S, Stout J, and McDermott J.(1988) VT: An Expert Elevator Designer that uses Knowledgebased Backtracking. AI Magazine, pages 95–112

Mittal S and Frayman F (1989). Towards a Generic Model of Configuration Tasks. In Proc. of Eleventh Int. Joint Conf. on AI IJCAI-89, pages 1395–1401, Detroit, Michigan, USA

Najman O and Stein B (1992). A Theoretical Framework for Configurations. In Proc. of Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: 5th International Conference, IEA/AIE-92, pages 441–450

OMG.(2006) Meta Object Facility Core Specification, version 2.0, formal/2006-01-01. Object Management Group

OMG (2007). Unified Modeling Language: Infrastructure, version 2.1.1, formal/07-02-06. Object Management Group

Ranze C, Scholz T, Wagner T, Günter A, Herzog O, Hollmann O, Schlieder C, and Arlt V (2002). A Structure-based Configuration Tool: Drive Solution Designer - DSD. In Eighteenth national conference on Artificial intelligence, pages 845–852, Menlo Park, CA, USA, American Association for Artificial Intelligence.

Smiałek M, Bojarski J, Nowakowski W, Ambroziewicz A, and Straszak T. (2007) Complementary use case scenario representations based on domain vocabularies. Lecture Notes in Computer Science, 4735:544–558, 2007.

Soininen T, Tiihonen J, Männistö T, and Sulonen R (1998). Towards a General Ontology of Configuration. Artificial Intelligence for Engineering Design, Analysis and Manufacturing (1998), 12, pages 357–372

Stefik M (1995) Introduction to Knowledge Systems. Morgan Kaufmann, San Francisco, CA

Stumptner. M (1997) An Overview of Knowledge-based Configuration. AI Communications, 10(2):111–126

Tiihonen J, Heiskala M, Paloheimo K-S, and Anderson A (2006).Configuration of Contract Based Services. In C. Sinz and A. Haag, editors, Configuration Workshop, 2006, Workshop Proceedings ECAI, Riva del Garda