On-demand and Model-driven Case Building Based on Distributed Data Sources

Mark van der Pas[1,2][0000−0001−6091−9340], Remco Dijkman[1][0000−0003−4083−0036], Alp Akçay[1][0000−0003−2000−6816], Ivo Adan[1][0000−0002−4493−6367], and John Walker[2]

[1] Eindhoven University of Technology, Department of Industrial Engineering & Innovation Sciences, Eindhoven 5600MB, The Netherlands
[2] Semaku B.V., Eindhoven 5617BC, The Netherlands
m.c.a.v.d.pas@tue.nl

Abstract. The successful application of Case-Based Reasoning (CBR) depends on the availability of data. In most manufacturing companies these data are present, but distributed over many different systems. The distribution of the data makes it difficult to apply CBR in real-time, as data have to be collected from the different systems. In this work we propose a framework and algorithm to efficiently build a case representation on-demand and solve the challenge of distributed data in CBR. The main contribution of this work is a framework using an index for objects and the sources where data about those objects can be found. Next to the framework, we present an algorithm that operates on the framework and can be used to build case representations and construct a case base on-demand, using data from distributed sources. There are several parameters that influence the performance of the framework. Accordingly, we show in a conceptual and experimental evaluation that in highly-distributed and segregated environments the proposed approach reduces the time complexity from polynomial to linear order.

Keywords: CBR frameworks · Case representation · Case base building · Distributed systems · Industry 4.0 · Semantic Web

1 Introduction

One of the challenges identified for Case-Based Reasoning (CBR) research is the acquisition of cases from raw data [11]. In complex manufacturing settings numerous different systems are used, that typically operate independently, in which these raw data are stored. Especially in Industry 4.0 with the digitization of individual assets, for example using the Asset Administration Shell (AAS) [20], data will be highly distributed. On top of that data from multiple companies along the supply chain might be required. This distribution poses a challenge of collecting the data from the relevant sources. One solution to make these distributed data available for CBR is to push all data to the CBR system at the moment they are generated. However, this means duplicating large amounts of data, the majority of which might not be relevant. Duplicating data can be avoided by only storing the data in the source systems and collecting the data on-demand, i.e. during case retrieval. The challenge in on-demand data collection is to minimize the time it takes to collect the data and construct the case base. A good example where those problems are seen is the handling of manufacturing quality incidents [4]. Those incidents are related to a small fraction of produced objects and data from many different systems is required to analyze them. Generating a case representation in such scenarios often requires significant manual effort and is based on a fixed structure, see for example [8].

To solve the challenge of distributed data, we propose a framework in which only information about what sources contain data about what objects is stored centrally. Specifically, all relevant objects are indexed with pointers to the sources where data about those objects can be found. The case model (also referred to as case structure or vocabulary), which defines the object classes as well as the properties (relations and attributes) required to describe a case, can be used to select the relevant objects. We identify multiple dimensions that impact the time complexity of the problem. In both a conceptual and experimental evaluation we show how the proposed algorithm performs compared to an approach with no index, taking into account the identified dimensions. Finally, we demonstrate that the proposed framework and algorithm reduce the time complexity significantly and enable real-time building of case representations.

The remainder of this work is structured as follows. First, we will introduce the related work in Section 2. Subsequently, Section 3 describes the proposed method in more detail, followed by a conceptual evaluation in Section 4. Section 5 presents our implementation based on Semantic Web Technologies and Industry 4.0 concepts, including an experimental evaluation of its performance. The last sections contain a discussion, a conclusion with a summary of the findings and ideas for future work.

2 Related Work

In this section we look into related research on distributed systems in CBR and approaches for collecting data from distributed sources.

One of the architectural patterns in CBR is distributed case-based reasoning [21]. In distributed CBR systems the cases are stored in multiple distributed sources (knowledge bases). Distributed CBR systems are applied to several domains. Pla et al. [19] propose a CBR system for medical diagnosis, whereas Tran et al. [27] for fault management in communication networks. On the other hand Clood CBR [18], SEASALT [1,2], jcolibri2 [23] and F-CBR [16] are domain independent frameworks for sharing and retrieving knowledge. The frameworks use a modular and agent- or microservices-based design, where the cases are distributed over multiple case bases and the main challenge is to retrieve similar cases from those distributed bases. In contrast, we consider a situation where the data for one case is split over multiple sources and the main challenge is to collect the data from the distributed sources, build the case representation(s) and construct the case base in an efficient way.

Most work on collecting data from distributed sources is about the development of efficient federated query engines. The main challenge here is to efficiently query multiple (distributed) data sources. The engines operate over distributed servers that expose data. The main engines developed are FedX [25], SPLENDID [12], and SemaGrow [9]. All of them use information and metadata about the federated data sources to optimize the query plan. The generation of the query plan introduces quite some overhead, which limits the performance of the engines. The federated query engines are often optimized to execute analytic style queries that operate over large parts of the data set. However, in CBR we are often interested in only a small fraction of the data set for describing cases.

Verborgh et al. [28] propose a framework for more efficient querying of distributed sources, that aims to balance the costs between the client and server. Different link-based traversal algorithms are proposed for querying those sources [26,28]. The main idea is to iterate through the sources to discover the data matching query patterns. The query engine does not know beforehand what objects are present in what sources, therefore all sources have to be requested, which will limit the performance when there are many sources. In comparison to our approach, federated query engines support more complex queries, but this comes at the cost of more complex and less efficient data collection.

An alternative to the federated query engines is Linked Data crawling [10]. Those crawling approaches assume that every source contains pointers to the other sources where data about certain objects can be found, such that software agents can autonomously query and discover the data sources. In large manufacturing organizations there are often many different legacy and siloed systems, that do not have the required pointers to other systems.

3 On-demand and Model-driven Case Building

In Section 3.1 the conceptual framework that supports on-demand case building is described along with the notation that is used throughout the paper. This is followed by a description of the algorithm in Section 3.2.

3.1 The Framework

Figure 1 gives an overview of the framework and notation we use to refer to parts of the framework, which is further elaborated in Table 1. The framework consists of three parts: a set of data sources, a case data model and an index. We will discuss the parts one by one. The data sources are the distributed systems where data about objects is stored that are relevant for a case. These sources can be large databases that contain data about many objects, or systems that serve data for a small set of objects. In a manufacturing setting an object can be a specific machine and data about this machine can be stored on some local server. The case representation model or vocabulary defines what type of data

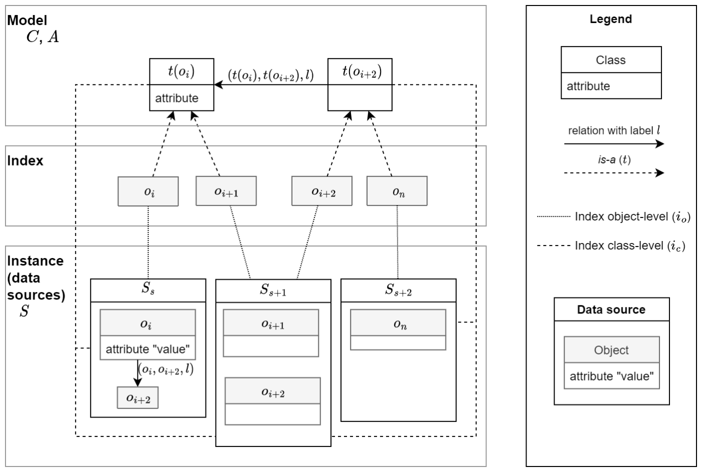


Fig. 1. Framework and notation.

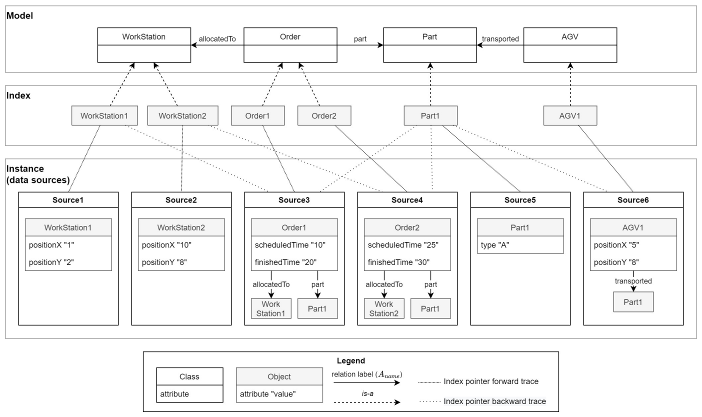


Fig. 2. Illustration of the framework based on an example.

(classes, relations and attributes) should be used to describe a case and defines the structure of the case base. In this work we consider an object-oriented case representation [5,6] that consists of a collection of objects that belong to a certain class, are described by a set of attribute-value pairs, and are related to each other by a set of relations. Note that relations are a special case of attribute-value pairs with a value referring to another object. Because the relations play an important role in the framework, we distinguish the relations from the simple attribute-value pairs (that have a literal value). For example, in a manufacturing setting with quality incidents the case model might consist of the classes ‘product’ and ‘machine’, with relation ‘produced on’. In the remainder of this work we will ignore the simple attributes. We assume that every object together with its simple attribute-value pairs can be retrieved in one request and therefore do not impact the complexity of the problem. The last component of the framework is the index that relates the data model to the data sources and can be used for efficient on-demand building of the case base according to the case model with data from the distributed sources. For that purpose the index will contain pointers to the data sources, where we consider indexing the sources on object- or class-level. The object-level index contains pointers for every object, while the class-level index only contains pointers for every class in the model. An example of the framework applied to a typical manufacturing setting can be found in Figure 2. Generating and updating the index is implementation specific. In Section 5.1 we describe the Industry 4.0 and Semantic Web [7] technologies that are used in our implementation.

Table 1. Overview of notation

Model	
C	set of classes
L	set of relation labels
$A \subseteq C \times C \times L$	set of (directed) relations between classes (C) with label (L)
Instance	
O	set of objects
$t : O \rightarrow C$	type of object (assignment to a class)
$E \subseteq O \times O \times L$	set of relations between objects (O) with label (L) if $(o, m, l) \in E$, then $(t(o), t(m), l) \in A$
$S = \{S_1, ..., S_s\}$, where $S_i \subseteq E$	set of data sources with relations stored in every source
Index	
$i_c : C \rightarrow \mathcal{P}(S)^3$	pointer from class to source(s) where data about objects of that class can be found
$i_o : O \rightarrow \mathcal{P}(S)^3$	pointer from object to source(s) where data about that object can be found

3.2 Algorithm

The next step is the definition of the algorithm to construct a case base with data from distributed sources. The index is used to target the relevant data sources to collect data from, which will reduce the number of requests required for the construction. Algorithm 1 takes as arguments the set of objects of interest to build a case representation for (O_{query}) and the case model, consisting of classes ($C_{case} \subseteq C$) and relations ($A_{case} \subseteq A$). Starting from the objects of interest, the algorithm iterates over the objects it encounters until no new objects of interest for the case base are found (line 7). For each object it encounters (line 10), all sources where data about those objects are stored (line 11) have to be checked. The relevant sources can be retrieved from the index, in Algorithm 1

[3] $\mathcal{P}(S)$ denotes the power set of S.

Algorithm 1 Model-driven case building

```
1: Input
2:     $O_{query}$                                  ▷ set of objects of interest
3:     $C_{case}$                                   ▷ set of classes in the case model
4:     $A_{case}$                                   ▷ set of relations in the case model
5: $O_{base} \leftarrow \emptyset$
6: $O_{next} \leftarrow O_{query}$
7: while $|O_{next}| \neq \emptyset$ do
8:     $O_{selected} \leftarrow O_{next} - O_{base}$
9:     $O_{next} \leftarrow \emptyset$
10:    for all $o \in O_{selected}$ do
11:        for all $S_k \in i_o(o)$ do
12:            $O_{forward} \leftarrow \{m|(o, m, l) \in S_k \wedge t(m) \in C_{case} \wedge (t(o), t(m), l) \in A_{case}\}$
13:            $O_{backward} \leftarrow \{m|(m, o, l) \in S_k \wedge t(m) \in C_{case} \wedge (t(m), t(o), l) \in A_{case}\}$
14:            $O_{next} \leftarrow O_{next} \cup O_{forward} \cup O_{backward}$
15:        end for
16:        $O_{base} \leftarrow O_{base} \cup \{o\}$
17:    end for
18: end while
19: return $O_{base}$
```

the object-level index is used, so it checks what sources are in the index for an object o ($i_o(o)$). Alternatively, the class-level index can be used, in that case the sources are selected by checking the sources for the class that an object belongs to ($i_c(t(o))$). For a given source, the next objects to explore are retrieved by checking the relations to other objects that are stored in the source. For forward trace the relations are selected where o is the origin (line 12), thus following the direction of the relations. Following the relations in the opposite direction, with o as the target (line 13), we refer to as backward trace. Furthermore, only relations that are present in the case model (A_{case}) and objects that belong to a class in the case model ($t(m) \in C_{case}$) are of interest. Both the objects retrieved in forward and backward trace are selected for the next iteration. The last step is to add the object (and its attribute-value pairs) to the case base on line 16. Once the algorithm terminates it returns the constructed case base (O_{base}), which is a union of the case representation for the objects of interest: $O_{base} = \bigcup_{o \in O_{query}} c_o$, where c_o is the set containing the representation of one case for an object o.

For example, suppose there was some incident with `Order1` from Figure 2 and we need to build a case representation for it. The representation should contain the workstation, the order it was assigned to and the part worked on by that order. Algorithm 1 is then initiated with $O_{query} = \{\texttt{Order1}\}$, $C_{case} = \{\texttt{Order}\}$, and $A_{case} = \{(\texttt{Order}, \texttt{Workstation}, \texttt{allocatedTo}), (\texttt{Order}, \texttt{Part}, \texttt{part})\}$. In the first iteration of the algorithm there will be a request to `Source3`, resulting in $O_{next} = \{\texttt{WorkStation1}, \texttt{Part1}\}$ and $O_{base} = \{\texttt{Order1}\}$. The next iteration will request `Source1` and `Source5`, with result $O_{next} = \emptyset$ and $O_{base} = \{\texttt{Order1}, \texttt{WorkStation1}, \texttt{Part1}\}$. After this iteration the algorithm terminates.

4 Conceptual Evaluation

In this section we present a conceptual evaluation of the framework and algorithm defined in the previous section. For the evaluation of the performance in terms of time complexity, we compare the proposed method to a naïve approach. The naïve approach assumes that there is no index and thus all sources have to be checked. The other extreme is to duplicate all data to the case base independent of the case model and the objects of interest for case representation (O_{query}). In this work, we assume that duplicating all data is not feasible and it is required to construct the case base on-demand. Therefore, we will focus on the computational costs of case building in terms of the number of requests to the distributed sources. In the evaluation the following dimensions are considered:

- Total number of objects ($n = |O|$);
- Fraction of objects of interest for the case base ($p = \frac{|O_{base}|}{|O|}$);
- Number of sources ($s = |S|$);
- Degree, which we define as the average fraction of sources an element in the index has pointers to ($d = \sum_{o \in O} \left(\frac{|\{S_k | S_k \in S \wedge ((o,m,l) \in S_k \vee (m,o,l) \in S_k)\}|}{|S|} \right) / |O|$);
- Number of classes in the model ($|C|$).

Note that in practice there can be many other factors that impact the actual time it takes to construct the case base, for example the network bandwidth. However, we assume that in general those costs grow proportionally to the number of requests, and therefore a lower number of requests means lower costs and better performance.

Given the dimensions above, we will evaluate the costs, expressed in the number of requests to sources (R), for three different protocols:

- naïve: no pointers, check all sources for a given object;
- object-level index: pointers for every object to sources for that object;
- class-level index: pointers for every class to sources for objects from that class.

Note that for the conceptual evaluation we assume that there is no overlap between case representations ($\bigcap_{o \in O_{query}} c_o = \emptyset$). However, in the experimental evaluation in Section 5 we will show that the amount of overlap between case representations will have an impact on the number of requests.

First, let us consider the naïve approach. Using this approach there has to be a request to every source (s) for every object that has to end up in the case base (np), such that:

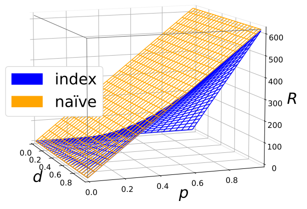
$$R_{naïve} = nps. \tag{1}$$

Alternatively, if we use the proposed framework and Algorithm 1, only sd sources have to be requested. Thus,

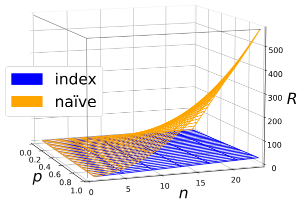
$$R_{index} = npsd. \tag{2}$$

Note that if every object or class is present in every source, $d = 1$ and $R_{index} = R_{naïve}$.

In the evaluation we consider a highly distributed environment, where data about each object is stored in a separate source, such that $s = n$. For this scenario we will investigate three relations. The first one is shown in Figure 3a where the number of sources and objects is fixed, such that if p increases, the difference in costs between the naïve ($R_{naïve}$) and index-based approach (R_{index}) will decrease linearly. Secondly, we can see in Figure 3a that the difference in costs is higher, when the data are more segregated between sources, thus if d increases the difference in costs between the two approaches decreases. Finally, when all objects can be reached using only forward trace, then $d = 1/s$ and the difference in costs between the two approaches will decrease polynomial in n (for a fixed p). This is shown in Figure 3b.



(a) $s = n = 25$



(b) $s = n$ and $d = 1/s$

Fig. 3. Costs of building a case base in terms of the number of requests to sources (R).

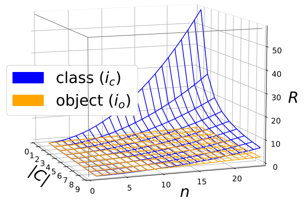


Fig. 4. Costs of building a case base in terms of the number of requests to sources (R), comparing class- and object-level index ($s = n$ and $p = 0.1$).

Next, we would like to quantify the difference between using the object- and class-level index. If we consider only forward trace, the average degree for the object-level index is $d_o = \frac{1}{s} = \frac{1}{n}$. Furthermore, if we assume that every object belongs to one class, then this degree for class-level index is $d_c = \frac{n/|C|}{n} = \frac{1}{|C|}$. This results in the following expressions for the costs using i_c and i_o, respectively: $R_{i_c} = n^2 p \frac{1}{|C|} = \frac{n^2 p}{|C|}$ and $R_{i_o} = n^2 p \frac{1}{n} = np$. From those expressions, we can derive that the difference between the costs using the class-level index (R_{i_c}) and the object-level index (R_{i_o}) will increase if the number of classes ($|C|$) decreases. More specifically, it is beneficial to use the object-level index, unless $n < |C|$. This is also visualized in Figure 4.

5 Experimental Evaluation

In this section our implementation of the framework, the experimental setup, and results are presented. The experiments are conducted using our implementation of the framework that adopts the Industry 4.0 (semantic) AAS [3,13] and Semantic Web [7] technologies as foundations. Therefore, also the data set is inspired by a Industry 4.0 modular production use case. The first goal of the experimental evaluation is to validate the implementation against the conceptual results and vice-versa. Subsequently, we will look at different scenarios with overlapping case representations.

5.1 Setup

Implementation of the Framework In terms of the levels used in Figure 1, 'Instance (data sources)' are implemented as AAS server(s), so the data are exposed according to the AAS model enriched with semantic identifiers accessible on a server[4]. The 'Model' is defined in SHACL [17], which is a more recent addition to the Semantic Web stack that it is used more regularly to define models and constraints in the manufacturing domain [15]. For the implementation of the algorithm we use an agent-based system implemented in Python, where every agent represents a specific source and maintains the 'Index' for that source. For the experiments only the forward trace pointers are included. The index is generated based on AAS metadata, consisting of the path where the data can be retrieved and the semantic identifier, which corresponds to a relation label (L) in the case model [24]. The index is updated based on events published by the AASs.

The agent-based approach makes the system modular and scalable, as data from different sources can be collected in parallel. Next to the AAS server and the agent-based system that implements the algorithm, there is a graph database (RDF [22] store) where all data collected by the agents is stored, and as such forms a case base. Using a graph structured case representation fits naturally with the structure of the AAS. However, for the implementation we made some assumptions to limit the scope and complexity of the data integration. First, we define the data set and sources, such that there is a straightforward mapping from the data elements in the AASs to unique identifiers. Next to that, we only consider 'simple' data types that can be one on one mapped to RDF literals.

[4] https://wiki.eclipse.org/BaSyx_/_Documentation_/_Components_/_AAS_Server

Hypotheses The first goal is to validate the implementation against the conceptual results from Section 4 and vice-versa. The validation should ensure that we did not miss an important dimension or interaction in the conceptual evaluation, which might occur in a practical setting. For this purpose we defined four hypotheses that are formulated based on the conceptual evaluation. The first three compare the costs for using no index (naïve approach) and the object-level index, while the last hypothesis is designed to show the difference between using the class- and object-level index.

H1 If p increases, the difference between $R_{naïve}$ and R_{index} will decrease.
H2 If d increases, the difference between $R_{naïve}$ and R_{index} will decrease.
H3 If $s = n$, the difference between $R_{naïve}$ and R_{index} will increase polynomial in n.
H4 If $|C|$ decreases, while n stays constant and there are multiple sources per class, the difference between R_{i_c} and R_{i_o} will increase.

For the conceptual evaluation, we assumed that there is no overlap between case representations. In the last experiments we will relax this assumption and show the impact of having overlapping case representations ($\bigcap_{o \in O_{query}} c_o \neq \emptyset$) when using the object-level index. More specifically, we look into how the system behaves when there is overlap between the case representations for increasing number of objects to build a case representation for. It is expected that more overlap between case representations will reduce the number of requests. For example, when multiple orders in the case base are related to the same workstation, then data about that workstation will only have to be collected once. To test this hypothesis the average overlap between representations can be altered, by varying the average number of relations per object in the data set ($\frac{|E|}{|O|}$). In general, we can state that if the average number of relations per object is higher, the probability that this object occurs in two distinct case representations is also higher, as it is related to more other objects. For example, if a workstation is related to all orders, it will occur in the case representation of each order and the number of relations of that workstation is higher compared to when it is related to only one order. The expected behaviour is captured in the last two hypotheses:

H5 If there is overlap between case representations, R will increase sub-linear in $|O_{query}|$.
H6 If $\frac{|E|}{|O|}$ is higher, the reduction in R due to overlapping case representations will be lower.

Data Set For the evaluation of our methodology we use a simulated data set describing a set of manufacturing assets, similar to the example in Figure 2. The data set is based on an actual use case using an Industry 4.0 modular production environment. In such an environment the objects are treated as independent entities and have their own AAS. The complete data set consists of 272 `Orders`, 14 `Workstations`, 17 `Parts`, and 20 `AGVs`. Every order is related to one workstation with relation `allocatedTo` and one part with relation `part`. Furthermore, every AGV is related to one or multiple parts (`transported`). All objects have their own AAS, with multiple elements representing the relations to other objects. The case model consists of all classes and relations, except `AGV` and `transported`. The objects of interest for case representation are from the class `Order`.

For testing the different hypotheses we use different subsets of the data set described above:

1. For the first hypothesis only orders are included (excluding the relations to other objects), we can then vary the fraction selected for the case base (p), by increasing the number of orders to build a case representation for.
2. For the second hypothesis, orders and their relation to workstations and parts are stored in different elements of the AAS. The fraction of sources each object is connected to (d) can then be altered by varying the connected AAS elements.
3. For the third hypothesis again only orders are included, but now the number of sources is altered by varying the number of orders that is included.
4. For the fourth hypothesis, different subsets of the data set are used to vary the number of classes ($|C|$). The number of objects is constant, but they are split over different classes.

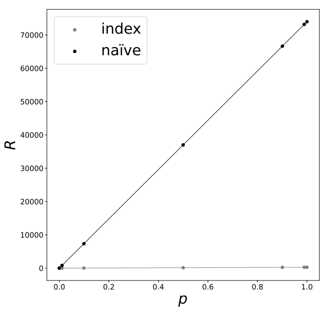
Hypotheses 5 and 6 (overlapping case representations) require a slightly different setup. For testing those hypotheses we generated distinct data sets, while utilizing the same underlying data model. In each data set a different number of workstations and parts is included, but again each order is related to one workstation and one part. This means that the number of relations remains constant, but the average number of relations per object differs. An overview of the four generated data sets can be found in Table 2.

Table 2. Description of the generated data sets for testing Hypotheses 5 and 6.

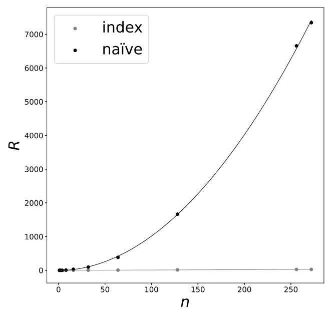
Scenario	# Orders	# Workstations	# Parts	Average # relations ($\frac{|E|}{|O|}$)
1	272	272	272	1.19
2	272	94	10	1.85
3	272	92	10	2.53
4	272	1	1	3.97

5.2 Results

Hypothesis 1 The results of the experiments for Hypothesis 1 can be found in Figure 5a. In accordance with the conceptual results, the number of requests, R, grows linearly in p for both the index-based and the naïve approach. However, there is a difference in the slope, which is equal to nds ($= 272 \times 1/272 \times 272 = 272$) for the index-based approach, compared to ns ($= 272 \times 272 = 73984$) for the naïve approach.



(a) Number of requests for different fraction of objects for case base.



(b) Number of requests for different number of objects in the sources.

Fig. 5. Results of experiments testing hypothesis 1 and 3, including regression lines.

Hypothesis 2 Based on the results in Table 3 we can accept Hypothesis 2: the number of requests for the naïve approach is constant if only d is varied, while the requests increase linear in d when the index-based approach is used. Therefore, the benefit of using the index will decrease if objects or classes are present in an increasing fraction of all sources.

Table 3. Results for testing Hypothesis 2.

d	0.0037	0.0074	0.011	0.015
$R_{naïve}$	29376	29376	29376	29376
R_{index}	27	54	81	108

Hypothesis 3 We can accept this hypothesis based on the results in Figure 5b. The number of requests is linear in n for the index-based approach, but (second order) polynomial for the naïve approach, which corresponds to the conceptual results in Section 4 in a scenario with $s = n$.

Hypothesis 4 As can be seen in Figure 6, the number of requests when using the object-level index is independent of the number of classes the objects are distributed over, while the number of requests when using the class-level index decreases when more classes are considered. These results correspond with the findings in Section 4, and therefore we can also accept Hypothesis 4.

Hypotheses 5 and 6 The results are summarized in Figure 7. In Scenario 1, where there is no overlap in case representation, similar behaviour is observed as

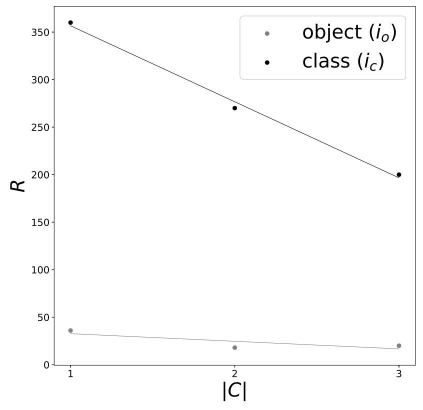


Fig. 6. Results of experiments testing hypothesis 4, including regression lines.

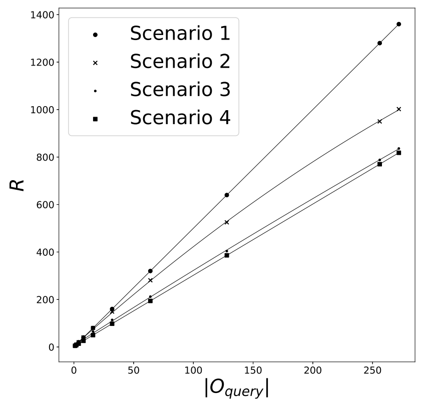


Fig. 7. Results for the scenarios in Table 2 (including regression lines).

in Figure 5a). Similarly, in Scenario 4 the number of requests increases linearly for increasing number of cases, this can be explained by the fact that the same workstation and part occur in all case representations, so the workstation and part will be collected once the first case representation is built. The sub-linear behaviour is especially clear for Scenario 2 with overlap between every two to four case representations. In summary, the effect of overlapping case representations increases with increasing number of cases and increasing average number of relations per object.

6 Discussion

We found that the experimental results follow the conceptual results for the computational complexity, which means the relatively straightforward expressions can be used to extrapolate theoretical complexity from smaller instances of the problem. The extrapolation can be used to evaluate bigger instances of the problem without the effort to develop complex simulations. The worst-case performance was observed for a highly distributed environment where every object has its own data source. In this scenario the performance for the naïve approach behaves polynomial versus linear if the proposed index-based approach is used. Polynomial behaviour of order two is not by definition unacceptable, but on industry scale the performance will rapidly decrease if every request takes a couple of milliseconds. Note that in the evaluations of the framework, the time and effort to build and maintain the index itself are not considered. Both can be achieved in different ways, but in general push-based (or event-driven) mechanisms are especially efficient for such tasks [29], and we argue that those costs are negligible in comparison to the cost savings that we found in our evaluation.

Although in this work the focus is on manufacturing environments, the framework can also be applied in other domains with distributed data sources that can be integrated using some (semantic) model. One example is the Web, that is distributed by nature. Furthermore, many web pages embed semantic (meta)data [14], which can be useful to include in a case representation that can be built using the proposed framework.

For the implementation some assumptions were made to make the data integration less complex, but in practice there might be heterogeneity in data sources and types. The advantage with the proposed framework and implementation is that most of the data integration can be done client-side, which makes the CBR system less dependent on the source systems. In comparison to federated query engines, our framework and implementation does not support executing more complex queries directly on the data sources. Instead, this could be an additional post-processing step, for example by executing a SPARQL query on the constructed RDF case base.

7 Conclusion and Future Work

In this work we proposed a framework that enables efficient construction of a case base from distributed sources. The main contributions are the model-based index and the algorithm to construct the case base. We demonstrated both conceptually and experimentally that the benefit of using the proposed index-based approach is most significant for highly distributed environments, where the complexity is reduced from polynomial to linear. Distributed environments are seen in multi-national (manufacturing) companies and supply chains that deal with big data volumes distributed over different locations and systems. Therefore, we showed the benefits of applying the framework to a manufacturing use case.

More research is required on integration of the framework in a complete CBR system, especially the retrieval step. There are multiple approaches to this integration, one option is to keep the representations of the cases in a central case base once they are built, then only the data for a new case presented to the system have to be collected from the distributed sources. An alternative is to only store the main case object identifiers centrally and construct the complete case base on demand, by collecting data describing the cases from the distributed sources. Also a combination of both is possible.

The framework presented in this work only supports retrieving data from the distributed sources, in contrast to distributed CBR frameworks which mainly focus on case retrieval and case base maintenance. An interesting direction to explore is how the framework proposed in this work can be combined with for example the SEASALT framework [1]. Another opportunity is to integrate the similarity computation in the framework. The algorithm can be adapted to stop retrieving data when it concludes that certain cases will not be among the most similar cases.

Acknowledgements This project is supported by the European Union‘s Horizon 2020 research and innovation programme under grant agreement No. 957204, the project MAS4AI (Multi-Agent Systems for Pervasive Artificial Intelligence for assisting Humans in Modular Production). In special we would like to thank the project partners for providing insights in their use cases and the reviewers for providing valuable comments and suggestions.

References

1. Bach, K.: Knowledge engineering for distributed case-based reasoning systems. Synergies Between Knowledge Engineering and Software Engineering **626**, 129–147 (2018). https://doi.org/10.1007/978-3-319-64161-4{_}7
2. Bach, K., Reichle, M., Althoff, K.D.: A Domain Independent System Architecture for Sharing Experience . In: LWA. pp. 296–303. Halle (9 2007)
3. Bader, S.R., Maleshkova, M.: The semantic asset administration shell. In: International Conference on Semantic Systems. pp. 159–174. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33220-4{_}12
4. Bergmann, R., Althoff, K., Breen, S., Göker, M., Manago, M.: Developing industrial case-based reasoning applications: The INRECA methodology. Springer Science & Business Media, Berlin (2003)
5. Bergmann, R.: Experience Management, Lecture Notes in Computer Science, vol. 2432. Springer Berlin Heidelberg, Berlin, Heidelberg (2002). https://doi.org/10.1007/3-540-45759-3
6. Bergmann, R., Kolodner, J., Plaza, E.: Representation in case-based reasoning. The Knowledge Engineering Review **20**(3), 209–213 (2005). https://doi.org/10.1017/S0269888906000555
7. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American **284**(9), 34–43 (2001)
8. Camarillo, A., Ríos, J., Althoff, K.D.: Knowledge-based multi-agent system for manufacturing problem solving process in production plants. Journal of Manufacturing Systems **47**, 115–127 (2018). https://doi.org/10.1016/j.jmsy.2018.04.002
9. Charalambidis, A., Troumpoukis, A., Konstantopoulos, S.: SemaGrow: Optimizing Federated SPARQL queries. In: Proceedings of the 11th International Conference on Semantic Systems. pp. 121–128. ACM, New York, NY, USA (2015). https://doi.org/10.1145/2814864
10. Charpenay, V.: Semantics for the Web of Things, Modeling the Physical World as a Collection of Things and Reasoning with their Descriptions. Ph.D. thesis, Universität Passau (2019)
11. Goel, A.K., Diaz-Agudo, B.: What's Hot in Case-Based Reasoning. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence. pp. 5067–5069 (2017). https://doi.org/10.1609/aaai.v31i1.10643
12. Görlitz, O., Staab, S.: SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In: Proceedings of the Second International Workshop on Consuming Linked Data (2011)
13. Grangel-González, I., Halilaj, L., Auer, S., Lohmann, S., Lange, C., Collarana, D.: An RDF-based approach for implementing industry 4.0 components with Administration Shells. In: 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA). pp. 1–8. IEEE (2016). https://doi.org/10.1109/ETFA.2016.7733503

14. Guha, R.V., Brickley, D., Macbeth, S.: Schemaorg: Evolution of structured data on the web. Communications of the ACM **59**(2), 44–51 (2 2016). https://doi.org/10.1145/2844544
15. Hooshmand, Y., Resch, J., Wischnewski, P., Patil, P.: From a Monolithic PLM Landscape to a Federated Domain and Data Mesh. Proceedings of the Design Society **2**, 713–722 (5 2022). https://doi.org/10.1017/PDS.2022.73
16. Jaiswal, A., Yigzaw, K.Y., Ozturk, P.: F-CBR: An Architecture for Federated Case-Based Reasoning. IEEE Access **10**, 75458–75471 (2022). https://doi.org/10.1109/ACCESS.2022.3188808
17. Knublauch, H., Kontokostas, D.: Shapes Constraint Language (SHACL) (2017), https://www.w3.org/TR/2017/REC-shacl-20170720/
18. Nkisi-Orji, I., Wiratunga, N., Palihawadana, C., Recio-García, J.A., Corsar, D.: Clood CBR: Towards Microservices Oriented Case-Based Reasoning. In: ICCBR 2020: Case-Based Reasoning Research and Development. vol. 12311 LNAI, pp. 129–143. Springer Science and Business Media Deutschland GmbH (2020). https://doi.org/10.1007/978-3-030-58342-2{_}9/FIGURES/6
19. Pla, A., López, B., Gay, P., Pous, C.: eXiT*CBR.v2: Distributed case-based reasoning tool for medical prognosis. Decision Support Systems **54**(3), 1499–1510 (2 2013). https://doi.org/10.1016/J.DSS.2012.12.033
20. Plattform Industrie 4.0: Plattform Industrie 4.0 - Asset Administration Shell - Reading Guide (2 2022), https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/AAS-ReadingGuide202201.html
21. Plaza, E., McGinty, L.: Distributed case-based reasoning. The Knowledge Engineering Review **20**, 261–265 (2006). https://doi.org/10.1017/S0269888906000683
22. RDF Working Group: Resource Description Framework (RDF) (2014), https://www.w3.org/2001/sw/wiki/RDF
23. Recio-García, J.A., González-Calero, P.A., Díaz-Agudo, B.: jcolibri2: A framework for building Case-based reasoning systems. Science of Computer Programming **79**, 126–145 (2014). https://doi.org/10.1016/j.scico.2012.04.002
24. Rongen, S., Nikolova, N., van der Pas, M.: Modelling with AAS and RDF in Industry 4.0. Computers in Industry **148** (6 2023). https://doi.org/10.1016/J.COMPIND.2023.103910
25. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: A federation layer for distributed query processing on linked open data. In: The Semanic Web: Research and Applications. ESWC 2011. pp. 481–486. Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21064-8{_}39/COVER
26. Taelman, R., Van Herwegen, J., Vander Sande, M., Verborgh, R.: Comunica: A modular SPARQL query engine for the web. In: The Semantic Web – ISWC 2018. vol. 17, pp. 239–255. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00668-6{_}15
27. Tran, H.M., Schönwälder, J.: DisCaRia - Distributed Case-Based Reasoning System for Fault Management. IEEE Transactions on Network and Service Management **12**(4), 540–553 (12 2015). https://doi.org/10.1109/TNSM.2015.2496224
28. Verborgh, R., Vander Sande, M., Hartig, O., Van Herwegen, J., De Vocht, L., De Meester, B., Haesendonck, G., Colpaert, P.: Triple Pattern Fragments: a Low-cost Knowledge Graph Interface for the Web. Journal of Web Semantics **37**, 184–206 (2016). https://doi.org/10.1016/j.websem.2016.03.003
29. Wingerath, W., Ritter, N., Gessert, F.: Real-Time & Stream Data Management. SpringerBriefs in Computer Science, Springer Cham (2019). https://doi.org/10.1007/978-3-030-10555-6