THOUGHTS ABOUT STRUCTURALIZATION, SPECIALIZATION, INSTANTIATION, AND METAIZATION

Lothar Hotz and Stephanie von Riegen
Hamburger Informatik Technology Center, Department Informatik, University of Hamburg, Hamburg, Germany

Keywords: Metamodeling, Knowledge representation.

Abstract: In knowledge engineering, ontology creation, and especially in knowledge-based configuration often used relations are: aggregate relations (`has-parts`, here called structural relations), specialization relation (`is-a`), and instantiation (`instance-of`). A combination of the later is called metaization, which denotes the use of multiple instantiation layers. Structural and specialization relations are mainly used for organizing the knowledge represented on one layer. Instantiation layers model different kind of knowledge, i.e. knowledge about sets, individuals, and knowledge about knowledge (metaknowledge). By applying reasoning techniques on each layer, reasoning on metaknowledge is enabled.

1 INTRODUCTION

For configuration-based inference tasks, like constructing a description of a specific car periphery system (Hotz et al., 2006) or drive systems (Ranze et al., 2002), the knowledge of a certain domain is represented with a *knowledge-modeling language* which again is interpreted, because of a defined semantic, through a *knowledge-based system* or *configurator*. Examples for knowledge-modeling languages are the Web-Ontology Language (OWL) or the Component Description Language (CDL) (Hotz, 2009). Further languages are e.g. described in (Harmelen et al., 2007). Such languages typically provide *concepts* or *classes* that gather all properties, a certain set of domain objects has, under a unique name. With concepts and instances a strict separation into two layers is made: a *domain model* (or *ontology*) which covers the knowledge of a certain domain (abbr. $layer^D$) and a *system model* (or *configuration*) which covers the knowledge of a concrete system or product of the domain (abbr. $layer^S$).

Properties of a concept that map to primitive data types, like intervals, values sets (enumerations), or constant values, are called *parameters* or *attributes*. Properties that map to other concepts or to instances are called *relations*. Knowledge-modeling languages provide structural, specialization, and instantiation as typical relations. A specialization relation relates a *superconcept* to a *subconcept*, where the later inherits the properties of the first. This relation (also called `is-a` relation) forms a *specialization hierarchy* or *lattice*, if a concept has more than one superconcept. 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 other structural relationships. *Instances* are instantiations of concepts and represent concrete domain objects (`instance-of`).

Additionally to concepts, instances, and their relations, constraints provide model facilities to express n-ary relationships between properties of concepts. Constraints can represent restrictions between properties like arithmetic relations or restrictions on structural relations (e.g. ensuring existence of certain instances).

In this paper, the use of structuralization, specialization, and instantiation is discussed. Even those relations are quite well-known they are sometimes confounded. Furthermore, when used with more than the two mentioned domain and system layers (see (Asikainen and Männistö, 2009; Hotz, 2009)) the instantiation relation is multiply applied, which leads to new modeling layers and thus probably to modeling difficulties. The creation of such multiple layers is called *metaization* (Strahringer, 1998).

In the following, we first consider all relations in more depth and give example of their use (Section 2 and Section 3). Afterwards, we discuss metaization and its use for configuration (Section 4). We end with a short discussion on related work and a conclusion.

Hotz L. and von Riegen S..
THOUGHTS ABOUT STRUCTURALIZATION, SPECIALIZATION, INSTANTIATION, AND METAIZATION.
DOI: 10.5220/0003666504570460
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development* (KEOD-2011), pages 457-460
ISBN: 978-989-8425-80-5

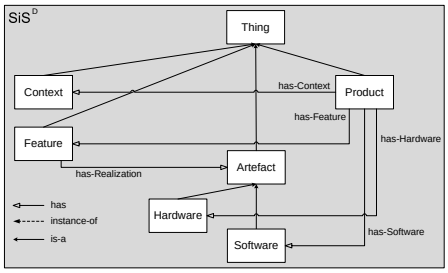

Figure 1: Extract from an upper-model for modeling software-intensive systems.

2 STRUCTURALIZATION

As already elaborated in (Hotz, 2009) configuration can be considered as model construction, because a description of a certain system (a configuration) is constructed by a configurator. Furthermore, (Hotz, 2009) emphasizes to consider the `has-parts` relation as a `has` relation that may be used for diverse aspects like `has-Realizations` or `has-Features` in software-intensive systems. For the typical use, a structural relation represents a compositional relation. In this case, between c and its relatives r, c denotes the aggregates and r denotes the parts. The underlying structural relation is used by configurators to construct the description and thus are the motor of configuration. Depending on what instances (of c or r) exist first, instances of the related concepts are created; e.g. this enables reasoning from the aggregate to the parts or contrariwise, from the parts to the aggregate. This semantic holds for every structural relation. Thus, introducing several structural relations enables the use of adequate domain names like `has-Features` or `has-Realizations`, and thus to facilitate maintenance.

Figure 1 pictures an *upper-model for software-intensive systems* (UMSiS, (Hotz et al., 2006)). It defines four asset types (features, context, hardware and software artefacts) which are common to most application domains of software-intensive systems. A product, i.e. the result of the product derivation, contains software and hardware artefacts as parts, these together realize particular features. Several structural relations are depicted, like `has-Realizations` and `has-Feature`. When using the upper-model for a specific domain, the UMSiS is extended with domain-specific knowledge about hardware and software artefacts, the existing features, relevant context aspects, etc. In the example above, the concepts are organized in different *spaces*. Each space represents a specific aspect of the domain and thus each configured product should have those aspects. Figure 1 provides the example of the feature and artefact aspects in the domain of software-intensive systems. Thus, *spaces* separate concepts of one layer. Through this grouping of concepts of one layer the configuration model is easier to manage for a knowledge engineer. Furthermore, concepts of different spaces are connected by a structural relation. This ensures that a configured product finally contains all modeled aspects. In contrast to this, in Section 3 we will see, how the instantiation relation separates concepts and instances on different *layers*.

3 SPECIALIZATION VS. INSTANTIATION

A concept describes a set of instances. The specialization relation (or subsumption or `is-a` relation) between two concepts c and s describes a subset relation, i.e. the set of instances of concept c is a subset of the set of instances of its superconcept s (see also (Brachman, 1983)). Or, as defined in `ontogenesis.knowledgeblog.org/699`: "c is-a s if and only if: given any i that instantiates c, i instantiates s". An instance of a class c is always an instance of each superclass s of c. We consider this aspect as the hinting characteristic for knowledge engineers: During knowledge modeling one can try to make a specialization between two domain aspects and test this characteristic. Thus, it is tested if an instance of c is also reasonably an instance of s. If it is false the knowledge engineer must not use a specialization but e.g. instantiation, because c and s are probably on different layers.

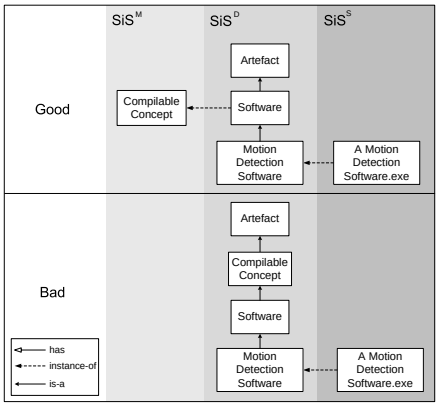

Figure 2: Good and bad use of specialization and instantiation in software-intensive systems.

An example for this situation is shown in Fig-

ure 2; it presents the confounded usage of specialization and instantiation relations in the aforesaid modeling of software-intensive systems domain (SiS). The system model layer (SiS^S) is covering specific individuals, here for instance the `A Motion Detection Software.exe`. This object is an instance of the `Motion Detection Software` (SiS^D) but no instance of `Compilable Concept`. `Compilable Concept` denotes a specific kind of concept. A concept typically specifies the description of the property structure of its instances. A `Compilable Concept` additionally can take this description and compile it to an executable file. Thus, in the "bad" use, `A Motion Detection Software.exe` is incorrectly considered as a concept, i.e. as a *description* of instances that can be compiled. Instead it is an *instance* of `Motion Detection Software`, thus a specific domain object not a concept and, of course, it is already compiled.

When a concept s is specialized to c all properties of s are inherited by c. By the time a concept is instantiated, properties of the created instance are initialized by values or value ranges specified in the concept. Thus, the concept determines the structure of the instance (i.e. the properties). In this sense, a concept says something about its instances, i.e. a concept is on a different layer than its instances. By reducing the value ranges according to user decisions or constraint computations the configurator subsequently creates a specific description consisting of instances, i.e. the configuration.

4 METAIZATION

For structuralization and specialization, the involved concepts are on one layer. However, for instantiation and metaization they are on different layers. By instantiating a concept one instance is created, i.e. a step from a set of instances to an individual element of this set is performed. If this step is cascadized, a concept c can be considered as an instance of another concept c_m, i.e. a step from a set of concepts to one specific concept is performed. The concept c_m is on a further layer. Figure 3 demonstrates this situation. The concept `Feature` is an instance of `Abstract Concept` which is a specialization of `concept-m`. The concepts on the metalayer CDL^M represent the modeling facilities of CDL, describing the concepts and relations of CDL. Concept `Artefact` is a typical CDL concept (it is an instance of `concept-m`). Beside concepts, also relations have a concept on CDL^M for representing them (not depicted in the Figure, see (Hotz, 2009)). Thus, CDL^M represents all what is known about CDL^D, i.e. concepts and relations.

Figure 3 presents the enhancement of Figure 1 by the additional layer SiS^M. SiS^M describes the SiS^D layer concepts `Feature`, `Software`, and `Hardware` as `Abstract Concept`, `Compilable Concept`, and `Manufacturable Concept`, respectively. Thus, it is a domain dependent extensions of CDL^M.

By doing so, constraints on concepts of SiS^D can be expressed. For example, a constraint represents that each feature should be realizable by an artefact. Such a constraint can check that each feature (a subconcept of `Feature`) should have a structural relation `has-Realization` to a subconcept of `Artefact`. These kinds of constraints may be hard to define, because typically they are not related to one specific concept but to several. Still, such constraints are usually part of some knowledge modeling guidelines.

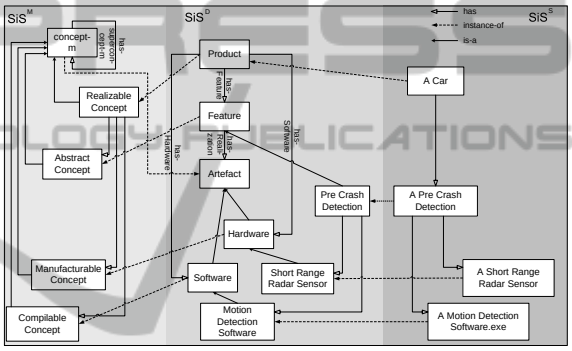

Figure 3: Modeling software-intensive systems.

In (Hotz and von Riegen, 2010), we introduce the Reasoning Driven Architecture (RDA) that allows the implementation of metalayers by using a configuration system on each layer. By doing so, each layer can be seen as a knowledge-based system that says something about the layer below. In the case of RDA, SiS^D contains the knowledge of domain objects, which again are represented on SiS^S. By introducing the metalayer SiS^M, knowledge about knowledge is made explicit, i.e. knowledge about the knowledge of domain objects. This enables the use of reasoning techniques for each layer, not only for the domain and system layers as it is typically the case in knowledge-based systems. The central point of such an implementation is a mapping between instances on one layer to concepts on the next lower layer (see (Hotz and von Riegen, 2010) for a mapping for CDL and (Tran et al., 2008) for a mapping for OWL or (Bateman et al., 2009)). Metalayers allow for handling (meta) tasks and services. For example, (Tran et al., 2008) proposes to provide statistics about the model (e.g. retrieve all knowledge elements about Pre Crash Detection). With a metalayer

like provided in Figure 3, during configuration of a software-intensive system one can call different external mechanisms for each specific metaconcept. For example, if an instance of an instance of `Compilable Concept` (e.g. an instance of `Software`) is configured, an external compiler mechanism can be called to realize the software. If an instance of an instance of `Manufacturable Concept` is configured, the warehouse can be contacted to check if the needed parts for the manufacturing are present. Thus, through the metalayer the actual configuration of a product can be monitored and reasoning on the configuration process can be processed.

5 RELATED WORK

The modeling approach, especially metaization (Strahringer, 1998), has similarities to the Model-Driven Architecture (Kühne, 2006; Atkinson and Kühne, 2003; Hotz and von Riegen, 2010), because of the explicitation of several layers. However, the introduction of reasoning systems for each layer allows the direct usage of existing reasoners for inferring on metalayers.

(Asikainen and Männistö, 2009) and (Haase et al., 2009) present also approaches that include semantics on the metalayer, similar to our approach. By doing so, reasoning methods on each layer as well as the capability to define domain-specific extensions on the metalayer is in principle enabled. Metaization as such is less considered in knowledge-based configuration. However, especially when learning methods, i.e. automated knowledge engineering, has to be used in changing environments, the automated monitoring of knowledge bases becomes crucial and is conceivable with the presented techniques.

6 CONCLUSIONS

In this paper, we state the differences of the main relations for modeling configuration knowledge, i.e. specialization, instantiation, and structuralization. By introducing and clarifying the use of instantiation on several metalayers, we open up a further modeling facility and sketch first usage of this metaization technique for knowledge-based configuration. In upcoming work, we will apply these techniques in learning environments in the field of robot vision.

REFERENCES

Asikainen, T. and Männistö, T. (2009). Nivel: a metamodelling language with a formal semantics. *Software and Systems Modeling*.

Atkinson, C. and Kühne, T. (2003). Model-Driven Development: A Metamodeling Foundation. *IEEE Softw.*, 20(5):36–41.

Bateman, J., Castro, A., Normann, I., Pera, O., Garcia, L., and Villaveces, J. (2009). OASIS common hyper-ontological framework (COF), Deliverable D1.2.1. Technical report, University of Bremen.

Brachman, R. J. (1983). What is-a is and isn't: An analysis of taxonomic links in semantic networks. *IEEE Computer*, 16(10):30–36.

Haase, P., Palma, R., and d'Aquin M. (2009). Updated Version of the Networked Ontology Model. Project Deliverable D1.1.5, Neon Project. www.neon-project.org.

Harmelen, F. V., Lifschitz, V., and Porter, B., editors (2007). *Handbook of Knowledge Representation (Foundations of Artificial Intelligence)*. Elsevier Science.

Hotz, L. (2009). Construction of Configuration Models. In Stumptner, M. and Albert, P., editors, *Configuration Workshop, 2009*, Workshop Proceedings IJCAI, Pasadena.

Hotz, L. and von Riegen, S. (2010). Knowledge-based Implementation of Metalayers - The Reasoning-Driven Architecture. In Felfernig, A. and Wotawa, F., editors, *Proceedings of the ECAI 2010 Workshop on Intelligent Engineering Techniques for Knowledge Bases (IKBET)*.

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.

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

Ranze, K., 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. *14. Conf. Innovative Applications of AI*.

Strahringer, S. (1998). Ein sprachbasierter Metamodellbegriff und seine Verallgemeinerung durch das Konzept des Metaisierungsprinzips. In *Proceedings of the Modellierung 1998*. Astronomical Society of Australia.

Tran, T., Haase, P., Motik, B., Grau, B. C., and Horrocks, I. (2008). Metalevel Information in Ontology-Based Applications. In Fox, D. and Gomes, C. P., editors, *Proc. of the 23rd AAAI Conf. on Artificial Intelligence (AAAI 2008)*, pages 1237–1242, Chicago, IL, USA. AAAI Press.