Towards a domain-specific language for knowledge maintenance of CBR systems

Pascal Reuss[1,2], Wasgen Muradian[2] and Klaus-Dieter Althoff[1,2]

[1]German Research Center for Artificial Intelligence, Kaiserslautern, Germany
[2]Institute of Computer Science, Intelligent Information Systems Lab, University of Hildesheim, Hildesheim, Germany

Abstract

Maintaining CBR systems can be a challenging task depending on the complexity of the domain and use case of the CBR system, the knowledge to maintain, and the applicable approaches. Many maintenance policies and strategies were developed during the last decades, but mostly for specific individual CBR systems and not broadly used. This paper describes a first step towards a domain-specific language for knowledge maintenance of CBR systems that could be used in a maintenance framework to bundle the various existing maintenance policies and strategies and give a knowledge engineer a tool to maintain CBR systems more efficient and effectively.

Keywords

Case-Based Reasoning, Knowledge Maintenance, Domain Specific Language,

1. Introduction

In order to ensure the competence and efficiency of a Case-based Reasoning (CBR) system, the knowledge of the system should be maintained at regular intervals or at specific events, such as the detection of performance losses. This kind of maintenance is referred to as knowledge maintenance in CBR systems and becomes particularly relevant when there are continuous changes in the application environment of the CBR system. The importance of knowledge maintenance in CBR systems has increased in recent years as it has been demonstrated that the handling of knowledge can have a significant impact on the performance, competence and quality of the system. [1]

One way to design and implement maintenance actions in CBR systems is to use a General Purpose Language (GPL) like Java or C++. While GPLs are easily accessible, the knowledge maintenance modeling process is complicated by the use of a GPL because it is domain-independent and therefore does not provide specific maintenance design functions and operators. A potential alternative to a GPL is the use of a Domain-specific Language (DSL) to model maintenance measures for CBR systems. A DSL would be tailored to the field of knowledge maintenance, providing the user with functions, terms, and operators that have been specifically developed for modelling and carrying out maintenance actions. From this point of view, a DSL makes it possible to model complex issues of knowledge maintenance in CBR systems in a simple way, providing a transparency regarding the maintenance objectives and the maintenance process.

LWDA'21: Lernen, Wissen, Daten, Analysen September 01–03, 2021, Munich, Germany
pascal.reuss@dfki.de (P. Reuss); klaus-dieter.althoff@dfki.de (K. Althoff)

In addition, such a DSL could be used in a maintenance framework, like a maintenance cockpit, to provide a knowledge engineer with multiple of maintenance methods ready to be applied to CBR systems.

In Section 2 the maintenance cockpit approach is described briefly to give an overview of the intended meta maintenance approach, while Section 3 describes the ongoing development of a DSL for knowledge maintenance of CBR systems. This section is divided into two subsections, with Section 3.1 giving an overview of the conducted domain analysis and Section 3.2 describes the current design state and language elements. The paper ends with a conclusion and an outline on the future development of the DSL.

2. Case Factories: A maintenance cockpit for CBR systems

The Case Factory (CF) approach is a cockpit for maintenance of multi-agent systems (MAS) with several distributed structural CBR systems. This cockpit is able to monitor the changes in the knowledge containers, evaluate the quality of the knowledge containers, and suggest maintenance actions if necessary. The suggested maintenance actions are passed to a knowledge engineer to confirm or refuse the maintenance actions. As a basis for the new approach, the SEASALT architecture [2] was chosen. This architecture can be used to build MAS with a so-called knowledge line, which is responsible for providing the required knowledge for solving a given problem. Each knowledge source within the knowledge line is managed by a topic agent, which has access to a structural CBR system and each CBR system has its own CF to maintain its knowledge. The original idea of the CF was introduced in 2006 by Althoff et al. [3] and applied in theory to the case bases of the travel medicine application docQuery [4]. The original CF has several shortcomings, especially when applying it to distributed CBR systems. It does not consider explicit dependencies between the knowledge containers of a CBR system, especially more than the case base, or between different CBR systems. Therefore, the original CF approach was extended to be useful for the maintenance cockpit when maintaining distributed CBR systems.

The central idea of the maintenance cockpit approach is that dependencies exist between knowledge containers in a single CBR system and between the knowledge containers of different CBR systems [5]. These dependencies should be considered when planning the maintenance of the single CBR systems and the overall MAS with several distributed CBR systems. Because the SEASALT architecture is designed for multi-agent systems and the new maintenance approach should be integrated into the architecture, the CFs and the organizational superstructure called Case Factory Organization (CFO) are designed as multi-agent systems as well. This way, the new multi-agent systems can be integrated as sub-societies into the overall multi-agent society of the SEASALT architecture. The dependencies allow an overall maintenance planning with respect to connections between the individual knowledge of different CBR systems. A dependency can be defined with a source, a target, and a direction [6]. To define the source and the target of a dependency, the knowledge of CBR systems is organized in a hierarchy with six knowledge levels[7]. Each subsequent knowledge level contains knowledge on a more detailed level than the one before. An overview of the defined knowledge level can be found in Table 1.

Based on this hierarchy, dependencies with different knowledge levels as source and target

Knowledge level	Contained knowledge	Example
Knowledge level 1	CBR system	CBR system 1
Knowledge level 2	knowledge container	vocabulary, case base
Knowledge level 3	specific case base	CB01, CB02
Knowledge level 4	specific case, similarity measure, and adaptation rules	case 123, simtax, rule23
Knowledge level 5	attributes, condition side, consequence side	aircraft type, systems, status, "'if status = inop"', "'then code = W3566"'
Knowledge level 6	specific attribute values, similarity values, weights, condition, consequence	A380, display, inoperable, 0.5, "'status = inoperable"'

Table 1
Knowledge levels, the contained knowledge, and examples

can be defined. The most abstract dependencies are based on knowledge level 1 and the most detailed dependencies are based on knowledge level 6. For example, on knowledge level 1 a dependency between a CBR system A and a CBR system B could be defined. The dependency does not contain enough knowledge to derive a specific maintenance action, but could be used for visualization purposes of dependencies. On knowledge level 6, a dependency between two specific values could be defined. For example, there could be a dependency between the value *A380* of the attribute *aircraft type* in the knowledge container *vocabulary* of *CBR system A* and the value *A380* of the attribute *aircraft type* of the *case123* in the case base *CB01* in the knowledge container *case bases* of *CBR system A*.

Beside the knowledge items and dependencies, there are several other information relevant for knowledge maintenance of distributed CBR systems: maintenance actions and transactions, monitoring and evaluation methods as well as their results, maintenance goals, strategies, plans and explanations. Not all information is required all the time, but different combinations are required for every maintenance approach. The idea of the maintenance cockpit is to provide a framework of software agents that can do knowledge maintenance of distributed CBR systems for different domains and use cases. Therefore, the cockpit has to deal with all this information even if only parts are used in specific maintenance use cases. While the maintenance cockpit is no new maintenance approach itself, it is a meta approach that can incorporate existing maintenance approaches.

3. A DSL for maintaining CBR systems

A DSL can help to create a common vocabulary for the existing different maintenance approaches and allow a meta approach like the maintenance cockpit to use the DSL to ease knowledge maintenance. By using specific notations of the knowledge maintenance domain, a DSL will have a higher expressiveness and enables a more efficient realization of maintenance policies for CBR systems than using a common GPL [8]. In addition, by using a high quality DSL within a meta maintenance approach, the quality of analysis, validation, optimization, and transformation of the domain-specific entities will be increased as well as the portability, reliability, and test

efficiency of the application [9][10]. which is in this case the maintenance cockpit.

Currently, there is no explicit DSL in the area of CBR. While there is a common language that is used to desribe the CBR cycle and the knowledge containers, the realization of CBR systems can and was done in several ways and several GPLs. Especially for maintenance purposes, different individual approaches exist (as can be seen in Section 3.1) and were implemented in different use cases. Therefore, the overall goal of the desired DSL would be to create an tool to have a more generic approach for designing and implementing knowledge maintenance for CBR systems.

Developing a DSL should be done in a systematic way to avoid faults and defects. This systematic development is a step-wise process to identify, collect, and classify relevant information of the target domain and transform this information to define the required language elements. The systematic development and application process defined by Mernik contains seven phases to generate a high quality DSL: decision, analysis, design, implementation, evaluation, application, and maintenance.[10][11]

The decision to develop a DSL should be considered carefully. Several factors have to be taken into account like the requirements to the developer, the efforts in time and costs, and the reusability of the DSL. Čeh and her colleagues defined a guideline for developing DSLs:

> "*...a DSL should be developed whenever it is necessary to solve a problem that belongs to a problem family and when we expect that in the future more problems from the same problem family will appear.*"[11]

The knowledge maintenance of (distributed) CBR systems is a problem family and the specific use cases for one or more CBR systems are the individual problems to solve. Therefore, a DSL for knowledge maintenance could be applied to a wide range of existing CBR systems. In addition, every year more CBR systems are developed in research and industry and for all these CBR systems the maintenance problem will occur, even if it is not solved for every system. But a DSL will ease solving the problem, especially in combination with a meta maintenance approach that combines and integrates different maintenance goals, policies, strategies, and actions into one maintenance framework. And the development of this meta framework itself, namely the maintenance cockpit, will also be easier with a designed and implemented DSL for knowledge maintenance. It could be used to provide the later users with different pre-defined policies and strategies as well as allow the definition of new maintenance approaches by combining the individual language elements of the DSL. Other decision factors in favor of a DSL are a unified vocabulary for maintenance approaches with syntax and semantics [10][12], the effort for the integration of dependencies between existing maintenance elements and future elements can be reduced [10], and the potential of reduced complexity of knowledge maintenance for the CBR systems [13]. After considering all these factors, we decided that the potential benefits of a domain-specific language for knowledge maintenance of CBR systems are greater than the effort for the development and therefore started to develop a DSL called ***D**omain-specific **L**anguage for **M**odelling **M**aintenance **S**trategies for Case-based Reasoning Systems* or short DLMMS^{CBR}.

In the rest of this paper we focus on the analysis and the design phase of the development process as far as we have proceeded and present a first design idea for the DSL. We describe briefly a domain analysis in the field of knowledge maintenance for CBR systems and present an excerpt of the derived language elements to describe the important aspects of the domain.

3.1. Domain analysis

The second phase of the development process for a DSL is an analysis of the targeted domain. First, the domain was divided into three topic areas for a later formal analysis: the fundamental aspects of knowledge maintenance (e.g. types of maintenance, maintenance goals, maintenance policies, and maintenance actions), existing methodological approaches (e.g., INRECA [14], SIAM [15], DISER/DILLEBIS [16], Case Factories [7]), and strategical approaches (e.g., competence-based maintenance, introspective learning, and flexible feature deletion). For the formal analysis of the topic areas the feature-oriented domain analysis (FODA) methodology is used. With FODA the concepts of the target domain can be identified and described and for each concept attributes are defined and transformed into a model of the domain. The attributes can either be mandatory or variable. Mandatory attributes contain information that are required to realize an identified concept. Variable attributes can be divided into optional or alternative attributes, with alternative attributes being a parent to optional attributes. Based on the properties of the alternative attribute, several optional attributes can be selected in parallel. FODA creates an attribute model for a given concept, which provides a hierarchical structure. The hierarchy of the model describes the ranking of the individual attributes. In addition to attribute properties, other means can be used to restrict the attribute selection. FODA allows the use of compositional rules to describe dependencies and restrictions of the selection combinations of attributes. For optional attributes additional justifications can be used in FODA to describe why an attribute should be selected.[17]

The analysis was conducted in four steps: in the first three steps information about the vocabulary, the functions, and the methodological as well as strategical approaches of the knowledge maintenance were collected and matching FODA models were built. As a tool for building the FODA models FeatureIDE, a plugin for the Eclipse development engine, is used [18]. In the last step, a domain model based on the previous defined FODA models was created.

At first the basic aspects of knowledge maintenance were defined in an initial FODA model. This model contains elements for the different types of maintenance defined by Swanson [19] and Lehner[20]. Figure 1 shows an excerpt of the resulting FODA model for these types of maintenance. This model was extended by DSL elements for maintenance goals defined by Smyth and McKenna [21] and for the maintenance policies from Wilson and Leake [22]. The maintenance goals are divided into competence, efficiency, and quality, while for the maintenance policies a more complex attribute structure was created based on the five categories *data collections*, *triggering*, *activity*, *operation types*, and *execution*. Each category is represented by an own attribute in the FODA model and has several sub-attributes. The current version of this FODA model representing a basic vocabulary has 62 attributes and up to four hierarchical levels.

Another FODA model is based on the analysis of the strategical maintenance approaches for CBR systems researched and developed in the last decades. Most of the maintenance approaches developed through the years are targeting the case base: competence-based knowledge maintenance [23][24][25], case-addition policies [26], soft CBM methods [27][28], inconsistency-based policies [29], complexity-based approaches [30], and adaptation-guided approaches [31] to name only some of the developed maintenance approaches for case bases. While most of the maintenance approaches target the case base, there are approaches for maintaining the other

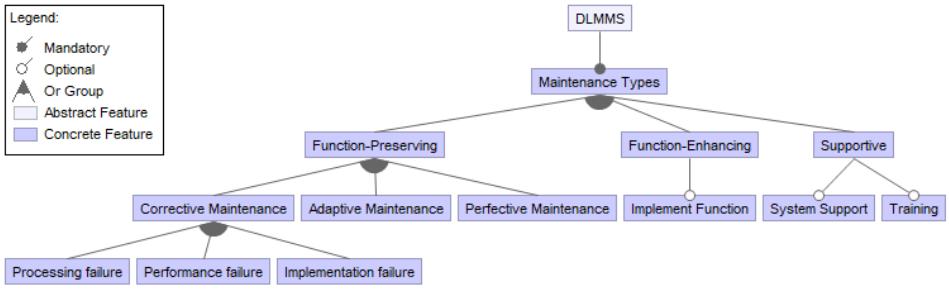

Figure 1: Excerpt of the FODA Model for maintenance types according to [19][20].

knowledge containers, too. For the similarity measures, Wettschereck and Aha compared different feature weighting methods and developed five dimensions to describe these methods: Model, weight space, representation, generality and knowledge [32]. The approach from Richter and Wess introduces a so-called relevance matrix to deal with irrelevant symptoms in the PATDEX/2 diagnosis system [33]. These relevance matrices are used in many CBR systems nowadays. Other approaches for learning feature weights present a framework for learning of similarity measures, which is able to learn local and global similarity measures as well as feature weights [34] or use domain-specific information to assign weights based on class frequencies for metric and symbolic feature-value pairs [35]. For vocabulary maintenance, approaches for flexible feature deletion [36], and a constraint-based approach for reducing redundancies and noise [37] can be named. The adaptation knowledge can be maintained for example by generating new adaptation rules by analyzing the solution of cases to find their variances [38] or by using the EAR algorithm for generating adaptation rules [39]. The FODA model created based on this step contains 89 attributes and also up to four hierarchical levels.

All these maintenance approaches for individual knowledge containers can be used in the Case Factories of the maintenance cockpit. While the maintenance cockpit does not propose a specific evaluation algorithm or maintenance action, it can incorporate the approaches mentioned above. The individual algorithms have to be implemented and integrated into the agents responsible for evaluation and maintenance. This requires sometimes a differentiation between the evaluation part of an algorithm, for example the evaluation of the cases to identify pivotal, support, and auxiliary cases, and the maintenance action itself, for example the deletion of auxiliary cases. The maintenance cockpit provides an organizational superstructure for maintaining distributed CBR systems and the available knowledge and the selected maintenance strategies decide which maintenance approaches to be useful. A domain-specific language would be very helpful to formulate all these maintenance approaches in a common vocabulary and implement them to be used by the maintenance cockpit or other meta-level frameworks. The implemented DSL could also be used by individual CBR systems to deploy maintenance policies and strategies.

3.2. DSL design

Once the domain scope has been defined and relevant information was collected in the analysis phase, the DSL can be designed in the next phase. Within this design phase, elements and constructs for DSL are developed based on the analysis results. In addition, a meaning is assigned to the individual constructs and elements. Before actively designing the objects of the DSL, there are two factors to consider in the design phase. The first factor is the extent to which the new DSL has a relationship with existing languages. Three variants are possible for this relationships: piggyback, specialization, and extension [10]. The second factor is the methodological approach that will be used for the design of the DSL. Several methodological approaches exist to specify the elements and constructs of a DSL, for example syntax specification, attribute grammar, and denotation semantics [40].

For our DSL, we have chosen to develop the DSL as an independent language with an extension relationship to other GPLs. The DSL should be developed independently of the elements of existing languages in order to avoid the development process and design of the DSL being restricted by existing languages. The extension aspect takes into account that the designed DSL may be integrated into a GPL in order to transform domain-specific expressions of the knowledge maintenance into source code. This could be achieved by integrating the DSL as a library into the GPL. As a methodological approach, the syntax specification was chosen to define the elements of the DSL based on the FODA models and the domain model. The DSL is based on a combination of two design concepts, which are used as guidelines during the design phase. The first concept is based on the idea to provide a common vocabulary for knowledge maintenance to describe the aspects of existing maintenance approaches. This way, a user should be enabled to apply any of these maintenance approaches to a CBR system supported by a guided process. The second concept will provide more flexibility and should enable the users to define their own maintenance approaches by using the defined vocabulary without being limited to the defined approaches within the DSL.

The first element of the DSL is an attribute to determine whether the maintenance modeling should use predefined structures or is individually defined by a user. Therefore, a variable *mainSelect* is created and has a non-terminal attribute *maintenanceOptions* as value. This value can be set with two terminal attribute values. A formal definition can be found in Equation 1.

$$\begin{aligned} mainSelect &= [maintenanceOptions] \\ \text{where maintenanceOptions } &\in \{individual, predefined\} \\ \text{and maintenanceOptions } &\neq \{\emptyset\} \end{aligned} \tag{1}$$

After selecting the predefined modeling option, basic maintenance information is displayed for the user to determine. Another variable named *condition* is created, which has three non-terminal values. These three non-terminal values can be set by different terminal attributes: the maintenance goal, the maintenance type, and the knowledge container to be maintained. The formal definition of the condition variable can be found in Equation 2.

$$
\begin{aligned}
condition &= \{[goal], [type], [container]\} \\
\text{where goal } &\in \{competence, efficiency, quality\} \\
\text{and type } &\in \{corrective, adaptive, perfecting\} \\
\text{and container } &\in \{casebase, vocabulary, similarity, adaptation\} \\
\text{and target, type, container } &\neq \{\emptyset\}
\end{aligned} \tag{2}
$$

Once the user has defined the modeling option and the basic maintenance information, the decisions based on the users decision are presented in which the user should insert further parameters or attributes to describe the maintenance process. For modeling purposes, a uniform framework is created which can be extended with attributes defined for maintenance strategies. The aspects of the maintenance policies, which were modeled in the first FODA model, are suitable for such a framework. A formal definition of this overall framework can be found in Equation 3.

$$
\begin{aligned}
maintenance_{pre} &= \{[collection], [trigger], [operation], [execution]\} \\
\text{collection} &= \{attribute_1, ...attribute_n\} \\
\text{trigger} &= \{attribute_1, ...attribute_n\} \\
\text{operation} &= \{attribute_1, ...attribute_n\} \\
\text{execution} &= \{attribute_1, ...attribute_n\}
\end{aligned} \tag{3}
$$

The *maintenance$_{pre}$* variable contains information about the data collection, the trigger, the tasks and the execution of the maintenance and is specifically designed for the predefined maintenance strategies. The four aspects contain any number of attributes, which vary according to the maintenance strategy and can be further specified, for example, by assigning parameters to them. Within the DSL, the predefined maintenance strategy for competence-based maintenance [23] can be represented as seen in Equation 4.

$$
\begin{aligned}
maintenance_{pre} &= \{[collection], [trigger], [operation], [execution]\} \\
\text{collection} &= \{competenceRating(coverage, reachability) \\
\text{and } &competenceAssignment(auxilary, pivotal, spanning, support)\} \\
\text{trigger} &= \{maxCaseNumber = [number]\} \text{ where } number \in \{1...n\} \\
\text{operation} &= \{FDS \text{ or } FUDS\} \\
\text{execution} &= \{offline \text{ or } online\}
\end{aligned} \tag{4}
$$

The attributes *coverage* and *reachability* are used as parameters in the attribute *competenceRating* to evaluate the competence of cases. The competence categories are bundled in the attribute *competenceAssignment* and represents the task of assigning the cases to the respective category. The trigger considered in this approach is the number of cases in the case base. The

threshold defines the maximum number of cases allowed for a case base. If this threshold is exceeded, the maintenance task is triggered. As maintenance actions or transactions, the footprint deletion strategy (FDS) or the footprint utility deletion strategy (FUDS) are defined here. Depending on the maintenance information selected, either one or both deletion strategies are proposed to the user. The execution of the selected maintenance task could take place both inside and outside the problem-solving cycle, so that the user can decide between two attribute values. The two values are offline or online execution according to the first FODA model. Other maintenance strategies can be defined in a similar way with different attributes and attribute values. This way, existing maintenance approaches can be defined within DLMMS^{CBR}.

For maintenance strategies that will be individually defined by the user, the same framework from Equation 3 is used, but the main variable in named *maintenance*$_{ind}$. The main difference is that the attributes and their values for *collection*, *trigger*, *operation*, and *execution* are not predefined as for a specific maintenance strategy, but can be selected from a finite set of attributes. This way, the user can define an own maintenance strategy, within certain boundaries. Currently, the DSL contains four different *triggers* that can be used: the maximal number of cases, attributes, and adaptation rules and the maximum problem solving time. The *operations* attribute contains information about maintenance actions and transactions. The basic maintenance actions are adding, changing, or deleting knowledge. Therefore, the attributes select-able as operations are defined in Equation 5.

$$\begin{aligned} operation = \{action([knowledgeElement])\} \\ \text{where } action \in \{add, change, delete\} \text{ and} \\ knowledgeElement \in \{Case_x, adaptRule_x, attribute_x, Case_x(attribute_x)\} \\ \text{and } x \in 1...n \end{aligned} \tag{5}$$

Selecting attribute$_x$ as action parameter means that the selected attribute is added, changed, or deleted globally. If Case$_x$(attribute$_x$) is selected as a knowledge element, it means that the selected attribute is accessed in a specifically selected case. Other maintenance actions could be the adjustment of attribute weights or similarity values. The knowledge elements are currently modeled on a high level and have to be extended to fit all knowledge levels of the hierarchy presented in Section 2.

4. Conclusion and Outlook

This paper presents the first steps on the way to a domain-specific language for knowledge maintenance in CBR systems, called DLMMS^{CBR}. We describe the meta maintenance approach of the Case Factories and the maintenance cockpit as a use case for the later completely defined and implemented DSL. In addition, we give an overview of the current development state of the DSL based on the domain analysis and the ongoing design step. The current state of the DSL is preliminary and it will be developed further during the next month. The goal is to represent the majority of existing maintenance approaches within the DSL and implement it as a JAVA library to be used in the maintenance cockpit framework.

References

[1] P. Reuss, Case Factories: A Maintenance Cockpit for distributed structural Case-based Reasoning Systems, Ph.D. thesis, University of Hildesheim, Hildesheim, Germany, 2019.

[2] K. Bach, Knowledge Acquisition for Case-Based Reasoning Systems., Ph.D. thesis, University of Hildesheim, 2013. Dr. Hut Verlag München.

[3] K.-D. Althoff, A. Hanft, M. Schaaf, Case factory - maintaining experience to learn, Advances in Case-Based Reasoning Lecture Notes in Computer Science 4106/2006 (2006) 429–442.

[4] K.-D. Althoff, M. Reichle, K. Bach, A. Hanft, R. Newo, Agent based maintenance for modularised case bases in collaborative mulit-expert systems, in: Proceedings of the AI2007, 12th UK Workshop on Case-Based Reasoning, 2007.

[5] P. Reuss, K.-D. Althoff, Explanation-aware maintenance of distributed case-based reasoning systems, in: LWA 2013. Learning, Knowledge, Adaptation. Workshop Proceedings, 2013, pp. 231–325.

[6] P. Reuss, K.-D. Althoff, Dependencies between knowledge for the case factory maintenance approach, in: LWA, 2015, pp. 256–263.

[7] P. Reuss, C. Witzke, K.-D. Althoff, Dependency modeling for knowledge maintenance in distributed cbr systems, in: D. W. Aha, J. Lieber (Eds.), Case-Based Reasoning Research and Development. International Conference on Case-Based Reasoning (ICCBR-17), June 26-28, Trondheim, Norway, Springer, 2017, pp. 302–314.

[8] F. Höwing, Effizeinte entwicklung von autosar-komponenten mit domänenspezifischen programmiersprachen (in german), in: O. Herzog, K.-H. Rödiger, M. Ronthaler, R. Kochke (Eds.), Informatik 2007 - Informatik trifft Logistik - Band 2, Gesellschaft für Informatik e.V., 2007, pp. 551–556.

[9] A. Deursen, P. Klint, Domain-specific language design requires feature descriptions, Journal of Computing and Information Technology 10 (2002).

[10] M. Mernik, J. Heering, A. M. Sloane, When and how to develop domain-specific languages, ACM Comput. Surv. 37 (2005) 316–344.

[11] I. Čeh, M. Črepinšek, T. Kosar, M. Mernik, Ontology driven development of domain-specific languages, Comput. Sci. Inf. Syst. 8 (2011) 317–342.

[12] M. Fowler, Domain Specific Languages, 1st ed., Addison-Wesley Professional, 2010.

[13] D. Wile, Supporting the dsl spectrum, Journal of Computing and Information Technology (cit@srce.hr); Vol.9 No.4 9 (2001).

[14] K.-D. Althoff, Evaluating case-based reasoning systems. the inreca case study., 1997. URL: http://www.iis.uni-hildesheim.de/files/staff/althoff/Publications/althoff-habil-1997-07-09.pdf, postdoctoral thesis, TU Kaiserslautern.

[15] T. Roth-Berghofer, Knowledge maintenance of case-based reasoning systems. The SIAM methodology., Akademische Verlagsgesellschaft Aka GmbH, 2003.

[16] M. Nick, Experience Maintenance Loop through Closed-Loop Feedback, Ph.D. thesis, TU Kaiserslautern, 2005.

[17] K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990. URL: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11231.

[18] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, T. Leich, Featureide: An extensible framework for feature-oriented software development, Science of Computer Programming 79 (2014) 70–85. Experimental Software and Toolkits (EST 4): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT-3 2010).
[19] E. B. Swanson, The dimensions of maintenance, in: Proceedings of the 2Nd International Conference on Software Engineering, ICSE '76, IEEE Computer Society Press, Los Alamitos, CA, USA, 1976, pp. 492–497. URL: http://dl.acm.org/citation.cfm?id=800253.807723.
[20] P. D. F. Lehner, Ergebnisse einer untersuchung zur wartung von wissensbasierten systemen (in german), Information Mangement 2 (1994) 38–47.
[21] B. Smyth, E. McKenna, Footprint-based retrieval, in: ICCBR, 1999.
[22] D. C. Wilson, D. B. Leake, Maintaining cased-based reasoners: Dimensions and directions, Computational Intelligence 17 (2001) 196–213.
[23] B. Smyth, M. Keane, Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems., in: Proceedings of the 13th International Joint Conference on Artificial Intelligence, 1995, pp. 377–382.
[24] B. Smyth, E. McKenna, Competence models and the maintenance problem, Computational Intelligence 17 (2001) 235–249.
[25] D. Mathew, S. Chakraborti, Competence guided model for casebase maintenance, in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, 2017, pp. 4904–4908.
[26] Q. Yang, J. Zhu, A case-addition policy for case-base maintenance, Computational Intelligence 17 (2001) 250–262.
[27] A. Smiti, Z. Elouedi, SCBM: soft case base maintenance method based on competence model, J. Comput. Science 25 (2018) 221–227.
[28] Q. Yang, J. Wu, Keep it simple: A case-base maintenance policy based on clustering and information theory, in: H. J. Hamilton (Ed.), Advances in Artificial Intelligence, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 102–114.
[29] K. Racine, Q. Yang, Maintaining unstructured case bases, in: D. B. Leake, E. Plaza (Eds.), Case-Based Reasoning Research and Development, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 553–564.
[30] S. Craw, J. Jarmulak, R. Rowe, Maintaining retrieval knowledge in a case-base reasoning system., Computational Intelligence 17 (2001) 346–363. doi:10.1111/0824-7935.00149.
[31] V. Jalali, D. Leake, Adaptation-guided case base maintenance, volume 3, 2014.
[32] D. Wettschereck, D. W. Aha, Weighting features, in: M. Veloso, A. Aamodt (Eds.), Case-Based Reasoning Research and Development, Springer Berlin Heidelberg, Berlin, Heidelberg, 1995, pp. 347–358.
[33] M. Richer, S. Wess, Similarity, uncertainty and case-based reasoning in patdex, Automated Reasoning (1991) 249–265.
[34] A. Stahl, Learning feature weights from case order feedback., in: Case-Based Reasoning Research and Development: Proceedings of the Fourth International Conference on Case-Based Reasoning, 2001.
[35] N. Elprince, O. Badawy, Feature weight maintenance in active case-based reasoning system, 2003.
[36] D. Leake, B. Schack, Flexible feature deletion: Compacting case bases by selectively

compressing case contents, in: E. Hüllermeier, M. Minor (Eds.), Case-Based Reasoning Research and Development, Springer International Publishing, Cham, 2015, pp. 212–227.
[37] S. Ben Ayed, Z. Elouedi, E. Lefevre, Cevm: Constrained evidential vocabulary maintenance policy for cbr systems, 2019.
[38] K. Hanney, M. T. Keane, Learning adaptation rules from a case-base, in: I. Smith, B. Faltings (Eds.), Advances in Case-Based Reasoning, Springer Berlin Heidelberg, Berlin, Heidelberg, 1996, pp. 179–192.
[39] V. Jalali, D. Leake, Extending case adaptation with automatically-generated ensembles of adaptation rules, in: S. J. Delany, S. Ontañón (Eds.), Case-Based Reasoning Research and Development - 21st International Conference, ICCBR 2013, Saratoga Springs, NY, USA, July 8-11, 2013. Proceedings, volume 7969 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 188–202.
[40] K. Slonneger, B. Kurtz, Formal Syntax and Semantics of Programming Languages: A Laboratory Based Approach, 1st ed., Addison-Wesley Longman Publishing Co., Inc., USA, 1995.