ProCAKE: A Process-Oriented Case-Based Reasoning Framework

Ralph Bergmann, Lisa Grumbach, Lukas Malburg, Christian Zeyen

Business Information Systems II, University of Trier, 54286 Trier, Germany
{bergmann,grumbach,malburgl,zeyen}@uni-trier.de
http://www.wi2.uni-trier.de

Abstract This paper presents ProCAKE – the process-oriented case-based knowledge engine of the CAKE framework, which has evolved from several research projects at the University of Trier over the years. ProCAKE constitutes a domain-independent framework that can be used to implement diverse structural or process-oriented case-based reasoning applications for integrated process and knowledge management. This paper gives an overview of the main components and demonstrates their application by examples.

Keywords: Knowledge Management · Process Management · Case-Based Reasoning

1 Introduction

Nowadays, workflow technology is widely used in business, scientific, and even private domains. However, implementing the technology involves a significant amount of knowledge, which is why integrated process and knowledge management is of great importance. *Process-Oriented CBR (POCBR)* particularly addresses this integration by applying and extending CBR methods for process and workflow management [6]. In general, many research prototypes were presented in the field of CBR but only few frameworks are publicly available for developing CBR applications. Recent frameworks such as *myCBR* [1,8] or *jColibri* [7] mainly focus on structural and textual CBR. We are not aware of any framework that is particularly tailored to the development of POCBR applications. To this end, we present ProCAKE, a generic framework for building structural and process-oriented CBR applications. The software is developed at the *Department of Business Information Systems II* at the University of Trier. The source code is freely available on our website[1]. ProCAKE constitutes the core of the CAKE framework [3] and builds the foundation for various past and ongoing research activities of our research group. The developed prototypes include a broad range of algorithms for retrieval and adaptation [5].

[1] See http://procake.uni-trier.de

2 Architecture Overview

The ProCAKE framework is written in Java while it uses XML for configuration and persistence. It has a pattern driven architecture and relies heavily on interfaces and factories. ProCAKE provides its own data type structure (also referred to as system classes) for defining the domain model and cases. The most commonly used types are:

Base Classes The base classes include *Atomic* classes such as *Boolean*, *Numeric*, *String*, *Chronologic*, and *Void*. *Atomic* classes can be combined with composite classes such as *Aggregate*, *Interval*, or *Collection*. By this means, cases for structural CBR can be represented.

NEST Classes The *NESTGraph* class is a specific composite class for representing workflows or processes as semantic graphs (cf. [4]). It encloses further composite classes for representing the graph elements. To name just a few, graph edges are represented by *NESTPartOfEdge*, *NESTControlflowEdge*, and *NESTDataflowEdge* classes and graph nodes are represented by *NESTWorkflowNode*, *NESTTaskNode*, and *NESTDataNode*, respectively. Graph element classes are linked to user classes that describe their semantics using custom composite classes.

ProCAKE implements many syntactic and semantic similarity measures for the various data types. Most of the measures are formally described in [2]. For example, measures for *Numeric* classes use linear, exponential, and threshold functions whereas measures for *String* classes apply Levenshtein or regular expressions. Several taxonomic measures exist for semantic similarity assessment. For the *NESTGraph* class, a measure is implemented that performs graph-matching with an A* search algorithm [4]. Several algorithms for retrieval are implemented: Besides a k-NN retrieval, several MAC/FAC approaches and an A* parallel retriever are implemented for accelerating retrieval with large case bases. An adaptation framework enables the integration and application of domain-dependent adaptation methods.

Both the user classes and similarity measures can be specified via XML. The core components for instantiating ProCAKE are:

Configuration A benefit of the pattern driven architecture of ProCAKE is the extensibility. For instance, the various implementations for persistence, retrieval, and adaptation are organized in factories. Implementations can be registered and configured in an XML file (usually named composition.xml) that is read once at start-up.

Data Model In addition to the system classes, custom-structured classes (referred to as user classes) can be defined as sub-types of system classes. At start-up, the system reads the model configuration files. Even though several custom models can be defined, the usual practice is to use a single model definition as default (usually named model.xml).

Similarity Model To allow for comparing system and user classes, similarity measures have to be defined for each class. Analogous to the data model, several similarity models can be defined while usually a single model (named sim.xml) is used as default. A similarity measure can be selected as default for a class, so that the measure is applied for all sub-classes of that class. It must be ensured that a similarity measure is defined for each system and user class.

Objects Analogous to the Java perspective, objects in ProCAKE represent the concrete data. However, object classes are used to explicitly model the relationship to the respective system and user classes. Every data used in a case or query for retrieval has to be represented by such objects. Consequently, the presented data type structure represents all possible data types ProCAKE can operate on.

3 Example Application

In the following, we demonstrate an exemplary application of ProCAKE. For this purpose, we consider cooking recipes as a simple form of workflows. The case base comprises 40 sandwich recipes[2]. A cooking workflow is represented as a semantic graph in which each node is associated with a semantic description (cf. Fig. 3). Preparation steps are represented as task nodes and ingredients are represented as data nodes. A workflow node represents general information about the recipe.

```
<AggregateClass name=”WorkflowSemantic” superClass=”Aggregate”>
    [...]
    <Attribute name=”preparation time (min)” class=”PreparationTimeType”/>
    <Attribute name=”calories” class=”CaloriesType”/>
</AggregateClass>

<AtomicClass name=”CaloriesType” superClass=”Integer”>
    <ValueInterval lowerBound=”0” upperBound=”1000”/>
</AtomicClass>

<AggregateClass name=”DataSemantic” superClass=”Aggregate”>
    <Attribute name=”name” class=”IngredientType”/>
    <Attribute name=”amount” class=”AmountType”/>
</AggregateClass>

<AggregateClass name=”AmountType” superClass=”Aggregate”>
    <Attribute name=”value” class=”ValueType”/>
    <Attribute name=”unit” class=”UnitType”/>
</AggregateClass>

<AtomicClass name=”ingredientType” superClass=”String”>>
    <ValueEnumeration>
        <TaxonomyOrder name=”ingredientTaxonomy”>
            <Node v=”ingredients”>
                <Node v=”cheese”>
                    <Node v=”mozzarella”/>
                    <Node v=”parmesan”/>
```

Fig. 1. Excerpt from the data model configuration file

[2] Recipes were extracted manually from https://allrecipes.com

For the semantic descriptions of the graph nodes, we define specific attributes with help of user classes in the data model (see excerpt in Fig. 1). The semantics of the workflow node is expressed by a class named *WorkflowSemantic*. It includes attributes such as *name* of type *string* or *preparation time* and *calories*, both of type *integer* with lower and upper bound (see Fig. 1, lines 1–9). Data nodes are described by a class named *DataSemantic* that includes attributes *name* and *amount* (see lines 11–14). The latter is further divided into *value* and *unit* (see lines 16–19). *Unit* can be one of predefined strings and *value* is an integer. Possible values for the *name* attribute are also predefined and taxonomically ordered (see lines 21ff.). The description of task nodes consists of a string object called *name* whose values are also taxonomically ordered.

```
 1   <AggregateAverage name=”DefaultMeasureWorkflowSemantic”
 2     class=”WorkflowSemantic”>
 3       <AggWeight att=”name” weight=”1” />
 4       <AggWeight att=”preparation time (min)” weight=”1” />
 5       <AggWeight att=”calories” weight=”1” />
 6   </AggregateAverage>

 8   <NumericLinear name=”DefaultMeasureCaloriesType” class=”CaloriesType”/>

10   <AggregateAverage name=”DefaultMeasureDataSemantic” class=”DataSemantic”>
11       <AggWeight att=”name” weight=”2” />
12       <AggWeight att=”amount” weight=”1” />
13   </AggregateAverage>

15   <AggregateMinimum name=”DefaultMeasureAmountType” class=”AmountType”/>

17   <StringEqual name=”DefaultMeasureUnitType” class=”UnitType”
18     caseSensitive=”false”/>

20  <TaxonomyClassicUserWeights name=”defaultMeasureIngredientType”
21    class=”ingredientType” order=”ingredientTaxonomy”
22    innerNodeInQueryStrategy=”optimistic” innerNodeInCaseStrategy=”optimistic”>
23       <Node value=”ingredients” weight=”0.001”/>
24       <Node value=”cheese” weight=”0.5”/>
```

Fig. 2. Excerpt from the similarity model configuration file

On the basis of this model, the similarity model (see excerpt in Fig. 2) defines similarity measures for comparing the objects. Similarity measures are determined for local attributes as well as for aggregate objects and whole workflow graphs. For attributes like *name*, *preparation time*, or *calories*, we apply simple measures such as Levenshtein distance for strings or a linear function for numeric values (see Fig. 2, line 8). Ingredient types are compared on the basis of the given taxonomy order and manually annotated similarity values (see lines 20ff.). The similarity of aggregate objects is mostly computed by a weighted average function (see lines 1–6 and 10–13). For the attribute *amount*, we apply the aggregate minimum (cf. line 15), because if the unit types are not equal (the similarity is 0), the value is not directly comparable. In this event, the minimum function ensures that the similarity of the value attribute is ignored. To compute the similarity of the workflow graphs, we apply the A* similarity measure.

An exemplary query and the corresponding local similarities to an example case are depicted in Fig. 3. Workflow nodes are represented as rhombuses, task nodes as rectangles, and data nodes as ovals. Semantic descriptions of the nodes are written in grey rectangles. Solid edges with description *po* indicate part-of edges whereas *cf* and *df* denote control-flow and data-flow edges, respectively.

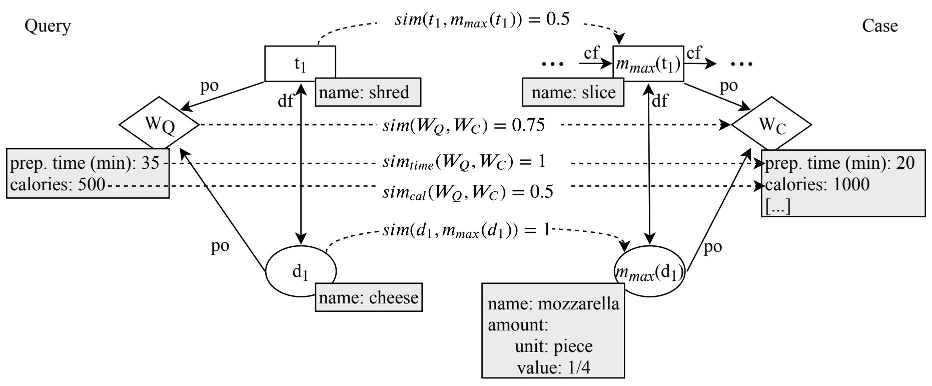

Fig. 3. Exemplary graph query and case

The query consists of three constraints: The desired *preparation time* is set to 35 minutes, whereas the desired amount of *calories* is set to 500. This information is annotated at the workflow node W_Q. Furthermore, shredded cheese is desired, which is represented by the partial workflow containing a data node *cheese* (d_1) as input to the task *shred* (t_1). The query graph can be used as input for retrieving suitable workflow graphs from the case base. For determining the similarity between two workflow graphs, an A* algorithm searches for the best mappings between the nodes and edges. Each mapping is rated with a similarity. Figure 3 depicts the similarities (dashed arrows) of the best possible mappings (m_{max}) between nodes. Please note that edges are also mapped and rated with similarities between the graphs. The global similarity of the workflows is obtained by aggregating the local similarities between the mappings of all elements of the query graph to the elements of the case graph.

The similarities of mapped elements are computed locally by comparing the semantic descriptions. In the given example, the data nodes are considered to be equal ($sim(d_1, m_{max}(d_1)) = 1$), as *mozzarella* is a child node of *cheese* in the taxonomy and further semantic attributes are not given in the query. The local similarity of the task nodes, here *shred* and *slice*, originates from the given values in the taxonomy. In the example, we assume that the common parent node is annotated with a similarity value of 0.5, i.e., $sim(t_1, m_{max}(t_1)) = 0.5$. The similarity of the workflow nodes is composed of an aggregated average of single similarity values for *preparation time* and *calories* attributes. The *preparation time* given in the query should not be exceeded. Thus, we apply a threshold measure, which sets the similarity value to 0, if the limit is reached. Since *preparation time* of the case is lower than that of the query, the obtained similarity is 1. To compute the similarity of the attribute *calories*, we use a linear numeric measure, leading to a similarity of 0.5. In conclusion, the overall similarity of the workflow nodes results from the sum of single weighted similarities: $sim(W_Q, W_C) = 0.5 * sim_{time}(W_Q, W_C) + 0.5 * sim_{cal}(W_Q, W_C) = 0.5 * 0.5 + 0.5 * 1 = 0.75$.

4 Future Work

We are continuously improving and extending features and the documentation. Our goal is to integrate domain-independent algorithms into the generic ProCAKE framework. For instance, current work focuses on transferring generalization and specialization methods for domain-specific workflow representations to arbitrary user classes. The latest version of ProCAKE already includes many features of our research prototypes. It particularly provides various similarity measures, several retrieval methods, and a generic adaptation manager. The example application of ProCAKE is publicly available to foster the implementation of new applications. To further facilitate the development, we are currently working on a graphical tool for visualizing and creating workflow graphs. We appreciate any suggestions or feedback and are open to extensions.

Acknowledgements. This work is funded by the German Research Foundation (DFG) under grant no. BE 1373/3-3.

References

1. Bach, K., Althoff, K.: Developing case-based reasoning applications using mycbr 3. In: Case-Based Reasoning Research and Development - 20th Int. Conf., ICCBR 2012. Proceedings. LNCS, vol. 7466, pp. 17–31. Springer (2012)
2. Bergmann, R.: Experience Management: Foundations, Development Methodology, and Internet-Based Applications, LNCS, vol. 2432. Springer (2002)
3. Bergmann, R., Gessinger, S., Görg, S., Müller, G.: The Collaborative Agile Knowledge Engine CAKE. In: Proc. of the 18th Int. Conf. on Supporting Group Work, 2014. pp. 281–284. ACM (2014)
4. Bergmann, R., Gil, Y.: Similarity assessment and efficient retrieval of semantic workflows. Information Systems **40**, 115–127 (2014)
5. Bergmann, R., Minor, M., Müller, G., Schumacher, P.: Project EVER: Extraction and Processing of Procedural Experience Knowledge in Workflows. In: Proc. of IC-CBR 2017 Workshops. CEUR Proc., vol. 2028, pp. 137–146. CEUR-WS.org (2017)
6. Minor, M., Montani, S., Recio-García, J.A.: Process-Oriented Case-Based Reasoning. Information Systems **40**, 103 – 105 (2014)
7. Recio-García, J.A., González-Calero, P.A., Díaz-Agudo, B.: jcolibri2: A framework for building Case-based reasoning systems. Sci. Comput. Prog. **79**, 126–145 (2014)
8. Stahl, A., Roth-Berghofer, T.: Rapid Prototyping of CBR Applications with the Open Source Tool myCBR. In: Advances in Case-Based Reasoning, 9th European Conf, ECCBR 2008, Proceedings. LNCS, vol. 5239, pp. 615–629. Springer (2008)