Appl Intell (2010) 33: 278–290
DOI 10.1007/s10489-009-0165-z

Prototype-based management of business process exception cases

Stefania Montani

Published online: 9 February 2009

Abstract Business process optimization may require to deviate from a default process model, in response to unexpected situations, thus raising exceptions. In this paper, we present a system for supporting end users in handling exceptions in business process management, which exploits the case-based reasoning (CBR) methodology. CBR offers the advantage of relying on operative knowledge, thus reducing the cost of knowledge elicitation, with respect to other methodologies.

To maintain and organize the case base, we resort to a type of generalized cases, known as *prototypes*. The use of prototypes allows us to structure the case base itself, thus speeding up retrieval, and avoiding redundancy. In our system prototypes are also intended as a means to help process engineers in defining revised versions of the process schema, in response to frequent exceptions.

The system is currently in use at one of the largest logistics centres in Italy.

Keywords Business process management · Case-based reasoning · Prototypes · Case-based maintenance · Exception handling support

1 Introduction

Business process management (BPM) is a set of (highly automated) activities aimed at defining, executing, monitoring and *optimizing* business processes, with the objective of making the business of an enterprise as effective and efficient as possible, and of increasing its economic success.

In particular, the optimization task may ask the enterprise to be able to flexibly deviate from the predefined process schema, in response to expected situations (e.g. new laws, reengineering efforts) as well as to unexpected ones (e.g. emergencies) [1]. Unexpected situations, in particular, require a prompt reaction, which operatively translates in generating and handling an exception to the business process execution. These exceptions are ad-hoc changes at the process instance level, operated by end users (in contrast to changes at the more general process schema[1] level, which can be forecasted and scheduled, and are operated by domain experts, i.e. process engineers).

Existing technology typically supports such ad-hoc changes [2], relying on different methods (e.g. rule-based and graph-based approaches, see also Sect. 4), which usually share the characteristic of being based on a strong (and time-consuming) formalization of domain knowledge. Moreover, in the existing systems, the effects of ad-hoc changes are normally kept local to the respective process instance (i.e. they do not affect other instances of the same process), and are normally not used to suggest a revised version of the underlying process schema to the process engineer.

In this paper, on the other hand, we propose to resort to case-based reasoning (CBR) [3] in order to support the management of exceptions in business process execution. CBR is a reasoning paradigm that exploits the specific knowledge of previously experienced situations, called *cases*. The use of CBR may mitigate the knowledge formalization effort, since representing a real world situation as a case is

S. Montani (✉)
Dipartimento di Informatica, Università del Piemonte Orientale, Alessandria, Italy
e-mail: stefania.montani@unipmn.it

[1] Intuitively, many instances of the same process schema may exist, e.g. the same plant maintenance procedure might have been instantiated and executed on different dates.

often straightforward: given a set of meaningful features for the domain, it can be sufficient to identify the value they assume in the situation at hand. The so-obtained set of ⟨*feature*, *value*⟩ pairs provides the problem description, which is typically coupled with information about the applied solution, thus completing the situation-action pattern adopted on that occasion. Such data encompasses an amount of domain knowledge, which can be memorized without the need of making it *explicit* in a more abstract and structured form, as it would be required by other methodologies (e.g. rule-based or model-based reasoning). CBR is particularly well suited for managing exceptional situations which can be neither foreseen nor preplanned. As a matter of fact, in the literature cases have often been resorted to in order to describe exceptions, in various domains (see e.g. [4–6]).

Within the proposed CBR framework, end users are enabled to *retrieve* past exceptions, in order to get suggestions on how to edit a process instance with ad-hoc changes. Moreover, we also propose an automated way of learning more general indications from ground exception cases through a proper *maintenance* procedure, and to store them in the form of *prototypes* [7]. Prototypes are a well-documented notion in the CBR literature, and are typically crucial for knowledge base organization, and for optimizing retrieval performances. In our tool, they are also exploited to support process engineers in (long term) revisions of process schemas, as envisioned in Fig. 1.

In summary, our contribution:

1. is based on well-established and well-documented methodological choices (as discussed in Sect. 2), which make it methodologically sound as well;
2. provides advances in BPM exception handling research, because:
   (a) it allows to mitigate knowledge formalization problems, and
   (b) at the same time, it proposes an automatic memory organization and maintenance procedure, which can make retrieval faster, and can support long-term revisions by process engineers.

As it will be discussed in Sect. 4, point 2(b) appears to be a particularly significant contribution in the existing literature panorama.

The paper is organized as follows. In Sect. 2 we introduce CBR preliminaries. In Sect. 3 we present the details of our contribution: in particular, in Sect. 3.1 we introduce case representation, while Sect. 3.2 deals with case base maintenance issues, and Sect. 3.3 describes our case retrieval facility. Our tool is in use at Interporto di Rivalta Scrivia S.p.A., one of the largest logistics centres in Italy, since May 2008. Section 3.4 describes our evaluation activity at Interporto, and provides objective measures of the application impact in this setting. In Sect. 4 we describe related work. Finally, Sect. 5 is devoted to conclusions and future work.

2 Case-based reasoning

CBR is a reasoning paradigm that exploits the knowledge collected on previously experienced situations, known as *cases*.

In the classical approach, a case consists of a *problem description* able to summarize the problem at hand, and of a *case solution*, describing the solution adopted for solving the corresponding problem; sometimes a *case outcome* may be stored as well.

The problem description can be represented as a collection of ⟨*feature*, *value*⟩ pairs, a format which was introduced in the early 90s [8], and which is still often resorted to. In this work, we basically adopt this format as well. Nevertheless, more complex representations are also possible. For instance, problems may be reported in the form of images, time series or text.

CBR can be summarized by the following four basic steps, known as the *CBR cycle*, or as the four *"res"* [3]: *retrieve* the most similar case(s) with respect to the input situation from the case repository, known as the *case base*; *reuse* them, and more precisely their solutions, to solve the new problem; *revise* the proposed new solution (if needed); *retain* the current case for future problem solving.

Actually, in many application domains it is common to find CBR tools able to extract relevant knowledge, but that leave to the user the responsibility of providing its interpretation and of formulating the final decision: reuse and revise are therefore not implemented. However, even retrieval alone may significantly support the human decision making process [9]. In the present work, we are following this policy as well.

Applying CBR can have a computational justification: as observed in the Introduction, by resorting to CBR the effort of knowledge acquisition and of knowledge representation is often mitigated, since, given a set of meaningful features for the application domain, it can be sufficient to identify the value they assume in the situation at hand to define the problem description of a new case. Cases represent an "implicit" form of knowledge, meant as an unstructured, operative knowledge type, which directly stores the problem-solution patterns that have occurred in time "as they are", without any effort in the direction of extracting more abstract information (e.g., of eliciting rules or models, which can be defined as "explicit" or "structured" knowledge) from them. Moreover, new implicit knowledge can be automatically stored in the case base during the every day working process. As the case library grows, more and more representative examples can be retrieved, and it can become easier to find a proper solution to the problem at hand by means of this paradigm.

While augmenting the case library content may be relatively easy, retaining new cases may raise some issues. Actually, the problem solving competence of the case base not

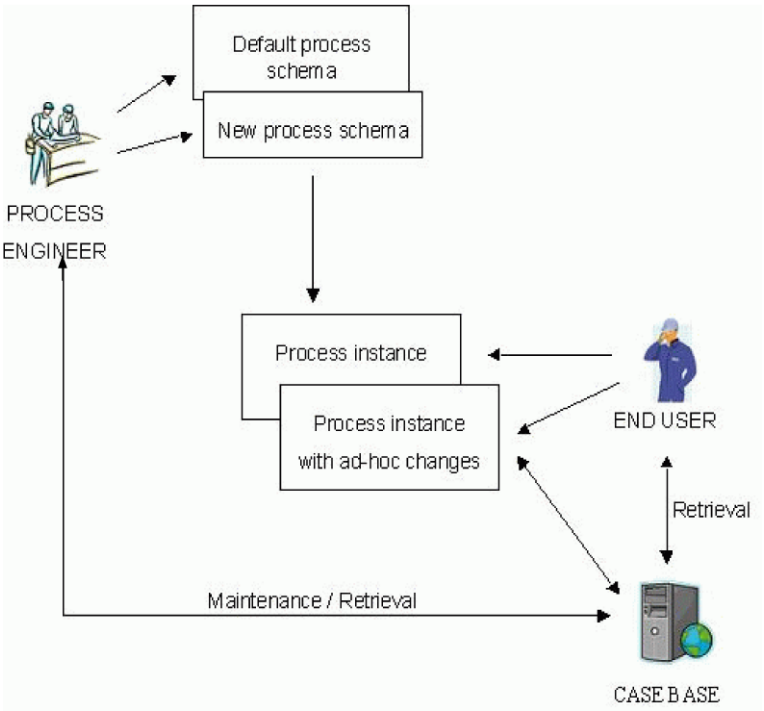

Fig. 1 CBR for exception handling in BPM. A process engineer normally issues a process schema, which is then instantiated and applied as a default procedure by end users. Due to an emergency, an end user may want to deviate from the default process schema, and generate a process instance with ad-hoc changes. Through our system, she can be supported in this activity by retrieving from the case base modifications that were applied to the default procedure in the past, motivated by similar reasons. The instance with ad-hoc changes finally edited by the end user is then saved as a new case in the case base. The case base content can be periodically analyzed, by activating the maintenance procedure we have implemented, which enables to learn more general indications (e.g. frequent changes) from the collected ground cases, and to store them in the form of prototypes. The process engineer can finally retrieve these prototypes, and be supported in issuing a new version of the process schema, which could, for instance, incorporate the most frequent changes once and for all

necessarily grows as much as its size. It has been shown (see e.g. [10, 11]) that storing too many cases may increase retrieval time unacceptably, while some cases could be deleted, since their problem-solution information is already represented in other existing ones.

These observations have led to a significant research effort in the direction of case base *maintenance*. Case base maintenance is an important process directly connected to the retain step of the CBR cycle, which may have a significant impact on the actual performance of the reasoning system. The work in [12] provides an interesting survey on the possible policies adopted in the literature to this end. Among these policies, a very promising one resorts to the definition and exploitation of *prototypes* [7]. Prototypes are a generalization from single to clustered typical cases. The main purposes of such a generalization knowledge are to:

- organize the case base;
- guide and speed-up the retrieval process;
- decrease the storage amount by erasing redundant cases.

In particular, the periodic reorganization of the case base to learn or update the prototype definitions, by taking into account the new acquired cases, automatically allows one to delete or disregard the redundant or useless ground cases. An evaluation of each newly acquired case is therefore not needed with this strategy.

In this work, we deal with case base maintenance issues resorting to prototypes.

The idea of relying on prototypes is founded over a well-established literature tradition, well examined in [13], in which the same notion is also referred to with different names. In particular, a notion somehow similar to the one of prototype was originally introduced in the theory of dynamic memory [14]; according to this theory, generalized knowledge is held by *Memory Organization Packets* (MOPs), which also organize specific experiences in cases. In this model, cases are the starting point for problem solving, while MOPs provide guidance for adaptation. In the 90s, Bergmann [15] introduced the concept of *generalized case*, intended as an entity that can be directly reused for wider ranges of problems than specific cases. A very early approach using an equivalent notion in instance–based learning research was represented by Protos [16]. The definition of prototype is also very similar to the one of *abstract case* [17], obtained by merging two or more cases with the same solution in a single entity.

The tool we have implemented is thus grounded on strong literature foundations.

Details of how it operates can be found in the next section.

3 Case-based reasoning for business process exception management

In this section, we introduce the details of our framework. First, we describe process schema primitives, and case representation ones (see Sect. 3.1). Cases are stored in the case base, within a hierarchical organization which relies on prototypes; details of case base maintenance and of memory organization are provided in Sect. 3.2. The retrieval procedure, which takes advantage of the case base hierarchical organization, is described in Sect. 3.3. Finally, our evaluation results, obtained in a real world setting, are presented in Sect. 3.4.

3.1 Process schema and case representation

BPM activities are based on a predefined process schema, consisting of the tasks to be executed, of their control flow connections, of the actors meant to perform them, and on the data which have to be provided to enable the task execution. In this section, for the sake of simplicity, we make the hypothesis that a single actor is responsible for completing all the tasks of the process schema.[2] Therefore, we need to represent: (i) tasks, (ii) control flow relations and (iii) data. As in many process modeling systems (see e.g. [18]), in order to enhance usability, we have defined a reduced set of representation primitives, enabling to describe a process schema.

According to our representation formalism, a process schema can be represented as a hierarchical graph, where nodes are the tasks to be executed, and edges are the control flow relations linking them. We can distinguish between atomic and composite tasks (plans), where atomic tasks represent simple steps in the process, and plans represent tasks which can be defined in terms of their components via the *has-part* relation. The overall process itself is a plan. Two different types of atomic tasks can be identified: (1) *actions*, i.e. tasks that describe an activity which must be executed at a given point of the process (e.g. to switch on a software device); (2) *decisions*, used to model the selection among different alternative paths (e.g. to test if a software device is responding or not; different actions will then be taken, depending on the answer).

Needed data (e.g. resources and constraints) are stored as *properties* of the actions in which they are resorted to.

Control relations, on the other hand, establish which tasks can be executed next, and in what order. In particular, the *sequence* relation explicitly establishes what is the following task to be executed; the *alternative* relation describes which alternative paths stem from a decision, and the *repetition* relation states that a task has to be repeated several times (until an exit test, modeled by means of a decision, becomes true). Join and fork constructs are modeled as well; thus, we can also represent parallel executions.

These primitives will be referred to in Sect. 3.4; in Fig. 3, in particular, boxes will be used to represent actions, diamonds to represent decisions and circles to represent joins. Additional control flow relations will be straightforwardly depicted by arrows.

A process schema or exception instance can be acquired by means of our tool's graphical interface, which also incorporates a set of logical consistency checking facilities. For instance, it automatically verifies that different alternatives only stem from decisions (and not e.g. from an activity).

The process schema, as well as its exception instances (i.e. *cases*), and prototypes built upon them, are then maintained in a relational database. Cases and prototypes share the same structure (i.e. they have the same features).

In particular, a case stores an atomic change made to a process schema at execution time, together with its problem description.

In the BPM domain, the problem description has to keep track of the *context* which motivated the exception raised by the end user. The unavailability of some needed data may justify an exception. Therefore, in our approach the problem description includes information about the presence of required technological resources, human competences, time constraints and additional (application-specific) data, all modeled as case features.

Since a user may need to adapt/change a process instance also when its applicability conditions are met (and thus when the context alone is not sufficient to clarify the reasons for raising an exception), we also add the possibility of justifying the reasons for deviation as free text. Such text can provide an insight of the user's motivations to a colleague that will retrieve that case in the future, but will not be resorted to when calculating distances in the retrieval step (see Sect. 3.3); an interpretation of textual features is left as a future work.

Another special feature we introduce is *reputation* (see also [19]). Reputation is a sort of score, set to 1 when the case is generated, and increased by 1 every time a user retrieves the case and judges it to be useful for her current problem. Reputation is decreased if the user retrieves the case, but then discards it. A high reputation is therefore an indicator of appropriateness. Users are also encouraged to decrease a case reputation if the modifications suggested in that case resulted in problems when applied in practice.

[2] A rather realistic assumption in some application domains, and in particular in the one in which we are currently working, see Sect. 3.4.

Finally, we introduce the *link* feature, to keep track of the parent relation between a prototype and the ground cases it subsumes. Details on feature setting in prototypes will be discussed in Sect. 3.2.

The atomic change to the process schema more properly represents the case solution. Different types of atomic changes may take place, namely: insertions, deletions and updates, of activities, decisions and control flow relations.

Cases thus store atomic elements of an exception. The overall set of changes applied as a single exception to a process schema on a certain date can be reconstructed by executing a query in the case base, restricted to the exception date.

Thus, an exception is not stored as a single memory item (i.e. as a single case) in the database, but it is a meta–level concept, whose ground implementation consists of a set of cases, that can be joined by means of their execution date. Since each case reports an atomic change, justified by a set of features, the set of features justifying the overall exception can be obtained as the union of the features associated to every atomic case composing it.

3.2 Case base maintenance

After a set of new exception cases are generated, we search for redundancies or partially matching features between their problem descriptions and the ones of the already stored cases, with the double aim of (1) hierarchically organizing the case base, and (2) avoiding to retain useless information. The maintenance procedure can be activated by process engineers through the tool's interface.

The generalized information that can be extracted from a set of ground cases is stored as a prototype. In particular, similarly to an abstract case [17], in our approach a prototype summarizes a set of ground cases sharing the same solution. Operatively, it has the same structure of a ground case, and its features are automatically calculated as follows. The feature values that all the ground cases share in their problem descriptions are kept unchanged in the prototype problem description. On the other hand, the features assuming different values in the ground cases take a null value in the prototype. However, if all the subsumed cases have a null value in a certain feature, except one, the prototype assumes the only non-null value in the feature at hand. The rationale behind this choice is that the prototype solution is justified by the union of the motivations (i.e. of the problem descriptions) of the subsumed cases.[3] The *reputation* feature of a prototype is initially set to the sum of the reputations of the ground cases it summarizes. The prototype also references all the ground cases it subsumes by means of the *link* feature—which allows for an easy memory navigation.

```
Let fki 1<=k<=N be the features in case ci
Let fkj 1<=k<=N be the features in case cj
Let soli, solj be the solutions of cases ci, cj
Let ri,rj be the reputations of cases ci,cj

Let fkp 1<=k<=N be the features in prototype p
Let solp be the solution of prototype p
Let rp be the reputation of prototype p
Let linkp be the link feature of prototype p


if soli==solj
  set solp=soli
  for all k
     if fki==fkj set fkp=fki
     else if fki==NULL set fkp=fkj
     else if fkj==NULL set fkp=fki
     else set fkp=NULL
  set rp=ri+rj
  set linkp={ci,cj}
```

Fig. 2 Pseudo-code of the prototype creation procedure, in the hypothesis of working with two cases, sharing the same solution

Figure 2 summarizes the procedure, in the hypothesis (without loss of generality) of working with just two cases, sharing the same solution.

Ground cases perfectly represented by a prototype (i.e. sharing all the same feature values of the prototype, or having a null value in a feature which assumes a non-null value in the prototype) can be deleted on demand,[4] thus avoiding redundancies. Actually, when retrieval efficiency is related to the case base size, keeping redundant cases only degrades performances, by increasing retrieval time [11].

In the current version of the system, due to the characteristics of the application domain (see Sect. 3.4), it was sufficient to define prototypes which only subsume ground cases. However the framework could be trivially extended to allow a multi-level hierarchy, in which more generalized prototypes reference more specific ones, progressively moving towards ground cases. Also observe that less strict policies in the prototype definition are allowed: for instance, it could be possible not to force all subsumed cases to share the same solution of the prototype, but just part of it [20]; moreover, a prototype could include a non-null feature value which is shared only by a certain percentage (lower than 100%) of the subsumed cases.

The use of prototypes, while allowing to automatically identify/delete useless cases, also imposes a hierarchical organization to the case base. Such an organization allows a quicker and more focused retrieval (see also [21]), since the identification of the most similar prototype (with respect to the input case) can be exploited to reduce the retrieval search space only to the cases subsumed by the prototype itself,

[3] Other policies are possible, and may be chosen depending on the application domain.

[4] In the application domain in which we are currently working (see Sect. 3.4) case deletion has not been required yet, due to the relatively low number of collected cases.

thus ignoring the rest of the case base (details on the retrieval procedure are provided in the following section).

Moreover, prototypes can support long-term revisions of the process schema by the process engineer. As a matter of fact, prototypes properly group sets of consistent examples, representing frequent, similar modifications to instances of the same process schema, which could justify the choice of issuing a new version of the default process schema itself, incorporating such changes once and for all. Within our tool, we allow the process engineer to retrieve a prototype, and then to progressively navigate down in its hierarchy, in order to inspect the details of the subsumed cases. This facility is meant to support her in such a schema revision activity, still leaving her the complete responsibility of the final decision.

Finally, it is worth observing that prototypes allow to extract more generalized knowledge from ground cases. However, such extraction does not require an explicit formalization of domain knowledge, and a consequent involvement of a domain expert. Actually, despite the fact that prototypes do summarize a set of (very similar) cases, and generalize them to some extent, they do not constitute highly abstracted evidence. Thus, our memory organization and maintenance strategy still keeps the knowledge elicitation advantages of the CBR methodology discussed in the Introduction.

3.3 Case retrieval

Within our framework, case retrieval is primarily conceived as a support for exception handling by end users.

When a user encounters an emergency in executing a process instance, she may ask our tool to retrieve suggestions from the case base, providing as an input the current data and resources information, which represent the context to be used for indexation, and which are interpreted as the input case features.

The most similar prototypes are then shown to the user; she can indicate a subset of them, thus restricting further retrieval just to the ground cases subsumed by the selected prototypes. She could also decide to stop the search at the prototype level, if she believes that the retrieval information is sufficient. Additional ground cases can also be retrieved if they are not indexed under any prototype in the taxonomy, but are similar to the query case.

In order to retrieve the most similar (i.e. less distant) prototypes/cases with respect to the input one, it is a common technique to provide a measure of distance in the features space. As already observed, prototypes are generalized cases, are physically stored in the same memory and share the ground cases structure. Thus, distance calculation can operate identically on prototypes and on ground cases. Generally speaking, the distance $d(c_i, c_j)$ between cases c_i and c_j can be computed as a weighted average of the normalized distances between their various features, that is:

$$d(c_i, c_j) = \frac{\sum_{f=1}^{N} w_f \cdot d(c_i(f), c_j(f))}{\sum_{f=1}^{N} w_f} \quad (1)$$

where $d(c_i(f), c_j(f))$ and w_f denote the normalized distance between feature f of cases c_i and c_j, and the weight associated with this feature, respectively. Weights can be properly set to state that some features are more "important" for retrieval relatively to the others. They have to be experimentally set and tuned, and their choice varies from domain to domain.

Various metrics can be relied upon to calculate $d(c_i(f), c_j(f))$; we are currently choosing the heterogeneous euclidean-overlap metric (HEOM) [22], a distance metric able to treat both symbolic and numeric variables, and to cope with the problem of missing data.

Prototypes and/or cases are inversely ordered by distance with respect to the query case; if two or more items have the same distance, they are further ordered by reputation.

In our approach, retrieval solutions are then shown to the user for a personal interpretation. After she has selected a prototype or case from the retrieval output list, since retrieved items represent atomic modifications within a more complex exception, she is also allowed to reconstruct the whole exception information happened on the date at hand, by properly querying the case base. We then leave to her the responsibility of the final decision.

Prototype retrieval can also support process engineers in schema redefinition, as already explained in Sect. 3.2.

3.4 Evaluation results

3.4.1 A real world application

The tool described in this paper is implemented in Java, and relies on MySQL for the case base storage, while the graphical interface has been implemented by resorting to JGraphPad software (an open-source product written in Java which can be extended to support additional features, and which can be properly interfaced with other software modules, see http://sourceforge.net/projects/jgraph/).

The current version of the tool has been made available at Interporto di Rivalta Scrivia S.p.A., Italy, since May 2008. Interporto, with its 1300000 m^2 of surface extension and with more than 500 employees, is one of the largest logistics centres in Italy. Interporto does not only take care of goods storage, but also of their preprocessing; in particular, several plants for food pre-processing are available. Engineers and employees responsible for goods and tanks movement, and for plants administration and security, follow specific process schemas for designing and executing equipment management and maintenance procedures. Our tool is

enabling them to keep track of unexpected exceptions, to deal with them, and to learn generalized knowledge which can be useful to restructure the schemas.

Our evaluation activity at Interporto has been structured as follows. First, we conducted a pilot study, in order to assess the reliability of our tool. The pilot study lasted eight weeks, and involved the adoption of the tool for supporting exception handling to the bag cutter machines maintenance process. Four identical bag cutters are available at Interporto, and cases were collected from all of them. The details and the main results of the pilot study are presented in Sect. 3.4.2.

After the pilot study results became available, we evaluated them together with the process engineer responsible for the bag cutter machines. Given the very encouraging outcome, in September 2008 we started a second, more extensive evaluation phase, which is still going on. Currently, the tool is being routinely applied to all the equipment management and maintenance procedures—and not only to the bag cutter machines, as in the pilot study. Section 3.4.3 provides an analysis of the impact of three months of usage of the tool at Interporto, and highlights advantages and improvement suggestions identified so far.

3.4.2 The pilot study

The eight-week pilot study involved the usage of the tool just on the four bag cutter machines. Each bag cutter available at Interporto automatically cuts bags (typically containing coffee and cocoa beans), empties them, and arranges the content in a tank for transportation. If the machine does not work properly, some beans may remain inside the bags, so that the bags have to be manually verified and emptied, highly increasing the time and cost of the activity. Thus, the status of the machine elements (e.g. blades, drive shaft, ball bearings) is periodically evaluated, and deteriorated pieces are properly treated or substituted.

Part of the default maintenance process schema is provided in Fig. 3, as represented by our tool's graphical interface, and is described below. Some details of the procedure that were useless for the discussion, have been removed, for the sake of legibility.

By following such part of the schema, (1) the slope of the drive shaft is first evaluated, and corrected, in case of need. Then, (2) the cutting performances are tested on a set of bags. If the result is not satisfactory, (3) the drive shaft and (4) the ball bearings are verified for possible fractures, and substituted if necessary. Finally, (5) the blades are sharpened.

In the first two weeks of our pilot study, during the everyday working activity, some exceptions were raised with respect to this procedure. In particular, in the first exception (case1), due to time constraints, step (2) was skipped. In the second exception, time constraints problems as well as the unavailability of the grindstone determined not only that step (2) was skipped, but also that the blades were not sharpened—see step (5)—but directly substituted with new ones. These two modifications were stored as two separate cases (case2 and case3), as explained in Sect. 3.1. In the third exception (case4), blade substitution instead of sharpening took place as well (step (5)), simply motivated by the fact that blades had not been substituted recently.

The tool organized these 4 cases under two prototypes: (p1) the blade substitution one; and (p2) the skipped cutting performance verification one.

Figure 4 shows the (main) features of the exception cases described above, together with the prototypes created from them.

After this hierarchy definition, end users exploited the tool for the subsequent six weeks. In particular, those who were affected by strict time constraints, could retrieve both the p1 prototype, and the p2 prototype (which code the presence of time constraints in their features). Both p1's solution (i.e. substituting the blades) and p2's one (i.e. skipping the cutting performance verification) were then suggested to them. Moreover, users were allowed to navigate the hierarchy, thus accessing the details of the ground cases subsumed by the two prototypes. By querying the case base on the basis of the date, for instance, they were also allowed to retrieve the whole exception composed by case2 and case3, thus obtaining the suggestion of applying both the solutions at the same time. All these data were provided to help users to better understand the solutions taken in the past, and to further guide them in decision making. However, they were always free to decide whether to reapply (one of) the retrieved solutions, or to edit a new one.

After six weeks of usage, 10 cases confirming the blade substitution change, motivated by various reasons, were collected and stored under prototype p1. The process engineer then evaluated this suggestion, in order to understand if it could be implemented within the default bag cutter process schema, being it a very frequently happening change. He judged the possibility of converting blade sharpening into blade substitution as a reasonable modification, and then decided to adopt it. Actually, even if this change obviously increases costs, it is also true that blades, due to the normal working activity, soon become so deteriorated that sharpening them during preplanned maintenance procedures can be not enough. Therefore, either sharpening frequency is significantly increased (which also increases costs and increases the machine unavailability time), or blades are always substituted. This second alternative, suggested by the tool, was preferred. The modified bag cutter maintenance procedure has been adopted in the second phase of our evaluation activity, described in the next section.

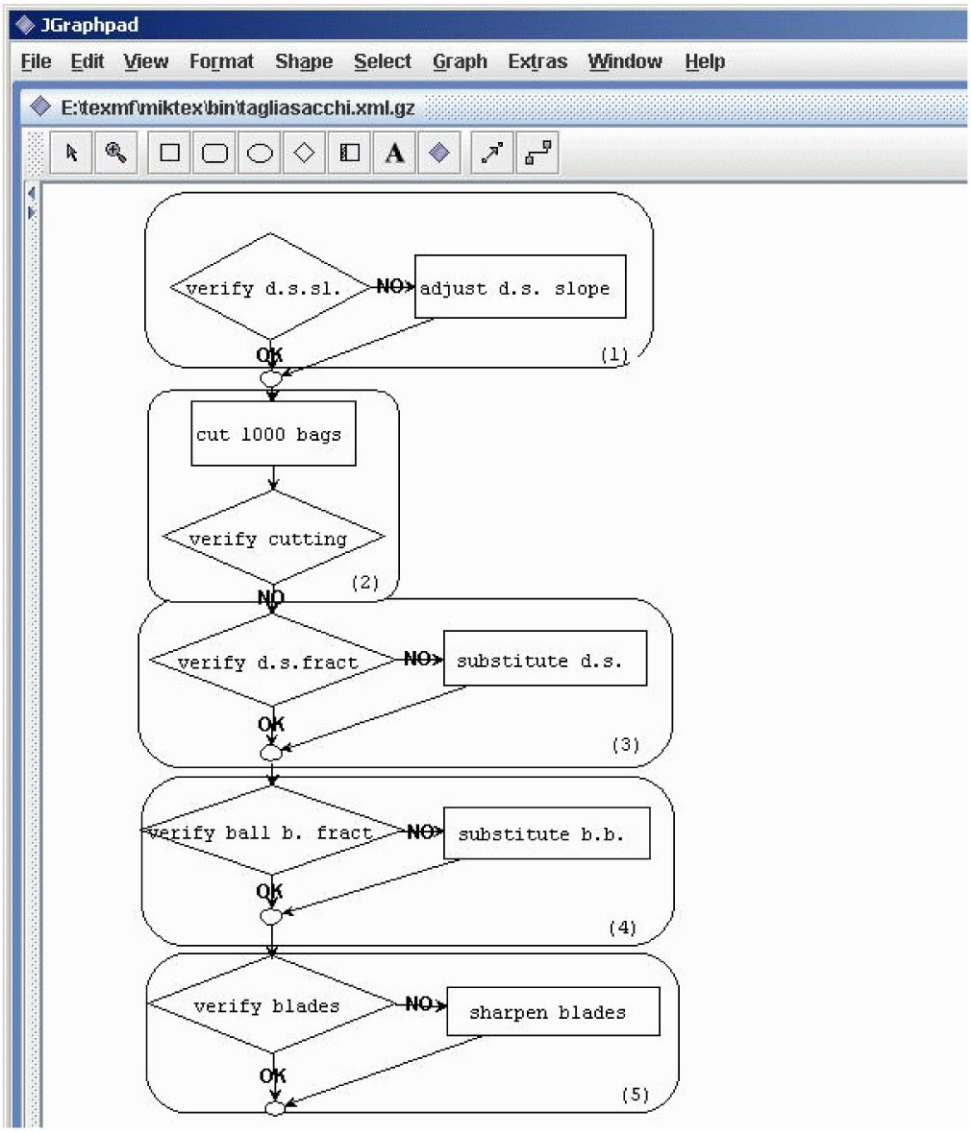

Fig. 3 Part of the bag cutter maintenance procedure, acquired by means of our tool

3.4.3 Routine adoption of the tool at Interporto: measurable outcomes

Our tool is being routinely used at Interporto since September 2008. In this section, we report on the impact of the first three months of usage, and discuss the improvement suggestions that emerged so far.

Several equipment and people were (and still are) involved in this extensive evaluation phase. In particular, the tool is being applied to process schemas concerning the management/maintenance of:

1. 150 carts for goods movement within the warehouse;
2. 6 carts for tanks movement;
3. 4 bag cutter machines (as in the pilot study);
4. 1 cocoa butter melting machine, which melts cocoa butter, filters such pre–processed food, and loads it into a tank;
5. 1 pallet assembler machine, which properly organizes food bags in a movable platform;
6. 3 identical freezing plants, globally serving 100000 m^3 of refrigerated warehouses.

Three process engineers and eight end users are exploiting the tool, and providing their feedback.

From September to November 2008, we were able to collect the following cases:

1. 2 cases related to the goods movement carts maintenance process schema;
2. 0 cases related to the tanks movement carts maintenance process schema;
3. 7 cases related to the new version of the bag cutter machines maintenance process schema (see Sect. 3.4.2). 5 of them were further indexed under 2 prototypes;

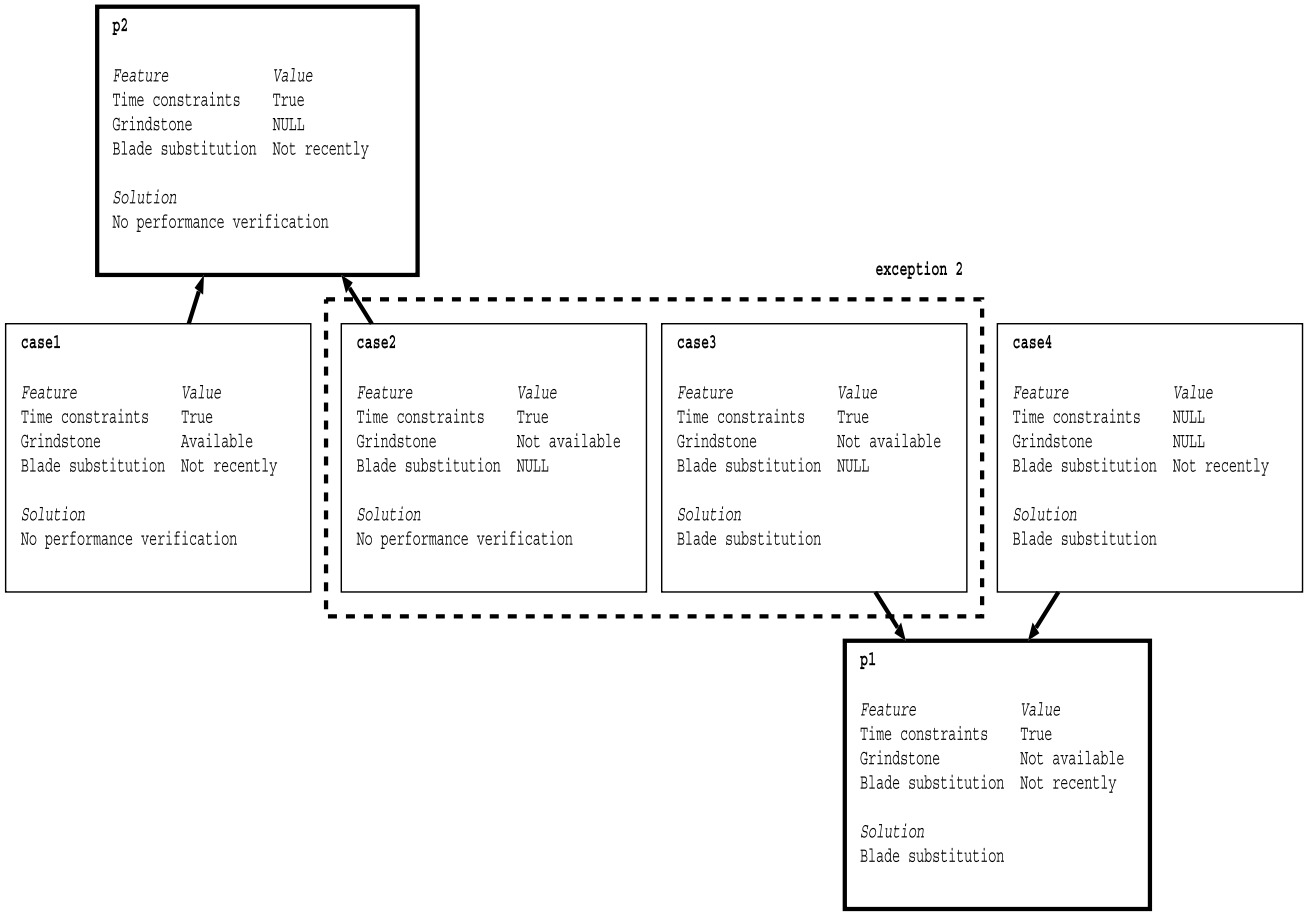

Fig. 4 A snapshot of the case base in the pilot study

4. 2 cases related to the cocoa butter melting machine maintenance process schema;
5. 13 cases related to the pallet assembler machine maintenance process schema. 3 of them were further indexed under 1 prototype;
6. 25 cases related to the freezing plants management process schema. The first 14 were collected during the first six weeks of usage, and were all indexed under 3 prototypes. Eventually they led to a process schema redefinition. The other 11 were collected during the following six weeks, and were related to the new version of the process schema. 8 of them were further indexed under 3 prototypes.

The very low number of exceptions raised for devices 1, 2 and 4 in the list above are justified by two reasons: the maintenance procedures are very simple, and the devices rely on a very well established technology and design. Devices 1, 2 and 4, therefore, have not been considered for measuring the impact of our tool on Interporto's outcomes.

On the other hand, devices 3, 5, and 6 raised a relatively high number of exception. This fact was foreseen by process engineers. Actually, machines 3 and 5 are not commercial devices (they have been partly designed by Interporto engineers themselves), and are still in their testing phase; therefore frequent exceptional needs can emerge during their maintenance. Also observe that device 5 is composed by a very large set of simpler modules. The collected cases often involved different modules, so that it was not possible to organize them under the same prototype. This also explains why no new maintenance process schema was issued by the process engineers for device 5 in these three months.

On the other hand, the freezing plants (device 6) management procedure is complex per se, and (similar) exceptions are frequent: as a matter of fact, environmental conditions, such as humidity and temperature, can strongly influence the freezers performances, and have a significant effect on electrical and water consumptions, motivating tuning activities and changes. Additionally, the I/O food flow in the refrigerated warehouses, and the initial food temperature and humidity, have to be considered as well.

Thus, devices 3, 5 and 6 were very useful to measure the impact of the *retrieval facility* of our tool, and device 6 was also useful to evaluate the impact of the *case base maintenance and schema revision suggestion facility* (while en-

gineers issued a revised process schema for device 3 only during the pilot study). Details are provided below.

Impact of the retrieval facility The main advantage of the adoption of the tool, focusing on the retrieval facility, was measured in terms of **time**: since, when raising an exception, end users were allowed to retrieve similar cases occurred in the past, they were able to manage the current problem more quickly, with respect to what happened before the tool was made available.

In particular, in 85% of the situations, end users could rely on (one of) the most similar retrieved cases or prototypes, in order to manage the emergency at hand, since (its) their motivations were indeed very close to the input case ones.

Thanks to end users' time savings, Interporto obtained several objectively measurable advantages:

1. machine unavailability time was reduced of about 20%, as an average, for devices 3 and 5. The freezing plants management procedure completion time was reduced of about 10% in the first six weeks, and of a further 20% after the new version of the process schema was issued;
2. 30% less overtime work had to be paid in the end users' wages;
3. end users could employ their time to properly complete other, less urgent procedures, which used to be delayed of several days before the tool was adopted;
4. as a consequence of items (1) and (3) above, with all devices working more properly thanks to the quicker and more effective maintenance procedures, Interporto was able to deliver its goods always on time during the evaluation period.

Outcomes (3) and (4) above were judged to be particularly relevant by Interporto managers, since they allowed them to be more compliant with ISO 9004, a guideline developed by the International Organization for Standardization (ISO), meant to improve business organizations performances—see http://www.iso.org/iso/home.htm.

However, in 15% of the situations, the tool was unable to retrieve very helpful suggestions. This could be partly due to the relatively low number of cases stored in the case base—a problem of knowledge competence that, very probably, will be automatically overcome in the future, as the tool adoption goes on. On the other hand, the issue may be also partly due to a low capability of the case features to capture the reasons why a specific change was made. Textual comments may be helpful to clarify the motivations of such "ambiguous" cases: in the future we will thus work on an automatic interpretation of such information, in order to improve the tool reliability.

Finally, end users were interviewed, in order to assess the tool usability (consider that they are not computer scientists). Globally, the tool interface was judged as quite easy to use, and user friendly. However, all the eight end users needed a training phase before using the system, and two of them, in several occasions, asked a computer technician to help them. The further simplification of the tool graphical interface will be another objective of our future work, and we will make a new release of the tool available as soon as these changes are completed.

Impact of the case base maintenance and schema revision suggestion facility During the overall evaluation, Interporto engineers issued just two new versions of the existing process schemas: one for the bag cutter machines (in the pilot study), and one for the freezing plants (in the second phase of the evaluation procedure). As already observed, the freezing plants management process is particularly complex, since it often has to be tuned/changed depending on weather or food conditions: this motivates the high number of (similar) exceptions happened during the three months of tool usage. On the other hand, no enough cases and prototypes were collected for the other equipments, as an average. A relatively high number of exceptions took place only for the pallet assembler machine, but, as explained before, it was not possible to identify many common suggestions in them.

The changes to the bag cutter machines process schema have been already discussed. In the following, we will thus draw some conclusions about the impact of the tool adoption, focusing on the case base maintenance and schema revision suggestion facility, by considering the freezing plants management procedure.

From the beginning of September to mid October 2008, 14 cases were collected, reporting modifications to such procedure. The tool was able to identify common changes in them, thus indexing all of them under 3 prototypes. Guided by memory navigation, the responsible engineer was helped in defining a new version of the management process schema, in which these very frequent modifications could be permanently addressed. According to the engineer's comments, the need for these modifications was not trivially evident. Moreover, with the help of the tool, he could immediately issue a new schema incorporating all of the three changes, instead of reaching the same result by issuing several versions in sequence, each one addressing only one change at a time, as it often happened in the past.

The main advantage was thus measurable in terms of **time**, but also of **quality**: the engineer was helped to quickly issue a better version of the process, which has been made available since October 2008, and which allowed Interporto to:

- reduce energy consumption of about 8% and water consumption of about 15%.

Despite the fact that the procedure is probably still ameliorable (further exceptions have been raised from mid October on), such savings help in keeping under control the impact that Interporto's activities have on the environment. This result is considered as very valuable by Interporto managers, since they aim at certifying the business organization according to the ISO 14001 environmental quality standard (see http://www.iso.org/iso/home.htm). Such a certification is not compulsory in Italy, but testifies the will to make the activities of an organization more and more sustainable.

As a final consideration, regarding the graphical interface, its usage did not represent a major issue for engineers, since they all have a strong computer science background.

We are aware that a deeper analysis of the impact of the tool adoption for long-term process schema revision requires more time and more examples; the collection and the interpretation of additional data about schema revision suggestions will thus be the main objective of our evaluation work from now on.

4 Related work

A wide literature exists about dynamic changes in BPM, as regards both modifications at the process schema level, and at the process instance level. An *adaptive workflow* approach is typically envisioned when supporting process schema changes (see e.g. [18, 23, 24]). The survey in [2] provides a comparisons of a set of works in this area, along the lines of several correctness criteria. Besides the specific differences, the survey reveals a trade–off between the complexity of the used representation model, and the flexibility of the system during runtime. Such observation supports our choice of defining a rather limited (though sufficiently expressive) set of representation primitives, in order to handle process changes in an easier way.

However, note that our work is only loosely related to these ones, since we primarily aim at supporting exceptions, i.e. changes at the process instance level. It is true that we also provide a form of support to process engineers in schema revision, when it is triggered by frequent exceptions; however, schema revision is not automated. Moreover, we do not deal with the problem of migration, which is one of the main concerns of the works described in [2]. Migration means ensuring that, after a process schema change has been operated, instances that have not progressed too far will be executed according to new schema, while instances whose state is not compliant with the new schema will be executed according to the old one. Migration was not an issue in our application domain; however, we will possibly consider this problem as a future research direction.

As regards the works in the area of handling changes at the process instance level, which are more closely related to ours, they can be subdivided into three main categories [2]:

- rule-based approaches (see e.g. [18]): they rely on the so called ECA (Event/Condition/Action) rules [25] to automatically detect logical failures, and to determine the needed process changes. ECA rules specify adaptation at an abstract and general level, independently of any concrete execution, and are fired at execution time;
- goal-based approaches (see e.g. [26]): they formalize process goals, and then apply planning techniques to automatically "repair" process instances when the goals are not met (it is worth noting that current planning methods are unable to treat some complex situations);
- process-driven approaches (see e.g. [1]): they try to restrict possible variants to the process in advance, by using e.g. graph grammars and graph reduction rules.

A common feature of these approaches is the need for a typically hard and time–consuming knowledge acquisition and formalization activity, which involves the cooperation of a domain expert and of a knowledge engineer, in order to define the rule base or the required process model and goals. This task might be extremely difficult in practice, especially in those applications in which a strong domain theory does not exist, or knowledge is rapidly changing, or the expert is not often available. Moreover, applying the formalized knowledge in concrete cases during runtime might raise some issues, due to unexpected peculiarities of the situation at hand, which cannot be completely captured by a domain model or rule base. Finally, some kinds of changes simply cannot be preplanned at all: most of the existing systems just deal with these situations by allowing an interaction with the end user through the system (graphical) interface, without any kind of reasoning support.

As observed in the Introduction, CBR seems to be a very well suited reasoning methodology for supporting also totally unexpected changes at the process instance level, since it strongly relies on operative and unformalized knowledge.

Actually, the CBR methodology has already been exploited in the BPM domain. The system in [27], for instance, uses generalized workflow templates (i.e. a kind of prototypes) as well as concrete cases of previously defined workflows, in order to help the user in authoring her model. However, this contribution deals with process schema modeling from scratch, and not with process instance changing.

The first proposal towards the use of CBR in BPM specifically for exception handling is represented by the work by Luo [28], which adopts a rule–based system for managing dynamic changes in business processes, but couples it with a case–based retrieval facility, able to support the end user in handling unforeseen situations. With respect to our contribution, here the use of CBR is much simpler and more limited, basically because CBR is not the main reasoning methodology in Luo's work.

CBR for handling exceptions in BPM is more extensively resorted to by Weber [19]. Weber captures exceptions by

means of a conversational CBR approach, in which features elicitation partially depends on the interaction between the user and the system, and many features are in the textual form. On the other hand, we have been able to define the case structure in advance, thus relying on a more classical CBR approach. Weber's system periodically evaluates the case base content, in order to identify dependencies among cases: for instance, it is able to reveal if, when a certain change c1 is applied, a second change c2 is always applied as well. Such a facility can be seen as a means to support the process engineer for an informed revision of the process schema, after several exceptions have been collected. We have not treated this aspect explicitly; however, since we store atomic changes as cases (as Weber does), and we allow the user to reconstruct the whole set of changes that took place within a single exception date, we could easily highlight the same kind of relations; the implementation of this feature is foreseen as a future work.

Minor [29] has developed another pure CBR system for agile workflow support, which enables process revision. In the system, cases represent a process revision, as a pair of two workflows: the original one, and the modified one. These examples of alterations can be retrieved to support new changes, in similar situations. Very interestingly, Minor distinguishes between changes in tasks and changes in the activity flow, and defines two different metrics for dealing with them. This aspect, which was also considered by Ciccarese [30] in the medical field, will be a topic for our future research as well.

However, neither Weber nor Minor, in the works cited above, describe an automatic procedure for maintaining the case base, as we do with prototypes. On the other hand, in our opinion case base maintenance is extremely important in the BPM domain, since the case base can rapidly grow, and thus needs to be organized, in order to speed up retrieval, and to avoid redundancies. As observed, prototypes hierarchically organize the case base content, can make retrieval faster, and can also support long-term revisions of the process, an issue which is only partially afforded in [19]. By allowing the extraction of more generalized (and yet unstructured) knowledge from ground cases, prototypes reveal frequent similarities, that can be relied upon for schema revision. On the other hand, they allow not to disregard peculiar details—left in the ground cases features—which would be lost in rule/model-based reasoning systems. To our knowledge, the use of prototypes thus represents the most interesting and original contribution of our approach.

5 Conclusions

Exceptions provide great opportunities for a BPM system, to learn, correct itself and evolve. In this paper, we have described an approach for handling exceptions in BPM systems, based on the CBR methodology. The choice of CBR allows to automatically acquire and increase operative knowledge, without requiring a hard and time consuming formalization of knowledge itself, as it is needed by other methodologies, such as rule-based or model-based reasoning. Moreover, we resort to prototypes for case base maintenance. Prototypes allow the extraction of more generalized (and yet unformalized) knowledge from ground cases, and enable to organize the case base, thus making retrieval faster, and avoiding redundancies. Additionally, by retrieving prototypes, the process engineer can discover and analyse frequent modification, thus being supported in a long-term revision of the process schema.

From a technical viewpoint, some enhancements are foreseen as a future work. In particular, at the moment we don't take into account the textual exception motivations introduced by end users, neither in the retrieval nor in the maintenance phase. An effort to take advantage from this free text feature would be very relevant, as suggested by the results of our evaluation study, since users' motivations may help in better distinguishing between potentially ambiguous situations. Textual CBR is an emerging research area (see e.g. [31]), which is making available a set of suitable techniques for textual features representation and retrieval, that could be fruitfully exploited in our application.

Always according to the evaluation results, we will study how to make the graphical interface even more user friendly, in order to be easily adopted by all end users.

Moreover, we plan to develop a facility to automatically extract dependencies among cases belonging to the same exceptional situation, along the lines described in [19].

We will also study how to distinguish between changes in the process tasks and changes in the activity flow, as in [29].

Finally, we will continue our evaluation at Interporto di Rivalta Scrivia S.p.A. More data about the usefulness of the tool for long-term process schema revision, in particular, will be collected and analysed during the future evaluation activity. We also plan to make available a new release of the software, including the enhancements discussed above, as soon as possible. Extensions and adaptations in order to adopt the tool in different business contexts are also foreseen.

Acknowledgements The author is grateful to the personnel of Interporto di Rivalta Scrivia S.p.A. who is taking part in the evaluation activity, and in particular to Dr. Giacomo Mongini.

References

1. Heimann P, Joeris G, Krapp C, Westfechtel B (1996) Dynamite: dynamic task nets for software process management. In: Proceedings international conference of software engineering, Berlin, pp 331–341

2. Rinderle S, Reichtert M, Dadam P (2004) Correctness criteria for dynamic changes in workflow systems—a survey. Data Knowl Eng 50:9–34
3. Aamodt A, Plaza E (1994) Case-based reasoning: foundational issues, methodological variations and systems approaches. AI Commun 7:39–59
4. Surma J, Vanhoof K (1995) Integration rules and cases for the classification task. In: Veloso M, Aamodt A (eds) Proceedings of the 1st international conference on case-based reasoning, Sesimbra, Portugal, October 1995. Lecture notes in computer science, vol 1010. Springer, Berlin, pp 325–334
5. Branting LK, Porter BW (1991) Rules and precedents as complementary warrants. In: Proceedings of the of 9th national conference on artificial intelligence, Anaheim, CA, USA, July 1991. AAAI Press, Menlo Park
6. Bichindaritz I, Kansu E, Sullivan K (1998) Case-based reasoning in care-partner: Gathering evidence for evidence-based medical practice. In: Smyth B, Cunningham P (eds) Proceedings of the 4th European workshop on case-based reasoning, Dublin, Ireland, September 1998. Lecture notes in computer science, vol 1488. Springer, Berlin, pp 334–345
7. Gierl L, Stengel-Rutkowski S (1994) Integrating consultation and semi-automatic knowledge acquisition in a prototype-based architecture: experiences with dysmorphic syndromes. Artif Intell Med 6:29–49
8. Kolodner JL (1993) Case-based reasoning. Morgan Kaufmann, San Mateo
9. Watson I (1997) Applying case-based reasoning: techniques for enterprise systems. Morgan Kaufmann, San Mateo
10. Zhu J, Yang Q (1999) Remembering to add: competence-preserving case-addition policies for case base maintenance. In: Proceedings of the international joint conference on artificial intelligence. Morgan Kaufmann, San Mateo
11. Smyth B, McKenna E (1999) Building compact competent case bases. In: Lecture notes in computer science, vol 1650. Springer, Berlin, pp 329–242
12. Leake DB, Smyth B, Wilson DC, Yang Q (eds) (2001) Special issue on maintaining case based reasoning systems. Comput Intell 17(2):193–398
13. Maximini K, Maximini R, Bergmann R (2003) An investigation of generalized cases. In: Ashley KD, Bridge D (eds) Proceedings of the 5th international conference on case base reasoning (ICCBR'03), Trondheim, Norway, June 2003. Lecture notes in artificial intelligence, vol 2689. Springer, Berlin, pp 261–275
14. Riesbeck CK, Schank RC (1989) Inside case-based reasoning. Lawrence Erlbaum Associates, Hillsdale
15. Bergmann R, Wilke W (1996) On the role of abstraction in case-based reasoning. In: Lecture notes in artificial intelligence, vol 1186. Springer, Berlin, pp 28–43
16. Bareiss E, Porter B, Wier C (1988) Protos: an exemplar-based learning apprentice. Int J Man-Mach Stud 20:549–561
17. Reinartz T, Iglezakis I, Roth-Berghofer T (2001) On quality measures for case-base maintenance. Comput Intell 17:214–234
18. Casati F, Ceri S, Pernici B, Pozzi G (1998) Workflow evolutions. Data Knowl Eng 24:211–238
19. Weber B, Reichert M, Wild W (2006) Case-based maintenance for CCBR-based process evolution. In: Roth-Berghofer T, Goker M, Altay Guvenir H (eds) Proceedings of the European conference on case based reasoning (ECCBR) 2006. Lecture notes in artificial intelligence, vol 4106. Springer, Berlin, pp 106–120
20. Schmidt R, Gierl L (2001) Case-based reasoning for antibiotics therapy advice: an investigation of retrieval algorithms and prototypes. Artif Intell Med 23:171–186
21. Lieber J (2002) Strong, fuzzy and smooth hierarchical classification for case-based problem solving. In: van Harmelen F (ed) Proceedings of the 15th European conference on artificial intelligence (ECAI-02), Lyon, France. IOS Press, Amsterdam, pp 81–85
22. Wilson DR, Martinez TR (1997) Improved heterogeneous distance functions. J Artif Intell Res 6:1–34
23. Sadiq S, Marjanovic O, Orlowska M (2000) Managing change and time in dynamic workflow processes. Int. J. Coop. Inf. Syst. 9:93–116
24. VanderAalst W, Basten T (2002) Inheritance of workflows: an approach to tackling problems related to change. Theor Comput Sci 270:125–203
25. Dittrich KR, Gatziu S, Geppert A (1995) The active database management system manifesto: a rulebase of adbms features. In: Lecture notes in computer science, vol 985. Springer, Berlin, pp 3–20
26. Beckstein C, Klausner J (1999) A planning framework for workflow management. In: Proceedings of the workshop on intelligent workflow and process management, Stockholm
27. Madhusudan T, Zhao JL, Marshall B (2004) A case-based reasoning framework for workflow model management. Data Knowl Eng 50:87–115
28. Luo Z, Sheth A, Kochut K, Miller J (2000) Exception handling in workflow systems. Appl Intell 13:125–147
29. Minor M, Tartakovski A, Schmalen D, Bergmann R (2008) Agile workflow technology and case-based change reuse for long-term processes. Int J Intell Inf Technol 4(1):80–98
30. Ciccarese P, Caffi E, Boiocchi L, Halevy A, Quaglini S, Kumar A, Stefanelli M (2003) The newguide project: guidelines, information sharing and learning from exceptions. In: Proceedings of the artificial intelligence in medicine Europe (AIME) 2003. Springer, Berlin, pp 163–167
31. Wiratunga N, Lamontagne L (2006) In: Workshop on textual case based reasoning: reasoning with text, European conference on case based reasoning (ECCBR), Oludeniz