Dependency Modeling for Knowledge Maintenance in Distributed CBR Systems

Pascal Reuss[1,2] and Christian Witzke[1] and Klaus-Dieter Althoff[1,2]

[1]Intelligent Information Systems Lab, University of Hildesheim
[2]Competence Center CBR, German Center for Artificial Intelligence, Kaiserslautern

Abstract. Knowledge-intensive software systems have to be continuously maintained to avoid inconsistent or false knowledge and preserve the problem solving competence, efficiency, and effectiveness. The more knowledge a system contains, the more dependencies between the different knowledge items may exist. Especially for an overall system, where several CBR systems are used as knowledge sources, several dependencies exist between the knowledge containers of the CBR systems. The dependencies have to be considered when maintaining the CBR systems to avoid inconsistencies between the knowledge containers. This paper gives an overview and formal definition of these maintenance dependencies. In addition, a first version of an algorithm to identify these dependencies automatically is presented. Furthermore, we describe the current implementation of dependency modeling in the open source tool myCBR.

1 Introduction

Knowledge-intensive systems are using a high amount of knowledge that is not stored in a single knowledge source, but distributed over several knowledge sources. This leads to a better scalability and maintainability. Especially for systems with several case-based reasoning (CBR) systems, distributing the knowledge over several small CBR systems rather than using one large CBR system, has a great benefit on maintainability. There exist several maintenance approaches for CBR systems [16, 20, 19, 9, 8] that aim to maintain a single knowledge container in a single CBR system. However, there could be dependencies between the knowledge inside a CBR system and between the knowledge of different CBR systems. These dependencies should be considered for maintenance actions to ensure the consistency and competence of the whole knowledge-intensive system. Based on these dependencies additional maintenance actions could be identified to avoid inconsistencies or competence loss. These dependencies [13] are used by the Case Factory approach for maintaining distributed CBR systems [12]. The initial definition of the dependencies were not sufficient and we analyzed the required granularity of the dependencies and the possible knowledge on different granularity levels. In this paper we present the refined definition of the maintenance dependencies and the current modeling and implementation in the open source tool myCBR [21, 5]. We also investigate the possibility of generating dependencies automatically to reduce the modeling effort.

In Section 2 we describe related research in the fields of knowledge modeling and maintenance. Section 3.1 contains a brief description of the Case Factory(CF) approach

and Section 3.2 describes in more detail the refined dependencies required for maintenance with CFs and a first version of the dependency generation algorithm. In Section 3.3 we present the current modeling possibilities in myCBR. Finally, we give a summary of our paper and an outlook to future work.

2 Related work

Knowledge modeling has been a focus of research in many communities in the last decades. Directly related to CBR, the knowledge containers [14] are a central point for knowledge modeling, which were extended with a maintenance container [10]. Our maintenance dependencies clearly belong to the maintenance knowledge of a CBR system and the Maintenance Map can be treated as an instantiation of a maintenance container. However, for our maintenance approach with explanations the knowledge from the other maintenance containers are required, too. The description of knowledge contents in CBR systems on a so-called knowledge level was first introduced by Aamodt [2]. The CBR community adapted the knowledge level view from the knowledge acquisition community to describe knowledge in CBR systems independent from the implementation. Our approach uses also the knowledge level view, but introduced different sub-levels of knowledge in the different knowledge containers. This way, we are able to build a hierarchy of knowledge for our maintenance dependencies to model abstract and detailed dependencies.

Dependency modeling between knowledge is researched in the economic domain. The focus is to model the dependencies between knowledge in firms and organizations or in business processes. Different approaches to model dependencies have been developed, for example strategic dependency diagrams [3], dependency modeling with OWL-DL, and a meta-model for dependencies [18]. In the business domain, knowledge dependencies are modeled to determine the performance of a firm or the scalability of business processes.

Software development also deals with dependency modeling. They are for example used to manage complex software applications and can identify violations of the architecture, evaluate the scalability of an application and identify hidden subsystems [17]. Also from the software development perspective comes an approach for a domain-specific dependency modeling language [1]. In our approach the knowledge levels and the defined hierarchy is a first step to a language for maintenance dependency modeling in CBR systems. An explicit language could help to identify all dependencies between the knowledge items in and between CBR systems.

3 Dependency Modeling for Knowledge Maintenance

This section describes the Case Factory maintenance approach and the required knowledge dependencies. We introduce different knowledge levels and a hierarchy of dependencies for CBR systems. In addition, we present a first algorithm for generating syntactic dependencies automatically and describe the current implementation of dependency modeling in myCBR.

3.1 Maintenance with Case Factories

The Case Factory approach is an agent-based maintenance approach for distributed CBR systems and integrated into the SEASALT architecture [4]. The SEASALT (**S**haring **E**xperience using an **A**gent-based **S**ystem **A**rchitecture **L**ayout) architecture is a domain-independent architecture for multi-agent systems to extract, analyze, share, and provide experience. The architecture consists of five components. The first component is the knowledge source component. This component contains so-called collector agents that are responsible for the extraction of knowledge from external knowledge sources. Knowledge sources could be databases, files, forums, or blogs. The second component, knowledge formalization, formalizes the extracted knowledge from the collector agents into a structural representation to be used by the third component, the knowledge provision component. This component manages the knowledge sources inside a multi-agent system (MAS) instantiated by the SEASALT architecture. Inside the knowledge provision component the so-called Knowledge Line (KL) is located. The KL contains a number of topic agents with access to internal knowledge sources like CBR systems. The basic idea is to modularize the knowledge among the topic agents and decide which knowledge is required to solve a given problem. The fourth component is the knowledge representation, that contains the underlying knowledge models for the different agents and knowledge sources. The last component is the individualized knowledge and contains the user interface for querying the system and displaying the solution [4].

The extended Case Factory approach extends the SEASALT architecture with a maintenance mechanism for CBR systems. If a topic agent has access to a CBR system, a CF is provided to maintain the CBR system. To coordinate several CFs a so-called Case Factory Organization (CFO) is provided, which consists of several agents to coordinate the overall system maintenance. A Case Factory consists of several agents that are responsible for different tasks: monitoring, evaluation, coordination, and maintenance execution. A monitoring agent will supervise the knowledge containers of a CBR system to notice changes to the knowledge like adding new cases, changing the vocabulary, or deleting cases. Monitoring agents will only notice the fact that changes have occurred and what has been changed. Evaluation agents are responsible for a qualitative evaluation of the consistency, performance, and competence of the CBR system. Which evaluation strategy is performed is up to the user. Existing approaches like utility footprint [19] or sensitivity analysis [7, 22] could be applied as well as new or modified evaluation strategies. The coordination agent will collect all results from the monitoring and evaluation agents and create maintenance actions based on the collected information. In addition, the agent will use the modeled dependencies to determine additional maintenance actions that should be performed and sent them to the CFO. The dependencies and their use will be described in more detail in the next section. Maintenance execution agents are responsible for executing the confirmed maintenance actions and adapting the knowledge of the CBR system.

The Case Factory Organization is a superstructure for coordinating the maintenance activities of each Case Factory and providing the knowledge engineer with the required information to confirm or reject maintenance actions. Therefore, several agents with different tasks are part of the CFO: coordination agent, maintenance planning agent, explanation agent, and communication agent. The coordination agent gets all main-

tenance actions from the different CFs, derives additional maintenance actions based on the dependencies between CBR systems and passes the list of maintenance actions to the maintenance planning agent. The planning agent is responsible for creating a maintenance plan from all derived maintenance actions. Therefore, the agent checks for duplicate or conflicting maintenance actions, the order of maintenance actions and for circular maintenance actions. Based on these checks a maintenance plan is generated and passed to the explanation agent. This agent generates a human-readable explanation for each action in the maintenance plan and adds it to the corresponding maintenance action. The enhanced plan is passed to the communication agent and displayed to the knowledge engineer. After a review by the knowledge engineer, the confirmed or rejected maintenance actions are passed back to the individual CFs [11, 12].

3.2 Dependencies

A central part of the Case Factory approach are the dependencies between the knowledge containers of CBR systems. These 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. This triple has been used by our first dependency definition and the source and target were defined as knowledge containers and can be found in equation (1).

$$\begin{aligned} d &= (kc_{sysS}, kc_{sysT}, t) \\ \text{where } kc &\in \{voc, sim, cb, ada\} \text{ and } sysS, sysT \in \{1 \dots n\} \text{ and } t \in \{u, b\} \end{aligned} \tag{1}$$

The triple consists of two knowledge containers and a direction. The first knowledge container kc_{sysS} defines the left side of a dependencies, the source knowledge container. The second knowledge container kc_{sysT} defines the target knowledge container. Knowledge containers could be the vocabulary, the case base, the similarity measures or the adaptation knowledge according to Richter [15]. The indices $sysS$ and $sysT$ identifies the CBR system, the knowledge container belongs to, assuming all given CBR systems have a number between 1 and n. The last part of the triple t represents the direction of a dependency, either (u)ni-directional or (b)i-directional and determines whether the source and the target knowledge containers can be swapped or not.

But the information about the affected knowledge containers is not sufficient. For example, the information that a dependency exists between the vocabulary of CBR system A and the vocabulary of CBR system B does only allow to derive a maintenance action for changing the vocabulary. This is not enough information for specific and executable maintenance actions. Therefore, we defined six knowledge levels for CBR systems to find the required granularity of knowledge for the dependencies. The knowledge levels are shown in table 3.2.

These knowledge levels are used to define a hierarchy of granularity for dependencies. The top level of the hierarchy is root level. It contains only one node, the root node. This root level could also be named as knowledge level 0. The second level represents the knowledge level 1 and contains nodes for the *CBR systems*. The third level, knowledge level 2, contains nodes for the knowledge containers of a CBR system: *vocabulary*,

Table 1. Knowledge levels, the contained knowledge, and examples

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	aircraft type, systems, status
Knowledge level 6	specific values	A380, display, inoperable

case base, *similarity measures*, and *adaptation knowledge*. The Knowledge level 3 contains nodes for the *case bases*, the other three branches have no nodes on this level. On knowledge level 4 nodes for the *cases*, *similarity measures*, and the *adaptation rules* can be found. Knowledge level 5 contains nodes for the *attributes* on all branches of the knowledge containers. The last level contains nodes for the *specific values* of each attribute. Figure 1 shows an example hierarchy with the knowledge levels.

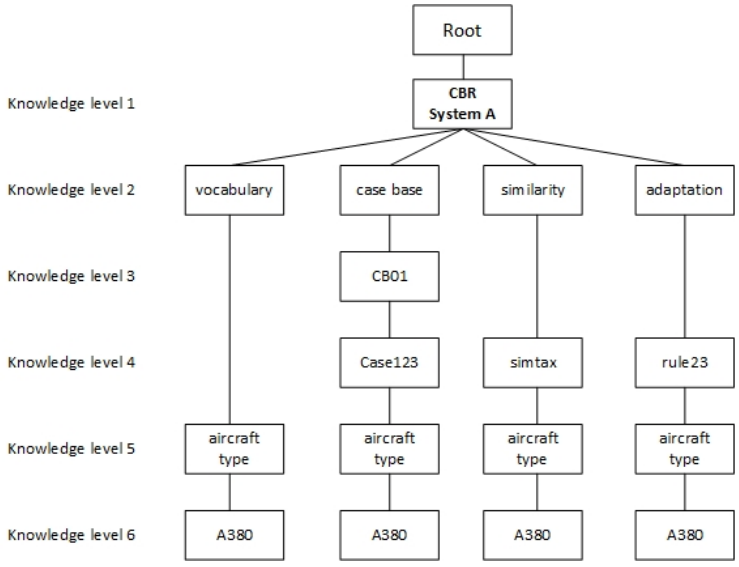

Fig. 1. Example hierarchy for granularity of dependencies

Based on this hierarchy, dependencies with different knowledge levels could 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 the knowledge on this level is required for the knowledge levels below to differ between the more detailed dependencies. In addition, it 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*. With this specific information, among others, a detailed maintenance action could be derived. The reworked dependency definition can be found in equation (2).

$$\begin{aligned} d &= (kle_{source}, kle_{target}, t) \\ \text{where } kle_{source} \text{ and } kle_{target} &\in \{hierarchy_n odes\} \text{ and } t \in \{u, b\} \end{aligned} \tag{2}$$

In the previous definition, the source and target of a dependency are knowledge containers. In the current definition, the source and target are knowledge level elements (kle), that can be found in the defined hierarchy. The hierarchy is a set of nodes and edges, but a dependency references only on nodes. Therefore, $hierarchy_n odes$ is a subset of the hierarchy, that only contains the nodes. To identify an element in the hierarchy, every element gets an id code based on the knowledge level, characters, and continuous numbers. The id code consists of alphanumeric characters and starts with the number of the knowledge level. The characters for each knowledge level are combined with an underscore. The knowledge level 0 will not be considered, because it contains no knowledge. The nodes on each knowledge level will be continuously numbered. The only exception is knowledge level 2. The nodes on this level are identified with the starting character of the knowledge container. For example the id code for the *case123* node would be 1_C_1_1_0_0, the id code for the specific value *A380* in the same branch would be 1_C_1_1_1_1. A dependency between the specific value *A380* of the attribute *aircraft type* in the *vocabulary* and the same value in *case123* in *CB01* in the attribute *aircraft type* could be found in equation (3):

$$d = (1\_V\_0\_0\_1\_1, 1\_C\_1\_1\_1\_1, u) \tag{3}$$

The dependencies will still be differentiated between intra- and inter-system dependencies. An intra-system dependencies is defined within a CBR system, while inter-system dependencies are defined between different CBR systems. An intra-sytem dependency is defined in equation (4), while an inter-system dependency is defined in equation (5):

$$\begin{aligned} d_{intra} &= (kle_{source}, kle_{target}, t) \\ \text{where } kle_{source} \text{ and } kle_{target} &\in \{hierarchy_n odes\} \\ \text{and } \#KL6 \text{ of } kle_{source} &\neq \#KL6 \text{ of } kle_{target} \\ \text{and } \#KL1 of kle_{source} &= \#KL1 \text{ of } kle_{target} \\ \text{and } t &\in \{u, b\} \end{aligned} \tag{4}$$

$$
\begin{aligned}
d_{inter} &= (kle_{source}, kle_{target}, t) \\
\text{where } kle_{source} \text{ and } kle_{target} &\in \{hierarchy_n odes\} \\
\text{and } \#KL1 \text{ of } kle_{source} &\neq \#KL1 \text{ of } kle_{target} \\
\text{and } t &\in \{u, b\}
\end{aligned}
\tag{5}
$$

Intra-system dependencies are defined within a single CBR system. Therefore, the source and the target knowledge level elements have the same CBR system identification value (#KL1), while they have different attribute value identification values (#KL2). This is required to avoid dependencies from specific values to themselves. This way we avoid circular processing of the same dependencies endless times. In contrast to the previous definition of intra-system dependencies, dependencies within the same knowledge container are permitted to model dependencies between different attributes of the vocabulary for example. For inter-system dependencies the CBR system identification value (#KL1) has to be different, while all other knowledge level elements could have the same identification number, for example when we have a backup copy of CBR system that contain the same knowledge.

In our first definition of dependencies[13] we introduced three trivial dependencies as intra-system dependencies. These trivial dependencies were defined between the vocabulary and the three other knowledge containers. On an abstract level this dependencies still exist, but on the knowledge levels 5 and 6, the number of dependencies cannot be defined in general, because the number of attributes and specific values depends on the knowledge modeling. Therefore, the new definition of our maintenance dependencies contain no trivial dependencies any more.

Based on the defined granularity of the dependencies, there could exist hundreds of dependencies in a CBR system. Modeling all these dependencies manually would cause a very high effort for the knowledge engineer. Therefore, the automated generation of dependencies based on a given knowledge model could reduce the effort and would allow the application of the Case Factory approach to existing CBR systems with a manageable effort. For an automated generation we have to differentiate between syntactic and semantic dependencies. A syntactic dependency is based on a syntactic compliance of values, for example the specific value *A380* of the attribute *aircraft type* in the *vocabulary* and the value *A380* in the same attribute in a given case. This dependency could be generated automatically by searching for the value *A380* in existing cases. If the value is set for the attribute in a case, a dependency is modeled. A first version of an algorithm to generate syntactic dependencies is shown in algorithm 1.1.

Listing 1.1. Algorithm for generating syntactic dependencies

Definitions:

V_a Set of values for attribute a
C_{cb} Set of cases in a case base cb

v_a specific value of attribute a
c_{cb} specific case of case base cb
v_{fct} specific value in similarity measure
v_r specific value in rule

Input:

A Set of attributes in the case structure
CB Set of case bases in a CBR system
R Set of adaptation rules
S Set of similarity functions

Output:

D Set of syntactic dependencies

Algorithm:

```
D = \emptyset

foreach (attribute a in A) {
  if (check(v_a exist in c_cb) {
          d_u = new d(v_a, v_c, u)
                if (exist(D, reverse(d_u s))) {
                        d_b = new d(v_a, v_c, b)
                        D = D - reverse(d_u) }
                else {
                        D = D + d_b }
        }
        if (check(v_a exist in v_fct) {
                d_u = new d(v_a, v_fct, u)
                if (exist(reverse(d_u))) {
                        d_b = new d(v_a, v_fct, b)
                        D = D - reverse(d_u) }
                else {
                        D = D + d_b }
        }
        if (check(v_a exist in v_r) {
                        d_u = new d(v_a, v_r, u)
                        if (exist(reverse(d_u))) {
                                d_b = new d(v_a, v_r, b)
                                D = D - reverse(d_u) }
                        else {
                                D = D + d_b }
        }}
return D
```

The algorithm iterates over all values of all attributes and compares them with the used values in the cases, the similarity measures and the adaptation rules. If a compliance is found a new dependency on the sixth knowledge level is created. The created dependency is defined a uni-directional. For every created dependency, the algorithm checks whether a reverse dependency exists, or not. If a reverse dependency is found, the newly created dependency is set to bi-directional and the reverse dependency is removed from the list. The new dependency is added to the list. This way explicit dependencies between the knowledge containers can be generated. Implicit dependencies between the knowledge containers can be found in several CBR tools and in myCBR, too. But these implicit dependencies cannot be used to generate maintenance actions and explanations. Therefore, some dependencies may exist implicitly and explicitly.

With the algorithm syntactic dependencies can be generated, but not semantic dependencies. A semantic dependency is based on user modeled connections between the knowledge items, for example a case referencing another case. A reference cannot always be identified automatically, therefore a dependency can only be generated under specific circumstances. For example, it could be checked if an attribute exists, that contains unique identifiers of cases. This attribute could be treated as reference attribute and the identifiers of the cases could be compared syntactically.

3.3 Dependency modeling using myCBR

We have extended the API and workbench of our tool myCBR to model and visualize dependencies. The algorithm for generation dependencies is not implemented yet. The API was extended with functions and classes to model, save, and load dependencies. Dependencies are stored in a so-called Maintenance Map. This Maintenance Map is based on the Knowledge Map[6] and stores all information about the dependencies. A Maintenance Map can be exported in RDF format. The following excerpt from an Maintenance Map shows an example modeling:

Listing 1.2. Excerpt from an example Maintenance Map

```
<rdf:Description rdf:about="'dependency1"'>
        <dep:source>1\_V\_0\_0\_1\_1</dep:source>
        <dep:target>1\_C\_1\_1\_1\_2</dep:target>
        <dep:type>1</dep:type>
        <dep:weight>1</dep:weight>
</rdf:Description>
<rdf:Description rdf:about="'dependency2"'>
        <dep:source>1\_V\_0\_0\_1\_1</dep:source>
        <dep:target>2\_V\_0\_0\_1\_2</dep:target>
        <dep:type>2</dep:type>
        <dep:weight>4</dep:weight>
</rdf:Description>
```

The example shows two dependencies, one intra-system and one inter-system dependency. For each dependency the source and the target are stored with their id code in the defined hierarchy. In addition, the direction of the dependency and the weight are stored. The direction can either be one or two, one if the dependency is uni-directional

and 2 if it is bi-directional. The weight defines the importance of the dependency, the higher the weight, the more important the dependency is. A higher weight can be used to rank maintenance actions based on the according dependency. The dependencies are stored with their id code in the Maintenance Map. After reading the dependencies the id code is transformed into the more detailed knowledge level information.

The workbench was extended with a maintenance view. This view allows a user the modeling and visualization of dependencies. Figure 2 shows the maintenance view and a modeled dependency. The current implementation allows the creation of Maintenance Maps for intra- and inter-system dependencies. Intra-system dependencies are stored into a so-called local Maintenance Map, while inter-system dependencies are stored in a so-called global Maintenance Map. The differentiation is only for organizational purpose. After the creation of a Maintenance Map the associated dependencies could be modeled. A dependency can currently be modeled up to knowledge level 5. The sixth knowledge level is under development. For a local Maintenance Map the target CBR system is automatically set to be identical with the source CBR system. Dependencies with missing information cannot be saved to avoid incomplete and inconsistent dependencies.

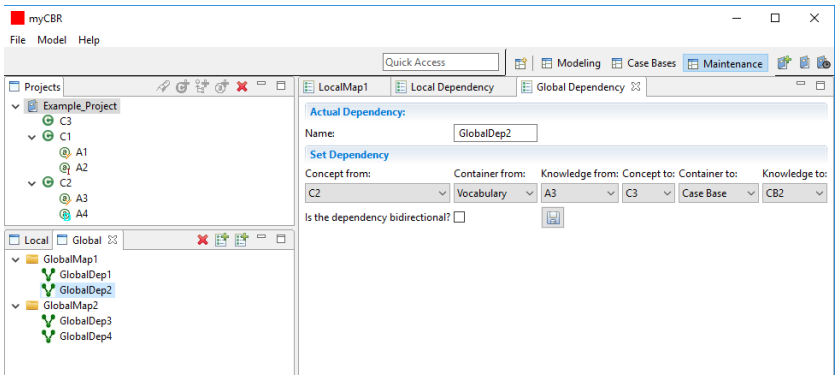

Fig. 2. Maintenance view with dependency

In addition, the workbench has a visualization component to generate an overview of the modeled dependencies. A simple list of dependencies would be adequate for few dependencies, but having dozens or hundreds of dependencies between several CBR systems, simple list would cause high effort to find and edit a specific dependency. Therefore, a graphic representation of the modeled dependencies would provide a better overview. Two different visualizations are currently implemented. Figure 3 shows a simple graph representation of the Maintenance Map. The vertices are arranged in a circle to get clear overview of the edges which represent the dependencies. Figure 4 shows a formatted graph of the Maintenance Map. The different knowledge levels are colored and associated groups of dependencies are displayed together. Vertices with no dependencies have a brighter color. Because the Maintenance Map can be edited

by a user, empty dependencies can be created. An empty dependency is not treated as inconsistent and is represented with a gray vertex in the graph.

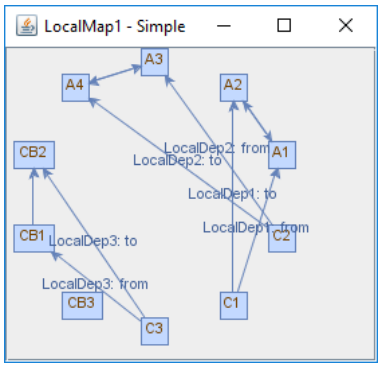

Fig. 3. Simple visualization of dependencies

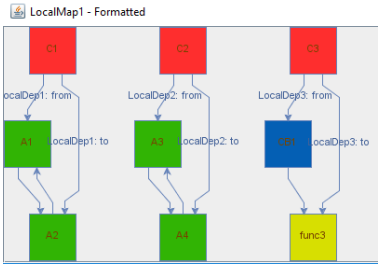

Fig. 4. Formatted visualization of maintenance map

An evaluation of the dependency modeling and the algorithm has not been done yet, because the implementation of the sixth knowledge level has to be completed before we can generate specific dependencies on the value level of the hierarchy.

4 Summary and Outlook

This paper gives an overview of the improvements for the maintenance dependency modeling. We describe the newly defined knowledge levels for CBR systems and how the associated hierarchy is used to model dependencies with different granularity. A formal definition of the improved dependencies is also given. In addition, we describe a first algorithm to generate syntactic dependencies automatically. Finally, we give an overview of the current implementation state of the dependency modeling in myCBR. Currently, we are integrating the knowledge level 6 into myCBR to be able to model all proposed granularities of maintenance dependencies. After the implementation is finished, we will use the improved dependencies in a multi-agent system with different CBR systems and associated Case Factories to evaluate the utility of the improved maintenance dependencies. In addition, we will improve the visualization of the dependencies to be able to show a scrollable visualization with different details for each knowledge level.

References

1. A Domain-Specific Language for Dependency Management in Model-Based Systems Engineering (09/2013 2013)

2. Aamodt, A.: Modeling the knowledge contents of cbr systems. In: Proceedings of the Workshop Program at the Fourth International Conference on Case-Based Reasoning. pp. 32–37 (2001)
3. Al-Natour, S., Cavusoglu, H.: The strategic knowledge-based dependency diagrams: A tool for analyzing strategic knowledge dependencies for the purposes of understanding and communicating. Inf. Technol. and Management 10(2-3), 103–121 (Sep 2009)
4. Bach, K.: Knowledge Acquisition for Case-Based Reasoning Systems. Ph.D. thesis, University of Hildesheim (2012), dr. Hut Verlag Muenchen 2013
5. Bach, K., Sauer, C., Althoff, K.D., Roth-Berghofer, T.: Knowledge modeling with the open source tool mycbr. In: Nalepa, G.J., Baumeister, J., Kaczor, K. (eds.) Proceedings of the 10th Workshop on Knowledge Engineering and Software Engineering (KESE10). Workshop on Knowledge Engineering and Software Engineering (KESE-2014), located at 21st European Conference on Artificial Intelligence, August 19, Prague, Czech Republic. CEUR Workshop Proceedings (http://ceur-ws.org/) (2014)
6. Davenport, T.H., Prusak, L.: Working Knowledge: How Organizations Manage What they Know. Havard Business School Press (2000)
7. Du, J., Bormann, J.: Improved similarity measure in case-based reasoning with global sensitivity analysis: an example of construction quantity estimating. Journal of Computing in Civil Engineering 28(6), 04014020 (2012)
8. Ferrario, M.A., Smyth, B.: Distributing case-based maintenance: The collaborative maintenance approach. Computational Intelligence 17(2), 315–330 (2001)
9. Nick, M.: Experience Maintenance Loop through Closed-Loop Feedback. Ph.D. thesis, TU Kaiserslautern (2005)
10. Patterson, D., Anand, S., Hughes, J.: A knowledge light approach to similarity maintenance for improving case-base competence. In: ECAI Workshop Notes. p. 6578 (2000)
11. Reuss, P., Althoff, K.D.: Explanation-aware maintenance of distributed case-based reasoning systems. In: LWA 2013. Learning, Knowledge, Adaptation. Workshop Proceedings. pp. 231–325 (2013)
12. Reuss, P., Althoff, K.: Maintenance of distributed case-based reasoning systems in a multi-agent system. In: Proceedings of the 16th LWA Workshops: KDML, IR and FGWM, Aachen, Germany, September 8-10, 2014. pp. 20–30 (2014)
13. Reuss, P., Althoff, K.: Dependencies between knowledge for the case factory maintenance approach. In: Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB, Trier, Germany, October 7-9, 2015. pp. 256–263 (2015)
14. Richter, M.M.: The knowledge contained in similarity measure. Invited Talk at the First International Conference on Case-Based Reasoning, ICCBR95 (1995)
15. Richter, M.M.: Handbuch der knstlichen Intelligenz, chap. Fallbasiertes Schlieen, pp. 407–430. Oldenbourg Wissenschaftsverlag (2003)
16. Roth-Berghofer, T.: Knowledge maintenance of case-based reasoning systems. The SIAM methodology. Akademische Verlagsgesellschaft Aka GmbH (2003)
17. Sangal, N., Jordan, E., Sinha, V., Jackson, D.: Using dependency models to manage complex software architecture. In: Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications. pp. 167–176. OOPSLA '05, ACM, New York, NY, USA (2005)
18. Sell, C., Winkler, M., Springer, T., Schill, A.: Two dependency modeling approaches for business process adaptation. In: Proceedings of the 3rd International Conference on Knowledge Science, Engineering and Management. pp. 418–429. KSEM '09, Springer-Verlag, Berlin, Heidelberg (2009)
19. Smyth, B., Keane, M.: 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. pp. 377–382 (1995)

20. Stahl, A.: 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)
21. Stahl, A., Roth-Berghofer, T.: Rapid prototyping of cbr applications with the open source tool mycbr. In: Advance in Case-Based Reasoning, Proceeding of the 9th European Conference on Case-Based Reasoning (2008)
22. Stram, R., Reuss, P., Althoff, K.D., Henkel, W., Fischer, D.: Relevance matrix generation using sensitivity analysis in a case-based reasoning environment. In: International Conference on Case-Based Reasoning. pp. 402–412. Springer (2016)