Dependencies between knowledge for the Case Factory maintenance approach

Pascal Reuss and Klaus-Dieter Althoff
pascal.reuss@dfki.de
klaus-dieter.althoff@dfki.de

Intelligent Information Systems Lab, University of Hildesheim
Competence Center Case Based Reasoning, German Center for Artificial Intelligence, Kaiserslautern

Abstract. In many knowledge-based systems the used knowledge is distributed among several knowledge sources. These knowledge sources may have dependencies between each other, which should be considered when maintaining these sources. An integrated maintenance approach for multiple Case-Based Reasoning (CBR) systems has to consider dependencies between the individual knowledge containers within one CBR system and the dependencies between the knowledge containers of different CBR systems. This paper describes the dependencies between knowledge containers in CBR systems from the perspective of the Case Factory approach and how possible maintenance actions could be derived from these dependencies.

1 Introduction

Today knowledge based systems handling a huge amount of knowledge to provide solutions to given problems. This knowledge is often distributed over several internal or external knowledge sources. These knowledge sources may be independent from each other, but they also may have dependencies between each other. In many systems the knowledge is distributed among sub-domains. For example a travel medicine application may have knowledge divided into knowledge about regions, hospitals and medication. Between the regions and the hospitals existing dependencies, because a hospital is linked to a specific region. If the spelling of the region is changed or the region is deleted, the corresponding hospital can not be found any more and there will be inconsistent knowledge. In the following we assume that all knowledge sources in a knowledge based system are CBR systems. When maintaining an application with several different CBR systems as knowledge sources, it is important to consider the dependencies between the knowledge inside these CBR systems. These dependencies could be between knowledge containers inside a single CBR system and between knowledge containers of different CBR systems. Current maintenance approaches for CBR systems focus on one single CBR system or a single knowledge container and considering

 In: R. Bergmann, S. Görg, G. Müller (Eds.): Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB. Trier, Germany, 7.-9. October 2015, published at http://ceur-ws.org

only dependencies inside a single CBR system. The extended Case Factory approach [10] considers the dependencies between knowledge containers. In this paper we describe the dependencies that could exist between knowledge containers from a Case Factory perspective and how these dependencies could be processed to derive possible maintenance actions. In Section 2 we give an overview of related work to this topis, while Section 3 describes briefly the Case Factory approach and the dependencies between knowledge containers in more detail. In addition, we describe the modeling of dependencies with the help of a Maintenance Map and an algorithm to identify and process dependencies and derive possible maintenance actions. Section 4 gives a short conclusion and an outlook to future work.

1.1 SEASALT architecture

The SEASALT (Shared Experience using an Agent-based System Architecture Layout) architecture is a domain-independent architecture for extracting, analyzing, sharing, and providing experiences [2]. The architecture is based on the Collaborative Multi-Expert-System approach [1] and combines several software engineering and artificial intelligence technologies to identify relevant information, process the experience and provide them via an interface. The SEASALT architecture consists of five components: the knowledge sources, the knowledge formalization, the knowledge provision, the knowledge representation, and the individualized knowledge. The knowledge sources component is responsible for extracting knowledge from external knowledge sources like databases or web pages and especially Web 2.0 platforms. The knowledge formalization component is responsible for formalizing the extracted knowledge from the Collector Agents into a modular, structural representation. The knowledge provision component contains the so called Knowledge Line. The basic idea is a modularization of knowledge analogous to the modularization of software in product lines. The modularization is done among the individual topics that are represented within the knowledge domain. The Topic Agents can be any kind of information system or service. If a Topic Agent has a CBR system as knowledge source, the SEASALT architecture provides a Case Factory for the individual case maintenance [2]. The knowledge representation component contains the underlying knowledge models of the different agents and knowledge sources. The synchronization and matching of the individualized knowledge models improves the knowledge maintenance and the interoperability between the components. The individualized knowledge component contains the web-based user interfaces to enter a query and present the solution to the user.

2 Related work

The DILLEBIS methodology from Markus Nick [8] focuses on identifying necessary maintenance actions using user feedback. He considers dependencies between knowledge sources only implicitly. A dependency can be assumed, if a user advises to change more than one knowledge source in his feedback. A knowledge engineer has to confirm a dependency manually. In our approach we define the dependencies explicitly and

process them automatically to give the knowledge engineer a list of possible maintenance actions. The SIAM methodology from Thomas Roth-Berghofer [11] focuses on maintenance for CBR systems and extends the CBR cycle with to additional steps for evaluation and maintenance of a single CBR system. Dependencies are considered only implicit in this methodology, too. The evaluation of the knowledge containers can show dependencies, if more than one knowledge container requires maintenance in a specific situation. But the confirmation of dependencies had to be done manually before a maintenance action can be performed. There are many maintenance approaches for CBR systems like [5], [6],[12], and [13] that presents strategies to maintain the case base or the similarity measures. But all of these approaches are only considering one knowledge container or a single CBR system, while we will consider dependencies between all knowledge containers of a single CBR system and dependencies between knowledge containers of different CBR systems. Leake and his co-authors worked with different multiple knowledge sources for CBR systems and the combination of maintenance actions to preserve the competency and efficiency of a CBR system [7]. Their approach is focused on a single CBR system, but the idea is also applicable for multiple CBR systems and may be combined with our approach.

3 Dependencies between knowledge containers in CBR systems

In a multi-agent system like *docQuery*, the knowledge is distributed over several knowledge sources. Each knowledge source is a software agent with an underlying CBR system, representing the knowledge of a sub-domain of the travel medicine domain. For example one CBR system contains knowledge about regions, anothr CBR system contains knowledge about medication. In the *docQuery* system exist seven different CBR systems for knowledge about regions, hospitals, medication, infectious diseases, chronicle diseases, activities and conditions (climate, security, etc) [9]. Between these CBR systems dependencies can be found, either because two CBR systems share the same vocabulary or cases are linked to each other. For example, the CBR systems for regions and infectious diseases have partially the same vocabulary and there are links between case from the region case base and the infectious disease case base. These dependencies have to be considered, when thinking about maintaining these CBR systems. The extended Case Factory approach [10] for maintaining CBR systems is able to consider these dependencies. A Case Factory is part of the *knowledge provision* component of the SEASALT architecture and is responsible for maintaining a single CBR system. Several software agents are monitoring the knowledge containers and propose possible maintenance actions, if defined conditions are met. Based on monitoring results and defined dependencies additional possible maintenance actions may be derived. A Case Factory can process the dependencies inside a single CBR system. Following our approach, each of the seven CBR systems has its own Case Factory to monitor and maintain the knowledge. To process dependencies between CBR systems, a so-called Case Factory Organization (CFO) is used. This high-level layer manages all Case Factories, the dependencies between knowledge containers of different CBR systems, and coordinates the maintenance process. There can be more than one CFO to manage the maintenance on different levels. A CFO can be used to split a system with multiple CBR system

into several organizational units. For example in the *docQuery* application it would be possible to have 4 CFOs. One CFO contains the region and hospital CBR systems, the second CFO the infectious diseases, chronicle diseases and medication CBR systems and the third CF contains the activities and conditions CBR systems. Each of this CFOs manage the dependencies between the corresponding CBR systems. The fourth CFO manages the dependencies between CBR system of different CFOs and can also be used to manage the overall maintenance process to identify maintenance actions that have to be processed in combination with other maintenance actions to address problems as stated in [7]. In the following section, the dependencies between knowledge containers from our Case Factory perspective are described in more detail.

3.1 Intra- and inter-system dependencies

A dependency exists between different knowledge containers. We define a dependency d as

$$d = (kc_{sysS}, kc_{sysT}, t)$$

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

A dependency can be described as a triple of two knowledge containers (*kc*) and the direction (*t*) of the dependency. The knowledge containers are the vocabulary (*voc*), the similarity measures (*sim*), the case base (*cb*), and the adaptation knowledge (*ada*). We assume there are 1 to n CBR systems. The indexes *sysS* and *sysT* identify the CBR systems a knowledge container belongs to, where *sysS* is the source of a dependency and *sysT* the target. The last element of the triple determines the direction of a dependency, either uni-directional (*u*) or bi-directional (*b*). A uni-directional dependency is only processed from the source knowledge container to the target knowledge container, while for a bi-directional dependency both directions have to be considered when deriving possible maintenance actions. From our Case Factory perspective two different categories of dependencies, intra-system and inter-system dependencies. Intra-system dependencies exist between different knowledge containers of the same CBR system, while inter-system dependencies exist between knowledge containers of different CBR systems. Distinguishing between intra- and inter-system dependencies is important for processing the dependencies. An intra-system dependencies can be processed by the corresponding Case Factory itself. If no dependencies points to another CBR system, there is no need to propagate the dependencie to the CFO.

An intra-sytem dependency is defined as follows:

$$d_{intra} = (kc_{sysS}, kc_{sysT}, t)$$

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

while an inter-system dependency is defined as follows:

$$d_{inter} = (kc_{sysS}, kc_{sysT}, t)$$

$$\text{where } kc \in \{voc, sim, cb, ada\}$$
$$\text{and } sysS, sysT \in \{1 \dots n\} \text{ and } sysS \neq sysT$$
$$\text{and } t \in \{u, b\}$$

There are three intra-system dependencies that could be called trivial dependencies. These trivial dependencies exist between the vocabulary and the other three knowledge containers and are uni-directional. The trivial dependencies are uni-directional, because the vocabulary sets the surrounding conditions of the other knowledge containers: changing the name of an attribute or its value range or creating a new concept for a taxonomy has to be done in the vocabulary and has then an effect on the other knowledge containers. Therefore the dependencies is only pointing from the vocabulary to the other knowledge containers and not backwards, too. These dependencies describe the fact that a change in the vocabulary has a direct impact on the other knowledge containers in the same CBR system. These trivial dependencies are defined per default for every CBR system and are defined as follows:

$$d_{triv} = (voc_{sysS}, sim_{sysT}, u) \quad \text{where } sysS, sysT \in \{1 \dots n\} \text{ and } sysS = sysT$$
$$d_{triv} = (voc_{sysS}, cb_{sysT}, u) \quad \text{where } sysS, sysT \in \{1 \dots n\} \text{ and } sysS = sysT$$
$$d_{triv} = (voc_{sysS}, ada_{sysT}, u) \quad \text{where } sysS, sysT \in \{1 \dots n\} \text{ and } sysS = sysT$$

3.2 Dependency modeling in a Maintenance Map

The dependencies between knowledge containers have to be defined by a knowledge engineer. The construct to store the modeled dependencies is a so-called Maintenance Map. The Maintenance Map is based on the Knowledge Map from Davenport and Prusak [4] and was adapted to multi-agent systems by Bach et al. [3]. A Maintenance Map can be represented as a bi-directional graph. The vertices represent knowledge sources, for example a CBR system, and the edges the dependencies between these knowledge sources. There are also loop edges from a vertex to itself to represent the trivial dependencies and it is possible to have multiple edges between two vertices to represent dependencies between multiple knowledge containers of CBR systems. In addition, the edges could be weighted to describe the importance of a dependency. The following figure 1 shows the Maintenance Map for the *docquery* application as a graph. There are dependencies between the vocabularies and the case bases for each CBR system and the number on the edges represent the importance of the dependencies.

Inside the Maintenance Map, the dependencies are modeled in RDF language to simplify the interchange of the Maintenance Map between MAS with multiple CBR systems. In the following we will describe an example based on the docQuery multi-agent system to show the modeling of dependencies:

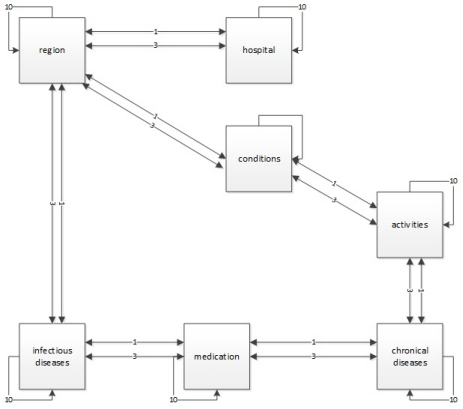

Fig. 1. Maintenance Map for the docquery application as graph

Listing 1.1. Exerpt from a Maintenance Map of the docQuery application

```
<rdf:Description rdf:about="'dependency1"'>
        <dep:kcsource>vocabulary</dep:kcsource>
        <dep:kctarget>vocabulary</dep:kctarget>
        <dep:cbrsource>DQ_region</dep:cbrsource>
        <dep:cbrtarget>DQ_hospital</dep:cbrtarget>
        <dep:type>bidirectional</dep:type>
        <dep:weight>1</dep:weight>
</rdf:Description>
```

For every dependency the required attributes are modeled in RDF language. The knowledge containers are set with the attributes *kcsource* and *kctarget*, while the CBR systems are set with *cbrsource* and *cbrtarget*. The attribute *type* determines whether a dependency is uni-diretional or bi-directional and the *weight attribute* defines the importance. In this example, the first dependency is an inter-system dependency between the CBR system for region information and the CBR system for hospital information. We have a dependency between the vocabularies of both CBR systems, because several attributes of the different case structures use the same vocabulary. The attribute values for the name of the region in the region CBR system and the region part of the hospitals address are the same. A change of a region's name in the first CBR system has to lead to a change of the same region's name in the hospital CBR system. This way inconsistencies in the knowledge should be avoided. The second dependency is an intra-system and trivial dependency. It exists between the vocabulary and the case base of the region CBR system. Changing the vocabulary may lead to a change of attribute values in one or more cases. This dependency is uni-directional, because an attribute value in a case can only be set after it is defined in the vocabulary. In addition, the Maintenance Map could contain information about preferred maintenance actions for knowledge containers based on the dependencies and required combinations of maintenance actions to preserve the problem solving competence. Information about evaluation strategies for the CBR systems and knowledge containers can be stored, too.

3.3 Deriving maintenance actions from dependencies

After defining dependencies for multiple CBR systems in a multi-agent system, these dependencies are used to derive possible maintenance actions to keep the knowledge in all CBR systems consistent. Each Case Factory derives possible maintenance actions for the assigned CBR system based on intra-system dependencies and the Case Factory Organization derives possible maintenance actions based on inter-system dependencies. In the following we present an algorithm on an abstract level to derive possible maintenance actions based on given dependencies. A maintenance action for this algorithm is defined as a change on a knowledge container *changeKC*. *changeKC(*$d.kc_{sysS}$*)* is a function that changes the knowledge container given as a parameter.

Listing 1.2. Algorithm to derive maintenance actions

```
Input:
D Set of given dependencies (intra- or intersystem)
M Set of initial maintenance actions
Output:
M_p Set of proposed maintenance actions

M_p = M
while (M not empty)
 for (m in M) {
        for (d in D) {
                if (d.kc_sysS == m.kc_sysS
                OR (d.kc_sysT == m.kc_sysS AND d.t == b)) {
                        if (!M_p.contains(changeKC(d.kc_sysT)) {
                                M_p.add(changeKC(d.kc_sysT))
                                M.add(changeKC(d.kc_sysT))
                        }
                }
                M.remove(m)
        }
 }
return M_p
```

The algorithm requires a set of defined dependencies D and a set of initial maintenance actions M as input. If M is empty, the algorithm terminates, because no starting point for the algorithm would be given. The output of the algorithm is a set of possible maintenance actions that could be proposed to the knowledge engineer. At first, the initial set of maintenance actions will be added to M_p, because these maintenance actions should be proposed, too. The condition for the while loop is that no new maintenance actions could be derived, so no more dependencies have to be considered. The inner loops process all defined dependencies and the initial and derived maintenance actions. If a new maintenance action is derived, it is added to M and M_p. A new maintenance action added to M leads to another cycle of the inner loop to determine if further dependencies fire for the new maintenance action. And the new maintenance actions is added to M_p to be proposed to the knowledge engineer. Two conditions are responsible for deriving new maintenance actions: If the source knowledge container of a maintenance action is the same as the source knowledge container of a dependency OR if the dependency is bi-directional and the source knowledge container of the maintenance action is the same as the target knowledge container of the dependency. If one condition is met, a new maintenance action is derived and added to the sets. A maintenance action can only be in a set once. After processing all dependencies for a maintenance action, this maintenance action is removed from M. This is necessary to have an empty list after processing all maintenance actions and dependencies.

4 Summary and Outlook

In this paper we describe the dependencies between knowledge containers of CBR systems from a Case Factory perspective to use them to derive possible maintenance actions. We describe the categories and elements of a dependency and show how defined dependencies could be modeled with the help of a Maintenance Map. In addition, we present an algorithm to use these dependencies to derive possible maintenance actions. The next steps in our work are to define and model all dependencies in our docQuery multi-agent sytem and detail, implement and test the algorithm to derive maintenance actions. Therefore, the possible maintenance actions and their combinations have to be defined. Based on the result of our evaluation, we will revise our algorithm and dependency modeling.

References

1. Althoff, K.D.: Collaborative multi-expert-systems. In: Proceedings of the 16th UK Workshop on Case-Based Reasoning (UKCBR-2012), located at SGAI International Conference on Artificial Intelligence, December 13, Cambride, United Kingdom. pp. 1–1 (2012)
2. Bach, K.: Knowledge Acquisition for Case-Based Reasoning Systems. Ph.D. thesis, University of Hildesheim (2013), dr. Hut Verlag Mnchen
3. Bach, K., Reichle, M., Reichle-Schmehl, A., Althoff, K.D.: Implementing a coordination agent for modularised case bases. In: Proceedings of the 13th UK Workschop on Case-Based Reasoning. pp. 1–12 (2008)
4. Davenport, T.H., Prusak, L.: Working Knowledge: How Organizations Manage What they Know. Havard Business School Press (2000)
5. Ferrario, M.A., Smyth, B.: Distributing case-based maintenance: The collaborative maintenance approach. Computational Intelligence 17(2), 315–330 (2001)
6. Iglezakis, I., Roth-Berghofer, T.: A survey regarding the central role of the case base for maintenance in case-based reasoning. In: ECAI Workshop Notes. pp. 22–28 (2000)
7. Leake, D., Kinley, A., Wilson, D.: Learning to integrate multiple knowledge sources for case-based reasoning. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence. pp. 246–251. Morgan Kaufmann (1997)
8. Nick, M.: Experience Maintenance Loop through Closed-Loop Feedback. Ph.D. thesis, TU Kaiserslautern (2005)
9. Reuss, P.: Concept and implementation of a Knowledge Line - retrieval strategies for modularized, homogeneous topic agents within a multi-agent-system (in German). Master's thesis, University of Hildesheim (2012)
10. Reuss, P., Althoff, K.D., Henkel, W., Pfeiffer, M.: Case-based agents within the omaha project. In: Case-based Agents. ICCBR Workshop on Case-based Agents (ICCBR-CBR-14) (2014)
11. Roth-Berghofer, T.: Knowledge maintenance of case-based reasoning systems. The SIAM methodology. Akademische Verlagsgesellschaft Aka GmbH (2003)
12. 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)
13. 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)