Service Discovery and Composition: PreDiCtS Approach

Charlie Abela
University of Malta
Department of Computer Science and AI
+356 2590 7295
charlie.abela@um.edu.mt

Matthew Montebello
University of Malta
Department of Computer Science and AI
+356 2340 2132
matthew.montebello@um.edu.mt

ABSTRACT
The proliferation of Web Services is fostering the need for service-discovery and composition tools to provide more personalisation during the service retrieval process. In this paper, we describe the motivating details behind PreDiCtS, a framework for personalised service-retrieval. In our approach we consider that similar service composition problems can be tackled in a similar manner by reusing and adapting past composition best practices or templates. The proposed retrieval process uses a mixed- initiative technique based on Conversational Case-Based Reasoning (CCBR), that provides i) for a clearer identification of the user's service requirements and ii) based on these requirements, finds suitable service templates that satisfy the user's goal. We discuss how retrieval can vary through the use of different CCBR algorithms and how adaptation can be performed over the retrieved templates thus providing the personalisation feature in PreDiCtS.

Categories and Subject Descriptors
I.2.6 [**Artificial Intelligence**] Learning

General Terms
Algorithms, Performance, Design, Experimentation.

Keywords
Web Services, Conversational Case-Based Reasoning, Semantic Web

1. INTRODUCTION
Reusability and interoperability are at the core of the Web Services paradigm. This technology promises seamlessly interoperable and reusable Web components that facilitate rapid application development and integration. When referring to composition, this is usually interpreted as the integration of a number of services into a new workflow or process. A number of compositional techniques have been researched ranging from both, manual and semi-automatic solutions through the use of graphical authoring tools, discussed in [24], [22], to automated solutions based on techniques such as AI planning, used in [20], [25] and others.

The problem with most of the composition techniques mentioned above is three fold (i) such approaches attempt to address service composition by composing web services from scratch, ignoring reuse or adaptation of existing compositions or parts of compositions, (ii) it is assumed that the requester knows exactly what he wants and how to obtain it and (iii) composing web services by means of *concrete* service interfaces leads to tightly-coupled compositions in which each service involved in the chain is tied to a Web service instance. Using this approach for service reuse, may lead to changes in the underlying workflow which range from slight modifications of the bindings to whole re-designing of parts of the workflow description. Therefore in our opinion, services should be interpreted at an abstract level to facilitate their independent composition. [13] adds, "*abstract workflows capture a layer of process description that abstracts away from the task and behaviour of concrete workflows*", and this allows for more generalisation and a higher level of reusability. A system can start by considering such abstractly defined workflow knowledge and work towards a concrete binding with actual services that satisfy the workflow.

The reuse of abstract workflows brings with it a set of other issues, such as the way that these workflows are generated, stored and retrieved. Therefore when deciding on which solution to adopt we considered the following motivating points:

- Reusability of compositions has the advantage of not starting from scratch whenever a new functionality is required.
- For effective reusability, a higher level of abstraction has to be considered, which generalises service concepts and is not bound to specific service instances.
- Personalisation of compositions can be achieved by first identifying more clearly the user's needs and then allowing for reuse and adaptation of these past compositions based on these needs.
- Compositions can be bound with actual services thus making them concrete.

In our approach we wanted to put the user (developer or otherwise) in a situation whereby he can reuse existing templates. Infact this approach is similar to that adopted in [21], [25], [11], and [27] which use pre-stored abstract workflow definitions or templates in their composition framework.

This kind of reusability has been widely investigated in work related to Case-Based Reasoning (CBR), which is amenable for storing, reusing and adapting past experience for current problems. Nevertheless CBR restricts the user to define a complete problem definition at the start of the case-retrieval process. Therefore a mixed-initiative technique such as CCBR [6] is more appropriate since it allows for a partial definition of the problem by the user, and makes use of a refinement process to identify more clearly the user's problem state.

In this paper we want to present, the motivation behind, and a prototype of the PreDiCtS framework. Through this framework we allow for i) the encoding and storing of common practices of compositions or templates within cases and ii) for the retrieval, reuse and adaptation of these cases through CCBR.

PreDiCtS' case definition is based on the CCBROnto ontology. This ontology is based on OWL and has been discussed in [3] and [4]. Each case definition is composed of three main components

that capture different knowledge related to a particular service template. The case-context defines information related to the case creator and provides a means through which a case-utility history is maintained. Each case encodes the problem to which it can provide a solution in the form of a set of question-answer pairs (qapairs). It is through this set of qapairs that the retrieval process can present the solution which represents the service workflow definition. Each solution is defined through an OWL-S [19] service definition. This includes both service profile and process descriptions. The latter is important since it defines the actual service workflow information.

Given a new problem or service template request, the PreDiCtS' approach allows first to retrieve a ranked list of past, similar templates which are then ranked and suggested to the requester. Through a dialogue process the requester can decide when to stop this iterative-filtering phase, and whether to reuse or adapt a chosen case.

In a future extension to this work it is envisioned that, given a suitable case, a mapping is attempted between the features found in the chosen template, to actual services found in a service registry. An AI planning component can be used at this stage to handle this mapping from an abstract to a concrete, executable workflow.

The rest of this paper is organized as follows. In Section 2 we will give some brief background information on CCBR and its application in various domains. In Section 3 we will present the architecture of PreDiCtS and discuss implementation details mainly focusing on the case-creator and case-retriever components, making references to a typical travelling scenario. We evaluate the prototype in Section 4 and in Section 5 we discuss future work and extensions. In the final section we provide some concluding results.

2. CONVERSATIONAL CASE-BASED REASONING

Case-Based Reasoning is an artificial intelligence technique that allows for the reuse of past experience to solve new problems. The CBR process requires the user to provide a well-defined problem description from the onset of the process, but users usually cannot define their problem clearly and accurately at this stage. On the other hand, CCBR allows for the problem state to be only partially defined at the start of the retrieval process. Eventually the process allows more detail about the user's needs to be captured by presenting a set of discriminative and ranked questions automatically. Depending on the user's supplied answers, cases are filtered out and incrementally the problem state is refined. With each stage of this problem refinement process, the system presents the most relevant solutions associated to the problem. In this way the user is kept in control of the direction that this problem analysis process is taking while at the same time she is presented with solutions that could solve the initial problem. If no exact solution exists, the most suitable one is presented and the user is allowed to adapt this to fit her new requirements. Nevertheless, this adaptation process necessitates considerable domain knowledge as explained in [18], and is best left for experts.

One issue with CCBR is the number of questions that the system presents to the user at each stage of the case retrieval process. This issue was tackled by [14] which defined qapairs in a taxonomy and by [2] through the use of knowledge-intensive similarity metrics. In PreDiCtS we took into account the possibility that the user opts to use different similarity measuring algorithms for different domains. Infact two approaches are allowed (with the possibility of adding others). One of these approaches is based on the similarity measure defined in [6] and used by [26] to handle workflow reuse. Another similarity measure is based on the taxonomic theory defined in [14]. Through this similarity technique, the abstract relations between qapairs and in particular the sub-class relation are considered to reduce the number of questions that the user is presented in each retrieval cycle.

2.1 Uses of CCBR

CCBR is mostly associated with customer-support systems, though its benefits have been tested in various other fields such as, business process and workflow management, software-component retrieval and in connection with Recommendation systems. In what follows we will consider the above scenarios in more detail.

2.1.1 Business Process and Workflow Management

Weber in her thesis [26] combines CCBR with rules and presents a solution for business process management. The created prototype is called CBRFlow and allows for more flexibility and adaptability in the management of workflows. The adopted hybrid approach takes the best of rule-based and case-based reasoning, though the rule-based component is allowed to have some precedence over the case-based component. Rules are generated from domain knowledge while case-based reasoning is used when no rules are available or updates to a rule exist in the form of cases.

The CCBR component uses the same case-similarity metric as that described by [6]. This similarity is computed by finding the difference between the number of the shared and conflicting observations, and then dividing the result by the total number of observations in a case. A normalisation function is used to set the result within the interval [0, 1].

2.1.2 Software Component Retrieval

In [1] the CCBR technology is used to solve the problem of software component retrieval, especially when the number of components involved is large. The proposed solution is called Conversational Component Retrieval Model or CCRM. A case represents a component and a knowledge-intensive CBR methodology is used to explore the context-based similarities between the user's query and the stored components.

A frame-based knowledge representation and reasoning system called CREEK [10] is used to unify the component-specific cases and the general domain knowledge. A knowledge-intensive similarity calculation is used to determine which knowledge in the knowledge base is relevant to the retrieval process and to calculate the similarity between a new case and the stored cases.

The question-answer interaction during the conversation is motivated by the fact that qapairs are easily understood and that the most informative and discriminating ones are presented to the user during a conversation. For this reason a set of predefined questions together with possible answers for each slot (i.e. for each relation between two concepts) are specified and an

information-gain metric algorithm is used to quantitatively measure the information that each slot can provide.

In our work we intend to resort to such frame structures through the use of OWL ontologies, in particular CCBROnto. We define cases whose solutions are service templates. These templates will be defined through a process definition language, such as OWL-S, though it is possible to use other languages, such as WS-BPEL.

2.1.3 CCBR and Recommendation Systems

[18] presented a web-based CCBR solution which is able to recommend solutions to scientist seeking resources (such as codes and data) related to an Earthquake Simulation Grid provided by the ServoGrid project [23].

A number of grid related ontologies were developed in RDF and these are used to represent case descriptions. Thus a case is considered to be a set of RDF triples. A domain independent CBR engine based on the Indiana University Case-Based Reasoning Framework (IUCBRF) [16] is used.

The implemented prototype uses the RDF ontologies to present questions about the desired resource characteristics and, typically to the CCBR process, which ranks cases based on the chosen answers. During each iteration, the system provides discriminating questions in a ranked order so that the irrelevant cases are incrementally filtered out.

Each case definition contains the problem and solution descriptions together with bookkeeping information such as the time of case creation, the contexts in which the case applies and also source or provenance information. Both the problem and solution are represented by a set of predefined features, where each feature is an RDF triple. During case-base initialisation, all possible <*predicate - predicate value*> pairs are extracted from the ontology and presented as features. The case retrieval mechanism is based on a threshold method which compares the set of features present in both user and case-problem definitions. Cases are ranked based on the number of common features whose values are consistent. Cases with unknown features or having inconsistent feature values are eliminated from the process.

The way in which cases are defined through RDF is consistent with how we envision our own solution. Though in this case, all generated triples are equally considered as possible qapairs. Furthermore, it seems that no reasoning was done on the RDF data, thus no advantage was taken from this when qapairs were presented to the user. In our solution we want to be able to exploit as much as possible the logic behind the concepts and relations within a case description by using an OWL reasoner. For example given that, a question related to some particular concept has already been presented to the user, it is superfluous to present another question whose concept is more generic than the one associated with the previous question.

2.2 Taxonomic CCBR

Taxonomic CCBR (TCCBR) tries to tackle the pervasive issue of expressing case contents and features at different levels of abstraction. The solution is based on the ability to make use of feature taxonomies.

The motivation behind the use of TCCBR is highlighted by three sources of abstraction:

- the different levels of domain expertise between users and developers
- the variations in information availability and the cost of acquiring it
- the variations in decision-making needs

If abstraction is ignored then problems such as unwanted correlation between features, redundancy in the number of questions presented to the user during conversation and inconsistencies in the case representations when new features are added are most likely to occur. TCCBR is defined to include:

- A set of questions which are used for indexing the cases. Each question can be associated with a set of answers.
- A set of taxonomies each one representing a set of qapairs which are related through either an *is-a-type-of* or *is-a-part-of* relation.
- A set of cases each having a problem definition in the form of a set of qapairs and a solution.

Furthermore, in TCCBR two important rules have to be applied to the set of qapairs in a defined case:

i. Only one qapair from a particular taxonomy can be included in each case

ii. The most specific available and applicable qapair is used to represent the case

The process of TCCBR as explained by [14] is divided into three main tasks (the third is optional though):

i. Case retrieval

ii. Conversation

iii. Case Creation

Case retrieval is subdivided into three main steps referred to as searching, matching, ranking and selecting. During this phase, cases are retrieved and ranked based on the questions that the user has chosen to answer. On the other hand the conversation process involves the identification and ranking of the most appropriate questions to present to the user after each iteration. If no suitable solution is found then a new case may be defined by specifying a new set of questions (or reuse existing questions) and a solution for this new problem.

The approach taken in TCCBR is very relevant to our research goal and infact this is one of the retrieval techniques adopted in PreDiCtS. The main theory behind TCCBR is discussed in detail in [14] and though we will make reference to this work we will not explain it here. Nevertheless in what follows we will explain in detail any deviations that we have taken from this original theory.

3. PreDiCtS

The PreDiCtS framework allows for the creation and retrieval of cases (the adaptation process is in the pipeline). The respective components that perform these two tasks are the *CaseCreator* and the *CaseRetrieval* (See Figure 1). PreDiCtS is written in Java and is developed in Eclipse. It uses a MySQL database to store the cases, which are based on CCBROnto, and makes use of both Jena and the OWL-S APIs.

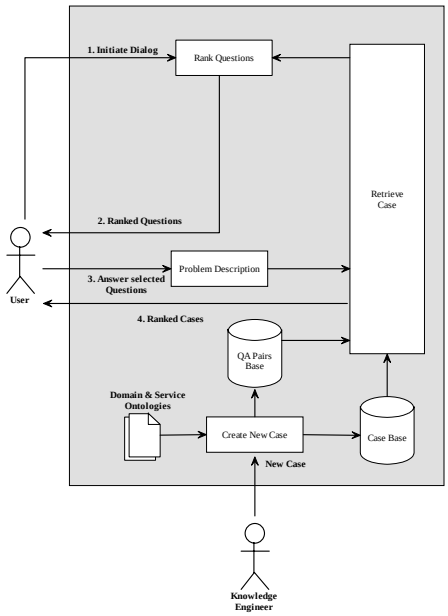

Figure 1: CCBR cycle adopted in PreDiCtS

To explain how PreDiCtS can be used to create and retrieve service templates we will make use of a typical travelling scenario, described in the next section. We will then explain how cases, which represent different problems related to this domain and their respective solutions, are created through the *CaseCreator*. Retrieval is handled by the *CaseRetrieval* component which allows the user to adopt different CCBR algorithms to find the cases with the most suitable service template. Figure 2 represents the main components in our framework.

3.1 Travelling Scenario

The travelling situation that we want to model here is related to an academic who wants to go abroad to attend a conference. The defined cases should represent the problem from an advisor's perspective and present a solution based on this knowledge. An advisor in this situation could have the role of a travelling agent, who is asking his client questions to identify what the latter requires so that he can eventually suggest the best solution.

Goal: User is to attend an **event** on some particular **date** in some particular **location**. A part of a travelling domain ontology is shown in Figure 3.

Looking at the ontology it is noticed that the concept *Person* is associated with three disjoint branches or taxonomies, *Event*, *Accommodation* and *Transport*. Thus the questions should be related to any of these taxonomies.

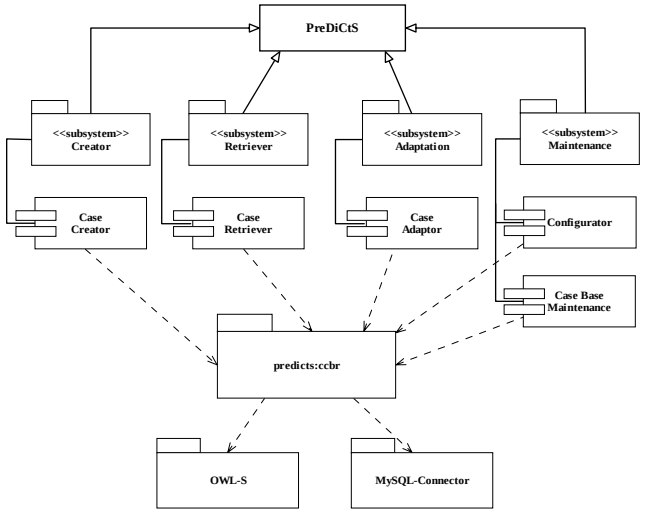

Figure 2: System Component Summary

After having identified the important aspects of the domain, we start by looking at the ontology to identify which typical questions might be asked in this situation. Questions should ideally capture a single aspect of the domain. For example, the most generic questions that are immediately identified are: *Do you want to attend for an Event?*, *Do you want to use Transport?* and *Do you want to reserve an Accommodation?*. Other questions, such as *Do you want to use a Plane?* And *Do you want to stay in a Hotel?* can be considered as being subsumed by the former set. The associated answer types for such questions are typically either a *Yes* or a *No*.

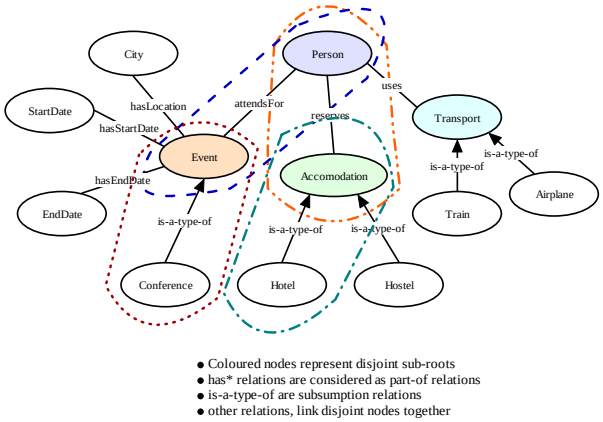

Figure 3: Travelling Domain Ontology

In Table 1 below we have listed some of these questions and have also associated them with a triple from the ontology. Each qapair is assigned a unique QID reference for that particular set. The appropriate link between the set of qapairs and the solution has to be defined by the creator. We adopt the methodology presented by [17] in which domain and task-knowledge are linked together.

Table 1: qapairs set for the Travelling Domain

QID	Description	Triples Set
		<Subject, Predicate, Object>
1	Problem is a Travelling Problem?	<TravellingProblem, subClassOf, Problem>
2	Do you want to attend a Conference?	<AttendConference, subClassOf, TravellingProblem>
3	Do you need transportation?	<Transportation, subClassOf, AttendConference>
4	Do you want to use a plane?	<Airplane, subClassOf, Transportation>
5	Do you want to use a train?	<Train, subClassOf, Transportation>
6	Do you want accommodation?	< Accommodation , subClassOf, AttendConference >
7	Do you want to stay in a hotel?	<Hotel, subClassOf, Accommodation>
8	Do you want to stay in a hostel?	<Hostel, subClassOf, Accommodation>
9	Is Conference registration required?	<Conference, subClassOf, AttendConference>

The domain is used to provide datatype information relevant to the service inputs and outputs. In our case the task knowledge is defined through an OWL-S definition. Thus for example the triple *<Hotel, subClassOf, Accommodation>* will provide, in the solution, a generic place holder for a *Hotel Reservation* service. Other services that might be useful to include in the solution are *Flight Booking*, *Train Reservation*, *Hostel Reservation* and *Conference Registration* services. Figure 4 represents a UML Activity Diagram of a particular solution for this domain. The use of this graphical representation to define a service workflow has also been adopted by [22] and [8].

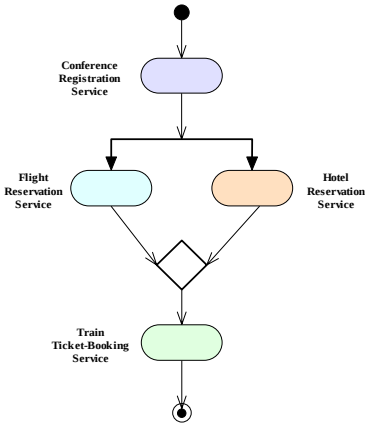

Figure 4: UML Activity Diagram for the Service Workflow

3.2 Case Creation

During case creation the expert user can define and add a new case to the case base. As already explained earlier, in CCBR a case consists of a case description and a problem and solution state. In PreDiCtS though, a case c_i is defined as a tuple:

$c_i = (dsc_i, cxt_i, \{q_1a_3....q_ia_j\}, act_i, hst_i)$ where;

dsc_i is a textual description for the particular case.

cxt_i represents a set of context related features, such as *Role* and *CaseCreator* information based on the *foaf:RDF* ontology definition.

$\{q_1a_3....q_ia_j\}$ is a representation of the problem state by a set of qapairs

act_i denotes the solution which is represented by service composition knowledge stored in an abstract template.

hst_i, is the usage history associated with each case.

Each case is based on the CCBROnto ontology which is described in more detail in [4] and can be found at [9]. The *CaseCreator* UI (see Figure 5) allows the user to add all the necessary information which is then translated into a CCBROnto case-representation in a manner transparent to the user.

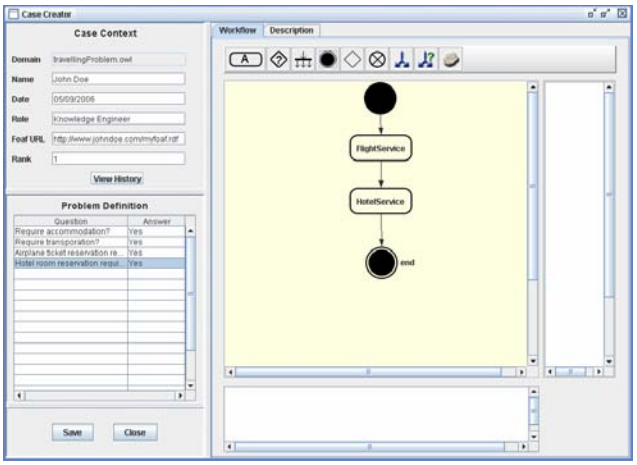

Figure 5: CaseCreator UI in PreDiCtS

The context description *cxt*, is important in an open-world such as the Web since this will be used as a discriminating feature during case retrieval. The action or solution definition *act*, represent the service compositional knowledge and can be defined through any composition language. In PreDiCtS we are using parts of an OWL-S service description, but the framework can be easily extended to work with other service languages such WS-BPEL. hst_i is an other feature which represents the usage-history of each case. This history could provide either positive (i.e. case was found useful) or negative feedback (i.e. implying that aspects of the case were not found ideal by past users) to the case user. This history information is used to generate a reputation value during case retrieval.

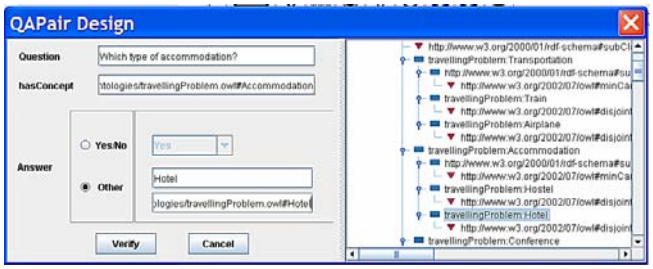

Figure 6: Question-Answer Pair Design Tool

An important aspect to consider when creating a new case is the definition of the problem through a set of qapairs. PreDiCtS' retrieval component uses two approaches to find suitable cases, one of which is based on an adapted version of TCCBR. For this reason a special qapairs-creation tool, shown in Figure 6, is provided that allows the user to easily associate a new qapair definition with domain ontology concepts.

Some adaptations have been made to the TCCBR theory to allow the system to work with ontologies and to be able to handle the open-world aspects required when defining the service template.

3.2.1 Rule 1

Only one qapair from a taxonomy can be included in a case (i.e. there is no abstract relation between concepts relating each qapair

in case). This is similar to TCCBR, unless these concepts associated to these qapairs are specifically defined as disjoint within the taxonomy.

Example:

```
<owl:Class rdf:ID="Hotel">
   <rdfs:subClassOf rdf:resource="#Accommodation"/>
   <owl:disjointWith rdf:resource="#Hostel"/>
</owl:Class>
<owl:Class rdf:ID="Hostel">
   <rdfs:subClassOf rdf:resource="#Accommodation"/>
   <owl:disjointWith rdf:resource="#Hotel"/>
</owl:Class>
```

Given the above situation a case can contain both questions:

Accommodation required is Hostel?

Accommodation required is Hotel ?

In this way the case covers the situation whereby a user might require staying at both a *Hotel* and a *Hostel* which are both subclasses of *Accommodation.*

3.2.2 Rule 2

The most specific available and applicable qapair is used to represent the case. We adapt this rule as is defined in the TCCBR theory. We look at a taxonomy as a dialogue composed of an ordered set of nodes (qapairs). We start by looking at both the domain of discourse and the different services that might be required (in the solution) to solve a particular issue. We extract those classes that are relevant to the problem that we want to model and give them an ordering. This ordering, though abstractly defined through the *subClassOf* relation, does not always imply that one class is in effect a *subClassOf* another, but rather that the question associated with that concept will be asked before or after another one associated with another concept. Therefore given the questions:

Accommodation required is Hotel? and *Do you need Accommodation?*, the former will be preferred over the latter because it is more specific and thus is considered to be closer to the solution.

3.2.3 Rule 3

We consider a qapair to be associated with a unique concept in the taxonomy. Thus for example, the question:

Accommodation required is Hostel? will be associated to the *Hostel* concept while *Do you need accommodation?* is associated to the *Accommodation* concept

We make use of reification to generate more knowledge about each statement. Infact a question will be associated with a reified statement that threats each component of a triple <*subject*, *predicate*, *objec*t>, as a *Resource*.

Example: For the question *Do you need accommodation?*

a reified statement with the following subject, predicate and object resources will be defined:

Subject: *Accommodation*

Predicate: *subClassOf*

Object: *AttendConference*

In this example, *Accomodation* is defined to be a *subClassOf AttendConference*. We envision that this technique will allow us, in the future, to work with other types of abstract relations such as those similar to *is-a-part-of* by considering other properties that associate classes together.

3.2.4 Service Template Creation

The case creator is provided with a visual-composer tool that allows him to easily create a workflow with the generic services that can solve a specific problem. The UML Activity Diagram representation is used to eventually generate an OWL-S Process definition. The Process ontology in OWL-S provides for the definition of a workflow of services and related properties. Since we wanted this description to be as generic as possible, each service definition is conceptually linked to an ontology of service-related concepts. Thus if the user adds a node that represents a *Flight Reservation* service, a generic atomic service definition will be generated whose input and output resources are defined by some external service-related ontology.

```
<process:AtomicProcess rdf:ID="FlightReservationService">
  <process:hasInput>
    <process:Input rdf:about="#FlightReservationInput">
      <process:parameterType
            rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"/>
    </process:Input>
  </process:hasInput>
  <process:hasOutput>
    <process:Output rdf:about="#FlightReservationOutput">
      <process:parameterType
            rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"/>
    </process:Output>
  </process:hasOutput>
</process:AtomicProcess>
```

In this manner when searching for actual services, these generic placeholders will be bound to actual service inputs and outputs. The workflow also defines information related to the order of execution of the services.

3.3 Case Retrieval

The *CaseRetriever* is responsible for the CCBR retrieval process. It takes as input the choice of similarity measure and problem domain and presents questions for the user to answer. The answered questions will then be used to generate a list of cases based on the similarity measure component.

It is up to the user to decide whether a case from the retrieved set of cases is suitable enough to solve his problem. In the situation where further problem-filtering is required, the user can decide to answer more questions, with the consequence that the list of retrieved cases is also filtered down.

The set of questions presented with every step in this filtering process are generated through a conversation-generation component which takes care of identifying which questions are best suited to be presented to the user in the next step. Different conversation-generation algorithms are available in PreDiCtS, depending on the type of similarity measure chosen initially by the user.

3.3.1 CaseRetriever UI

This is divided into three main components (see Figure 7). The top-most pane consists of two combo boxes; one displays a list of problem-domain ontologies while the other displays the different types of similarity methodologies that the retriever is capable of

using. At present this is limited to two, the *Default* CCBR and the *TCCBR* (Taxonomic CCBR) methodologies. We envision the use of a graph-based retrieval method, *GCCBR*, based on [7], in the near future.

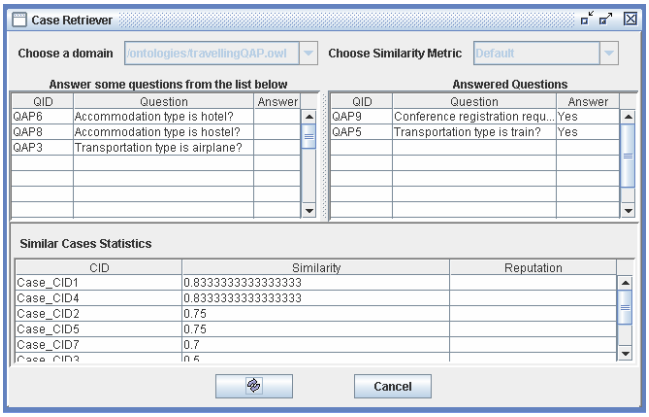

Figure 7: CaseRetriever UI

The middle left and right panes show two tables. The table on the left presents those questions that still require an answer from the requester. These are generated by the conversation-generation algorithm. They act as a filtering and refinement mechanism for the list of retrieved cases. The middle right-hand pane presents a table of already answered questions. The bottom pane also contains a table component, but this one shows the cases that have been retrieved during a CCBR retrieval cycle.

3.3.2 Default CCBR

This similarity measure is based on that presented by [6] and discussed in Section 2. The following similarity function is used:

$$sim_1(Q,C) = \frac{same(Q_{qa},C_{qa}) - diff(Q_{qa},C_{qa})}{|C_{qa}|}$$

where $|C_{qa}|$ represents the number of qapairs in the case problem definition. This function is also adapted by [26] but a normalisation function is also included to keep the similarity value between [0, 1]. The normalisation function is as follows:

$$sim_2(Q,C) = \frac{1}{2} * (sim_1(Q,C) + 1)$$

The technique used to present and rank new questions during a conversation is based on their frequency in the considered cases, though other different techniques could be utilised as defined by [2], [15] and others.

3.3.3 TCCBR

As discussed earlier in Section 2, retrieval in TCCBR is based on two processes, the case-retrieval process and the conversation generation process. We will discuss the steps associated to each one and any adaptations made in PreDiCtS.

The searching step, of the case-retrieval process in the original TCCBR starts by getting the user's textually-defined query and mapping this with the most similar qapair in the specific domain. In PreDiCtS though, the system presents the list of the most generic qapairs for a chosen domain, that is, those that are at the roots of the specific taxonomies.

The qapairs are not directly taxonomised, as explained earlier, but each question is associated with a triple *<subject, predicate, object>* which is defined in a problem related-ontology. This implicitly makes a qapair part of a taxonomy.

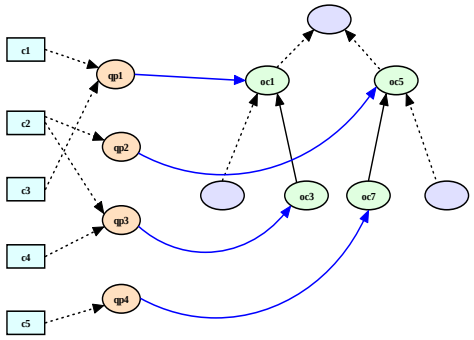

Figure 8: PreDiCtS qapair Taxonomy

Considering the situation in Figure 8 above, we assume that the set of cases, C_Q, and the set of qapairs, QP, are linked to an Ontology O as follows:

case set $C_Q = \{c_1, c_2, c_3, c_4, c_5, \ldots, c_i\}$

qapairs in $QP = \{qp_1, qp_2, qp_3, \ldots, qp_j\}$

ontology $O = \{oc_1, oc_3, oc_5, oc_7, \ldots, oc_k\}$

where, qp_1 can be seen to be present in two cases, c_1 and c_3, and is associated with ontology concept oc_1 (this is assumed to be the *subject* from the associated triple). Similarly qp_3 is associated with ontology concept oc_3 (which is the *subject* in yet another triple) and is present in two cases, c_2 and c_4. In this way qp_1 subsumes qp_3 based on the relation between the ontology concepts oc_1 and oc_3. Therefore during a case retrieval cycle, qp_1 will be asked before qp_3 since it is more generic. In a case c_i there will either be qp_1 or else qp_3, as per Rule 2 above, this provides for a reduction in the redundant questions being presented to the user during case retrieval.

The second step in the case-retrieval process is the matching step and in the original TCCBR this involves matching each qapair element in QP with the qapairs in each of the candidate cases. A ranked list of cases is established based on the following similarity measure $sim(qp_i, p_j)$:

$$sim(qp_i, p_j) = \begin{cases} 1 & \text{if } p_j \subseteq qp_i \\ (n+1-m)/(n+1+m) & \text{if } qp_i \subseteq p_j \\ 0 & \text{otherwise} \end{cases}$$

where,

qp_i is the question-answer pair in the user's query and

p_j is the question-answer pair in a candidate case

n = number of edges between qp_i and the root of the taxonomy

m = number of edges between qp_i and p_j

Having calculated such similarity between qapairs then an aggregate similarity metric is used to calculate the overall similarity between the user query QP and a case problem description, P_k. This aggregate similarity is calculated as follows:

$$Sim\ (QP, P_k) = \frac{\sum_{i \in QP, j \in P_k} sim(qp_i, p_j)}{T}$$

where, T in the original taxonomic theory represents the number of taxonomies, here it represents the number of disjoint branches in the domain ontology, that are associated with the qapairs. Cases are then ranked in a descending order based on this aggregate value.

In PreDiCtS we adopt the same similarity metric except that this similarity is computed on the qapairs' associated concepts rather then on the qapairs themselves. With reference to Figure 8 above, suppose that in the user's problem definition there are two qapairs, qp_1 and qp_4, while in the problem definition of the candidate case there are three, qp_3, qp_2 and qp_5.

Based on the associated concepts oc_1 and oc_3, the qapairs qp_1 and qp_3 are related by a parent-child relation, while the qapairs qp_4 and qp_2, which are associated with the concepts oc_7 and oc_5, are bound by a child-parent relation (see Table 2).

Table 2: qapair/Concept relations

User's Query		Case	
q-a pair	**concept**	**q-a pair**	**concept**
qp_1	oc_1	qp_3	oc_3
qp_4	oc_7	qp_2	oc_5
		qp_5	oc_2

The user's query though, does not contain a qapair in the candidate case that is related with qp_5. Using the adapted similarity metrics defined by TCCBR, we assume the following values:

$$sim\ (qp_1, qp_3) = sim\ (oc_1, oc_3) = 1$$

$$sim\ (qp_4, qp_2) = sim\ (oc_7, oc_5) = 0.5$$

for which the aggregate similarity will be

$$\sum sim\ (oc_i, oc_j) = \frac{1 + 0.5}{3} = 0.5$$

where the number of taxonomies here is 3 since:

a. the concepts oc_1 and oc_5 represent 2 disjoint branches in the ontology and

b. the concept oc_2 has to be considered as a separate disjoint branch.

The last phase of the case-retrieval takes the set of cases that are retrieved and rank-orders them in descending order based on the similarity score obtained from the previous step. Both the TCCBR and the PreDiCtS theories handle this step in a similar manner.

The conversation algorithm in PreDiCtS is the same as that defined by the TCCBR theory. The goal is to present the user with a ranked list of questions derived from the retrieved cases C_R. The process starts by considering all qapair taxonomies applicable to the cases in C_R. The score of each qapair in a taxonomy is based on the similarity scores obtained for the node-related cases. Each node takes the score of all related cases and the similarity of each parent node is the accumulation of the scores of its child nodes. A backward pass algorithm is used to calculate the score of each node. If the user problem definition contains a qapair from the taxonomy then the system selects its child nodes, else the most specific node that subsumes the set of retrieved cases is selected.

The question score is a tuple that includes <*taxonomy score, q-a pairs score*> as in Figure 9.

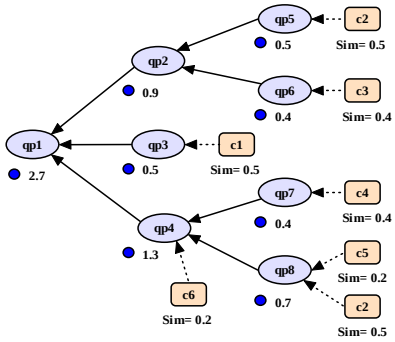

Figure 9:Question-Answer pair scoring

Example: $s(qp_4) = 1.3 = ((Sim_{c6} = 0.2) + (s(qp_7) = 0.4) + s(qp_8) = 0.7))$

4. EVALUATION

In this section we will present the main results of the tests that were carried out to evaluate the case retrieval performance in PreDiCtS (for a more detailed description see [5]). We considered a number of different requests to find suitable cases relevant to the travelling domain.

Table 3: Cases for the Travelling Domain

Case_ID	Solution	Problem
CID1	Conference + Hotel + Train	QID 5, QID6, QID9
CID2	Conference + Hotel + Airplane + Train	QID3, QID5, QID6, QID9
CID3	Conference + Hotel + Airplane	QID3, QID6, QID9
CID4	Conference + Hostel + Train	QID5, QID8, QID9
CID5	Conference + Hostel + Airplane + Train	QID3, QID5, QID8, QID9
CID6	Conference + Hostel + Airplane	QID3, QID8, QID9
CID7	Conference + Hostel + Airplane + Hotel + Train	QID3, QID5, QID6, QID8, QID9
CID8	Conference + Hotel	QID6, QID9
CID9	Conference + Hostel	QID8, QID9

A list of relevant cases (see Table 3) was developed, where each case had a unique CID (case ID). The problem definition consisted of sets of qapairs' ID references where each QID was also unique for the particular domain. The solution represented a set of services that were stored in a service template.

Four problems were identified and the performance of both the *Default CCBR* and the *TCCBR* approaches was considered. The identified problems were:

i. *The person (requester) wants to register for a conference, travel by train and stay in a hotel.*

ii. *The person (requester) wants to attend for a conference, requires accommodation and transportation.*

iii. *The person (requester) wants to travel by airplane for a conference and requires accommodation.*

iv. *The person (requester) wants to attend for a conference, has to travel by airplane and train and stay at a hotel.*

The main differences between the two approaches were in:

1. the number of questions that were presented to the requester by the conversation-generation algorithm, at the end of each retrieval cycle.
2. the accuracy of the similarity values.
3. the effect of leading the requester towards the Most Suitable Case (*MSC*).

The reason behind the first result is attributed to the fact that TCCBR considers the abstract relations between qapairs and thus limits the number of redundant questions to present to the user at each stage. This though, does not come without an initial effort on behalf of the case base designer. Infact time has to be dedicated to create suitable qapairs taxonomies that reflect a particular problem domain and then to associate these to the appropriate cases.

The second and third results both depend on the first one. The former is due to the fact that redundant qapairs are not considered in the case-similarity computation and this gives a more accurate figure at the end. The latter is the end result of the taxonomic aspect when designing the qapairs set. Infact this leading effect to solution-finding in more pronounced in the TCCBR then in the Default CCBR. Though again this is highly dependent on the design aspect of the case base.

5. FUTURE WORK

In this section we will consider how this work can be extended and improved by the inclusion of an adaptation component and feedback mechanism.

5.1 Adaptation

The addition of this component to PreDiCtS completes the CCBR cycle. Adaptation is closely related to the retrieval process, since it can be considered as the personalisation of a retrieved case to suit more effectively the requester's needs.

The main issue that has to be considered is the decomposition of a case into its basic components and then the ability to adapt each component separately. An adapted case will then be added to the case base and tested to ascertain its effect on the case base, after which it might be re-adapted or retained in the case base.

The adaptation of cases can be considered as a personalisation process through which the requester or designer can change aspects of a case to provide a more suitable problem-solution relation.

The possible changes can include:

- the addition and removal of services from the template definition (or solution)
- editing of input/output for particular services, including changes to associated ontological concepts
- changes in the order of execution of services or changes to the control constructs used
- adding or removing qapairs that make up the problem definition. Editing qapairs is a more complex process and will surely have a negative effect on the case base. So great care has to be taken in this case.

The most important aspect of this adaptation component will be the UI that provides the visualisation of all subcomponents of the case requiring adaptation. A mapping from the template definition back to UML activity diagrams is required. This will provide an easy way to adapt the solution. This component will be accessed also from within the retrieval component, where it would be possible to adapt the *MSC* to obtain a higher similarity to the problem at hand.

5.2 Feedback Mechanism

In Section 3 we have mentioned the importance of having a feedback mechanism included in PreDiCtS. It is a process that strives to maximise the usefulness of the case base.

We have identified that such a mechanism can help:

- the requesters to identify how a case has been used, by whom (here we refer to a process which clusters users not the actual person) and whether it has been reputed useful.
- the designer when importing cases from third-parties, since, based on the reputation of cases, then he can decide whether to import or not further cases from this source.
- the designer when performing maintenance on the case base; those cases that have a negative reputation may be removed from the case base.

Such reputation mechanism may be based on user feedback such as that discussed in [27] and in [15]. It is similar to the feedback mechanism of recommendation systems and considers the overall feedback given by case users to generate a reputation score. This score represents the usefulness of a particular case to solve a specific problem. For this reason we have included in CCBROnto a way to structure this information as part of the case history. We have also identified the need to compute a trust value, for the case creator. This trust value will be based on the global reputations of the cases provided by that particular case supplier. If the overall case-reputation is less then a certain threshold then this implies a lower trust level and that source will not be used any more. In a way this is similar to how eBay [12] reputes its sellers and buyers, though PreDiCtS will take action and remove this source from the list of possible case-suppliers.

6. CONCLUSION

In this paper we presented the motivation behind PreDiCtS. The use of the underlying CCBR technique as a pre-process to the service discovery and composition is promising since it provides for inherent personalisation of the service request and thus as a consequence also more personalised compositions. The tests that were performed showed that the design of both the case base and qapairs affects the retrieval process and to some extend, this also depended on the similarity measure. The most important difference between these two similarity measures was infact the number of relevant questions that the taxonomic similarity measure presented vis-à-vis the frequency based similarity measure during the conversation.

It will be interesting to see how PreDiCtS will continue to develop. Meanwhile we hope that the work presented in this paper provides an initial step towards the adoption of such mixed-initiative processes in the personalisation of the discovery and composition of Web services.

7. REFERENCES

[1] Aamodt, A., Gu, M., Tong, X., Component retrieval using conversational case-based reasoning. Proceedings of the ICIIP 2004, International Conference on Intelligent Information Systems. Beijing, China, October 21 - 23, 2004

[2] Aamodt, A., Gu, M., A Knowledge-Intensive Method for Conversational CBR, Proc. ICCBR'05, Chicago, August 2005

[3] Abela, C., Montebello, M., PreDiCtS: A Personalised Service Discovery and Composition Framework, in proceedings of the Semantic Web Personalisation Workshop, SWP 06, Budva Montenegro, 11th-14th June 2006

[4] Abela, C., Montebello, M., CCBR Ontology for Reusable Service Templates, in proceedings of the Demos and Posters session of the 3rd European Semantic Web Conference (ESWC 2006), Budva, Montenegro, 11th - 14th June, 2006

[5] Abela, C., Personalised Service Discovery and Composition Based on Conversational Case-Based Reasoning, MSc thesis, Department of Computer Science and AI, University of Malta, September 2006.

[6] Aha, D.W., Breslow, L.A., Muñoz-Avila, H., Conversational case-based reasoning, Applied Intelligence, 14, 9-32. (2001).

[7] Bernstein, A., Kaufmann, E., Kiefer, C., Bürki, C., SimPack: A Generic Java Library for Similarity Measures in Ontologies, University of Zurich, Department of Informatics, August 2005

[8] Buckle, M., Abela, C., Montebello, M., A BPEL Engine and Editor for the .NET framework, , accepted at the ECOWS 2005 conference, Växjö Sweden, November 2005

[9] CCBROnto, http://www.semantech.org/ontologies/CCBR-Onto.owl

[10] Creek, http://creek.idi.ntnu.no/

[11] Deelman, E., Gil, Y., et al, Mapping Abstract Complex Workflows onto Grid Environments**,** Journal of Grid Computing, Vol. 1, No. 1, pp 9--23, 2003

[12] eBay, http://www.ebay.com

[13] Goderis, A., et al, Seven bottlenecks to workflow reuse and repurposing, 4th Int. Semantic Web Conf., Galway, Ireland, 6-10 Nov. 2005

[14] Gupta, K., Taxonomic Conversational Case-Based Reasoning, Proceedings of the 4th International Conference on Case-Based Reasoning, 2001

[15] Hefke, M., A Framework for the successful Introduction of KM using CBR and the Semantic Web Technologies, I-Know 2004

[16] IUCBRF, Indiana University Case-Based Reasoning Framework http://www.cs.indiana.edu/~sbog-aert/CBR/

[17] Kim, J., Gil, Y., Towards Interactive Composition of Semantic Web Services, In AAAI Spring Symposium on Semantic Web Services, Palo Alto, California, USA, 2004

[18] Leake, D., Aktas M.S., Pierce M., Fox, G.C., A Web based Conversational Case-Based Recommender System for Ontology aided Metadata Discovery. Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04),Pages: 69 - 75

[19] OWL-S, http://www.daml.org/services/owl-s/1.1/

[20] Peer, J., A POP-based Replanning Agent for Automatic Web Service Composition, Second European Semantic Web Conference (ESWC'05), 2005

[21] Rajasekaran, P., et al, Enhancing Web services description and discovery to facilitate composition, First International Workshop, SWSWPC, July 2004

[22] Scicluna, J., Abela, C., Montebello, M., Visual Modelling of OWL-S Services, IADIS International Conference WWW/Internet, Madrid Spain, October 2004

[23] ServoGrid project, http://www.servogrid.org/

[24] Sirin, E., Parsia, B., et al, Filtering and selecting semantic web services with interactive composition techniques, IEEE Intelligent Systems, 19(4): 42-49,2004

[25] Sirin, E., et al, Planning for web service composition using SHOP2, Journal of Web Semantics, 1(4):377-396, 2004

[26] Weber, B., Integration of Workflow Management and Case-Based Reasoning, Supporting Business Processes through an Adaptive Workflow Management System, PhD thesis, University of Innsbruck, 2003

[27] Weber, B., et al, CCBR-Driven Business Process Evolution, Proc. 6th Int. Conf. on Case-Based Reasoning (ICCBR'05), Chicago, August 2005