Integrating Case-Based Reasoning with Adaptive Process Management

Jakob Pinggera[1], Stefan Zugal[1], Barbara Weber[1], Werner Wild[2], and Manfred Reichert[3,4]

[1]Quality Engineering Research Group, University of Innsbruck, Austria
[2]Evolution Consulting, Austria
[3]Inst. Databases and Information Systems, Ulm University, Germany
[4]Information Systems Group, University of Twente, The Netherlands

Zusammenfassung The need for more flexiblity of process-aware information systems (PAIS) has been discussed for several years and different approaches for adaptive process management have emerged. Only few of them provide support for both changes of individual process instances and the propagation of process type changes to a collection of related process instances. The knowledge about changes has not yet been exploited by any of these systems. To overcome this practical limitation, PAIS must capture the whole process life cycle and all kinds of changes in an integrated way. They must allow users to deviate from the predefined process in exceptional situations, and assist them in retrieving and reusing knowledge about previously performed changes. In this paper we present a proof-of concept implementation of a learning adaptive PAIS. The prototype combines the ADEPT2 framework for dynamic process changes with concepts and methods provided by case-based reasoning (CBR) technology.

1 Introduction

Process-aware information systems (PAIS) allow coordinating the execution of business processes by providing the right tasks to the right people at the right time. In order to support a broad spectrum of business processes, PAIS must be flexible at run-time. Ad-hoc deviations from the predefined process schema as well as the quick adaptation of the process schema itself due to changes of the underlying business processes must be supported [7, 13].
When exceptional situations are handled in a too rigid manner, users are forced to bypass the system, resulting in limited traceability and information loss. Some PAIS recognize the need for more flexible systems and support the definition of ad-hoc modifications for a single process instance, as well as changes of the underlying business processes [7, 11, 8]. However, when the same or similar exceptions occur more than once, a new ad-hoc modification has to be defined each time. In addition, the knowledge about exceptions is not kept in the system and thus lost. Due to this, memorization and reuse of exceptional knowledge is highly desirable and should be supported by the PAIS. When similar exceptions occur

frequently they should no longer be dealt with in an ad-hoc way, but should be covered by updating the process model itself. Frequent similar exceptions indicate that the process model does not adequately reflect the real world business process. The process engineer should then be supported by the PAIS to utilize the collected knowledge for improving the process model at hand [14, 12]. To tackle this problem the conversational case-based reasoning (CCBR) Tool has been developed.
ADEPT2 [10, 9, 8] as a PAIS allows ad-hoc changes as well as changes at the process type level. In addition, ADEPT2 supports the migration of running instances to a new schema version. CCBR Tool enhances this functionality with the ability to trace information about the modifications. In the context of ADEPT2, CCBR Tool is responsible for annotating details about the performed changes. This information in turn is used for an intelligent reuse of modifications. Integrating CCBR Tool in ADEPT2 combines the functionalities of both tools, supplementing one another perfectly. This report shows how the integration has been accomplished and gives a guide on using CCBR Tool in the context of ADEPT2.

This report should help to quickly get familiar with the architecture and usage of CCBR Tool. The first part (Section 1 and 2) consists of a user manual, which is intended for users who only want to use CCBR Tool. It contains a user guide describing CCBR Tool's main functionality and explains the usage of CCBR Tool and how to get started with it. The second part (Developer Manual) deals with more technical details of CCBR Tool and is for anyone who wants to extend or integrate CCBR Tool in some 3rd party application. It gives an overview of CCBR Tool's architecture and design (cf. Section 3.2, 3.4), describes how 3rd party applications can communicate with CCBR Tool (cf. Section 3.5, 3.6) and shows how it can be extended (cf. Section 3.7)

2 User Manual

The user manual is for persons who just want to use CCBR Tool. It shows how CCBR Tool can be installed and started quickly. In addition, it describes the major use cases of CCBR Tool.

2.1 Quickstart

On the following pages all tasks, which have to be fulfilled in order to start CCBR Tool for the first time, are described.

Installation of CCBR Tool CCBR Tool is implemented as a bundle of Eclipse [1] plugins, consequently an installation of Eclipse 3.2 is necessary.
To install CCBR Tool, import all plugins (see Table 3.4) in your workspace. If there is no need for a database configuration (as in most cases the default settings work fine), Section 2.1 can be skipped and proceeded with Section 2.1.

Database Connection CCBR Tool uses the Hibernate Framework [2], thus a wide variety of databases is supported. Currently CCBR Tool is tested with two of them. On the one hand with MySQL [5] Server and on the other hand with HSQLDB [3] database. By default the HSQLDB database is used. If you like to switch to another database, see Section 2.1.

Generate Database Schema In order to create the database schema launch the `GenerateSchema` class, which is located the package `org.cbrflow.cbr.server.generate`, as a normal Java Application. By default the schema is build on the HSQLDB database.

Generate Test Data To generate test data simply run the `GenerateCaseBase` class in the `org.cbrflow.cbr.server.generate` package as a normal Java Application.

Configure Database Connection The database connection is configured in the `hibernate.cfg.xml` file, which can be found in the `org.cbrflow.cbr.server.hibernate` package. Currently there is a configuration for the MySQL database and one for the HSQLDB database implemented. By default the MySQL configuration is commented out.
To switch to MySQL, remove the comment tags around the MySQL configuration and comment out the one for HSQLDB. Please ensure that your MySQL database contains a schema called "cbr" and a user "cbr" with password "cbr". For a different user / password, just edit the corresponding tags in the configuration XML (change the property tags with name ="hibernate.connection.username" and "hibernate.connection.password"). As your database is schema is empty, you need to create the tables for persisting data - this is explained in Section 2.1.

Starting CCBR Tool After importing all plugins in your Eclipse workspace, open the plugin `org.cbrflow.cbr.cbrToolRCP` and open the file `cbrtool.product` by doubleclicking. Pressing the button "Launch the product" (left bottom of the editor) will start the application. If you want to use XML-RPC communication, additionally product `xmlRpc.product` needs to be started, since this product represents the remote CCBR Client.

Starting ADEPT TestClient with CCBR Tool Beside CCBR Tool you will need to download following plugins constituting ADEPT:

- `Additional ADEPT Files`
- `ADEPT2Plugin`
- `de.aristaflow.adept2.ui.editor`
- `de.aristaflow.adept2.ui.editor.loggerView`
- `de.aristaflow.adept2.ui.instanceview`
- `de.aristaflow.adept2.processrepositorymanagerplugin`
- `de.aristaflow.adept2.processvisualizationplugin`

- `de.aristaflow.adept2.ui.swtgui`
- `FormEnvironmentPlugin`
- `OpenOfficeIntegration`
- `TestClient`
- `WorklistVisualizationPlugin`
- `org.cbrflow.cbr.adeptintegration`

In order to start ADEPT TestClient with CCBR Tool from within Eclipse several tasks have to be fulfilled.

- *Copy conf folder* The *conf* folder has to be copied from the `TestClient` plugin to the location of your current Eclipse installation (e.g. C:\Programme\Eclipse). This has to be done only once.
  Please note that the *conf* folder may not be visible in the Java perspective of Eclipse. Nevertheless, it can be discovered using the Resource perspective.
- *Create X: drive* In order to use the OpenOffice [6] integration of ADEPT use your command prompt to switch to the *Additional ADEPT files* folder, which is available from CVS, and execute *subst X: .* . This creates a new drive, which has to contain a folder called *Innsbruck*. This has to be done after each restart of the system. OpenOffice should be installed to the default location in order to use it.
- *Edit MS Office path* To use Microsoft Office with ADEPT the path to your Office installation has to be adapted. Check the *Trip.template* file in `ADEPT2Plugin_2.0.0.jar` in the
- `ADEPT2Plugin` plugin. The *Trip.template* file is located in the folder `de/aristaflow/adept2/july`. Replace all occurences of the path to an Office installation with the path of your installation. Be aware of spaces in the path, which could cause trouble.

ADEPT can be started by opening the `TestClient.product` and pressing the launch button.

2.2 CCBR Tool Major Components

Figure 1 shows the major components of CCBR Tool and how it interplays with ADEPT.

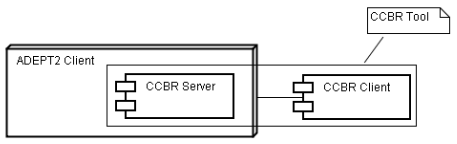


Abbildung 1. CCBR Tool, -Client and -Server

CCBR Client CCBR Client allows for the adding of new cases as well as for the retrieval and reuse of existing ones. As a rich client it does not only consist of a graphical user interface, but also contains business logic.

CCBR Server The server is responsible for data persistency, what makes CCBR Tool independent of the persistency provided by a 3rd party application. Furthermore CCBR Server offers a interface for communication with CCBR Client.

CCBR Tool CCBR Tool denotes the combination of CCBR Client and CCBR Server.

ADEPT ADEPT [8], which is shown on the left hand side, offers the functionalities of a Process Management System, which enables the user to model, execute and monitor processes. Additionally ADEPT provides an exception handling mechanism, which allows the user to perform appropriate actions. This is supported by CCBR Tool, which simplifies the retrieval of similar exceptions and known solution. In order to gain the benefits of CCBR Tool ADEPT invokes the functionalities offered by CCBR Server.

2.3 CCBR Tool Manual

In combination with ADEPT one has the possibility to use two features of CCBR Tool. On the one hand the user can reuse cases by calling *retrieve case* and on the other hand he is able to create a new case. In order to launch CCBR Tool the user has to select an item in the ADEPT Worklist and click afterwards on the CBR button (cf. Fig. 2), which opens the retrieve user interface (cf. Chapter 2.3).

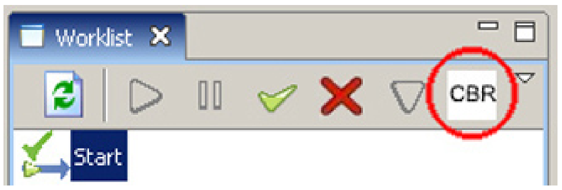
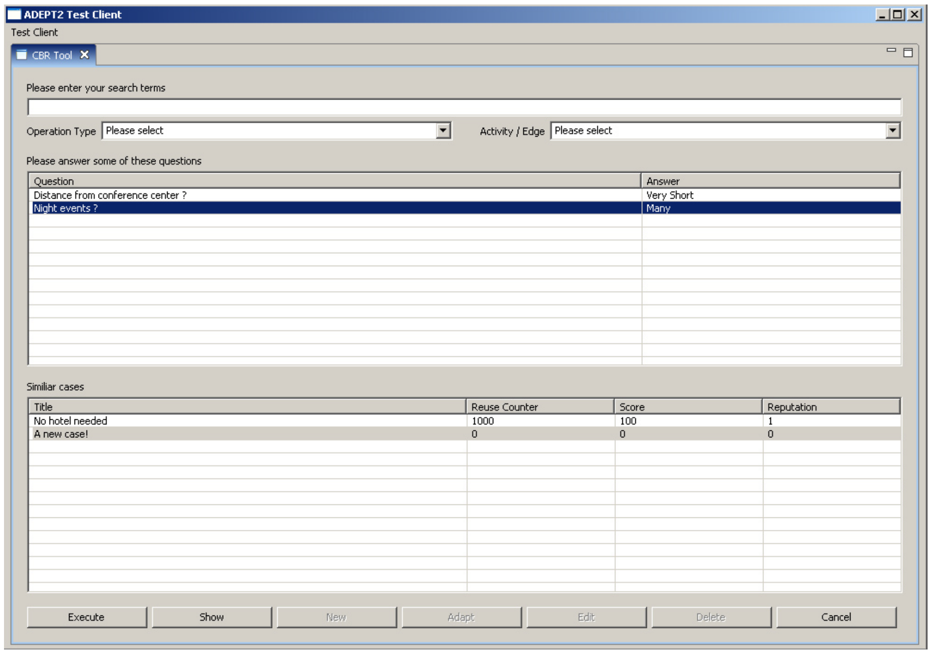


Abbildung 2. The ADEPT worklist

Retrieve Case After starting CCBR Tool the graphical user interface for case retrieval is opened. The table located on the upper half of the screen (cf. Fig. 3) contains all questions of the visible cases and offers the user the possibility to select the possible answers by using the drop down boxes included on the right hand side of the table. By selecting several answers the case list is reordered to show the best matching case on the first position. The ordering is calculated



Please enter your search terms

Operation Type: Please select

Activity / Edge: Please select

Please answer some of these questions

Question	Answer
Distance from conference center ?	Very Short
Night events ?	Many

Similiar cases

Title	Reuse Counter	Score	Reputation
No hotel needed	1000	100	1
A new case!	0	0	0

Abbildung 3. Retrieve a Case

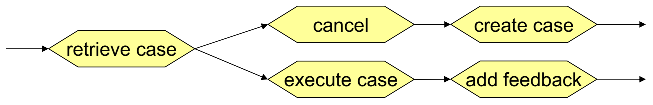


Abbildung 4. Possible executions of CCBR Tool.

by dividing the number of correctly answered questions minus the number of incorrectly answered questions by the total number of questions in the case. As a result one can easily find cases, which accomplish his needs.
For each case in the table the user can take a closer look by clicking the Show button (cf. Chapter 2.3).
In retrieval mode the user has two options (cf. Fig. 4):

– *Execute Case* If one has found a similar case, he can select the respective case and press the *Execute* button on the bottom left side. As a consequence ADEPT receives the change operations, which are defined in the case, and invokes them. If the change operation invocation was successful, the user has the possibility to add a feedback in the dialog shown in Figure 9. On the top of the form one can select a feedback grade which indicates the quality of the case and in the box below additional comments can be entered, which can be useful for later viewers.

– *Cancel* Alternatively, if one does not find a case satisfying his needs, he can abort the interaction with CCBR Tool by pressing the *Cancel* button, which offers the user the possibility to restart CCBR Tool in the *case creation* mode to construct a new case according to his wishes (cf. Chapter 2.3).

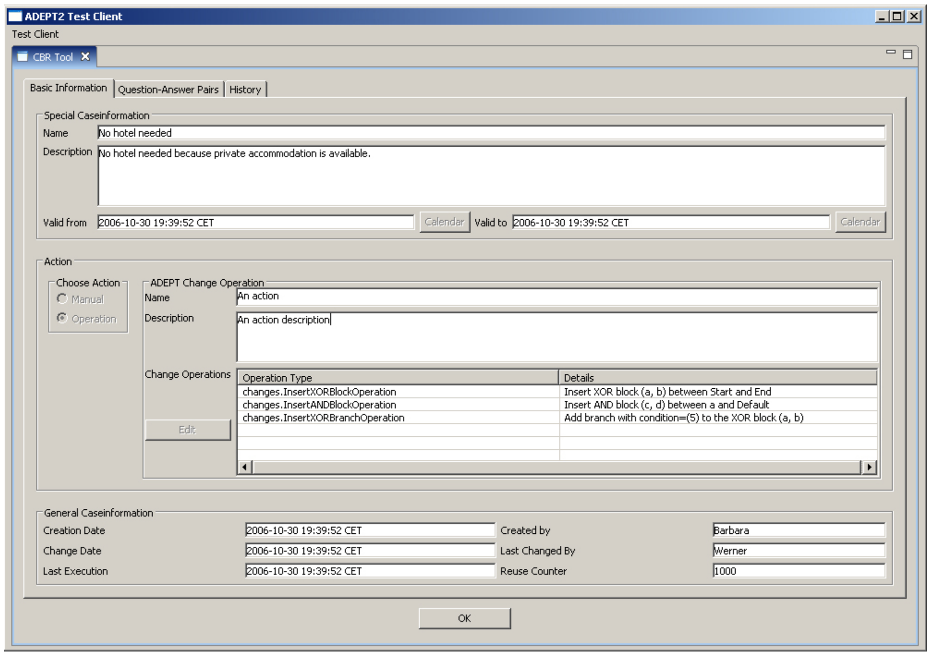
Show Case After selecting one element in the case list, the user can take a closer look to the case by pressing the show button at the bottom of the window. This switches to another, three tabbed, user interface.

– *Basic Information* The basic information tab (cf. Fig. 5) offers the user information like the case name, its description and about the action associated to the case. Additionally facts like the creation date or the date of the last execution are announced. Note that there is no way to edit this data.
– *Question Answer Pairs* On the second tab one can find information about the question and answer pairs associated with the case (no screenshot). All pairs are shown in the table in the middle of the window.
– *History* The third tab (cf. Fig, 6) is responsible for displaying given feedbacks. Basically there are two different feedback types. On the one hand evaluation feedbacks, which are created by a user and contain a feedback grade and a comment, and on the other hand auto generated feedbacks, which are written, for example, on case creation. Again, it is not possible to add feedbacks in the show mode, which we are currently in.

By pressing the OK button at the bottom of the window the user returns to the already known retrieve user interface (cf. Fig. 3).

New Case After aborting the case retrieval process by pressing the cancel button or invoking the close functionality of the CCBR Tool Tab the user is asked whether he wants to create a new case (cf. Fig. 7).
On the first sight the user interface looks identical to the one used in the *Show Case* mode (cf. Chapter 2.3). This is true in *Case Creation* mode except the fact that the user is now able to edit all fields but the ADEPT change operations, which are not entered by CCBR Tool, but via the graphical editor of ADEPT and then passed to CCBR Tool at its start.
The second difference is the button bar at the bottom of the window, which offers the user the following options (cf. Fig. 8).

– *Save* Saves the newly created case to the database. In order to save a case, all necessary information must have been entered. Incorrect input or missing data results in an error message.
– *Save and Execute* Saves the case as described above and executes it immediately (cf. Chapter 2.3).
– *Cancel* Cancels the interaction with CCBR Tool and shows the ADEPT user interface without any changes on the business process.



Special Caseinformation

Name	No hotel needed
Description	No hotel needed because private accommodation is available.
Valid from	2006-10-30 19:39:52 CET
Valid to	2006-10-30 19:39:52 CET

Action

Choose Action: Manual [✓ Operation]

ADEPT Change Operation

Name	An action
Description	An action description

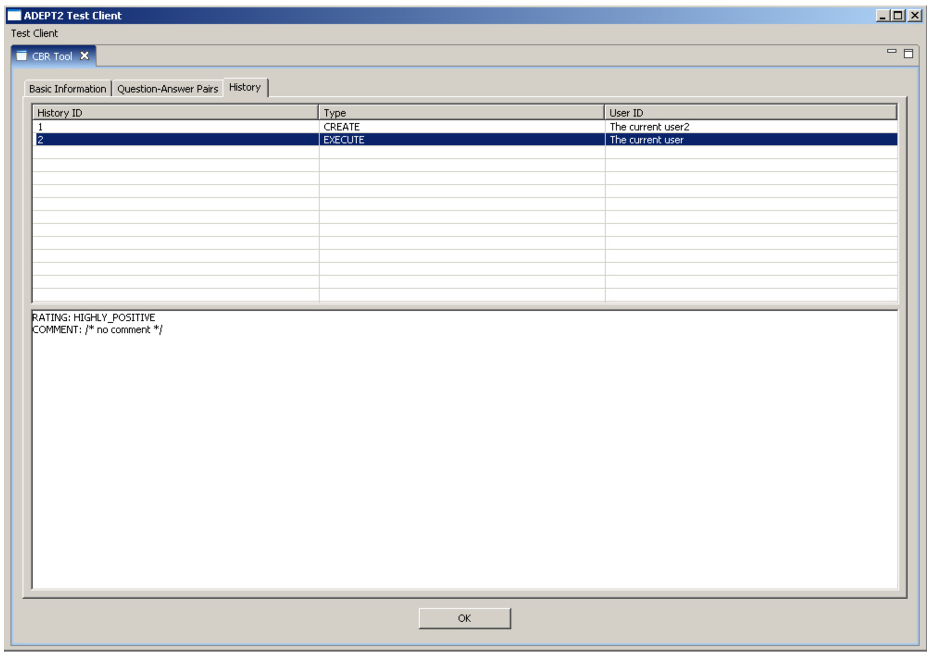
Change Operations

Operation Type	Details
changes.InsertXORBlockOperation	Insert XOR block (a, b) between Start and End
changes.InsertANDBlockOperation	Insert AND block (c, d) between a and Default
changes.InsertXORBranchOperation	Add branch with condition=(5) to the XOR block (a, b)

General Caseinformation

Creation Date	2006-10-30 19:39:52 CET	Created by	Barbara
Change Date	2006-10-30 19:39:52 CET	Last Changed By	Werner
Last Execution	2006-10-30 19:39:52 CET	Reuse Counter	1000

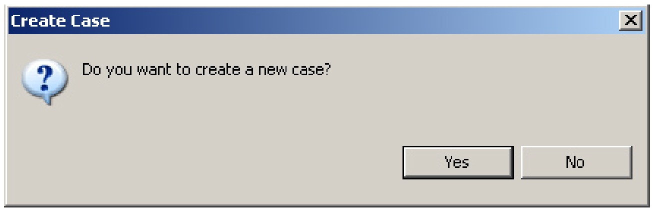
Abbildung 5. Basic Information for the case



History ID	Type	User ID
1	CREATE	The current user2
2	EXECUTE	The current user

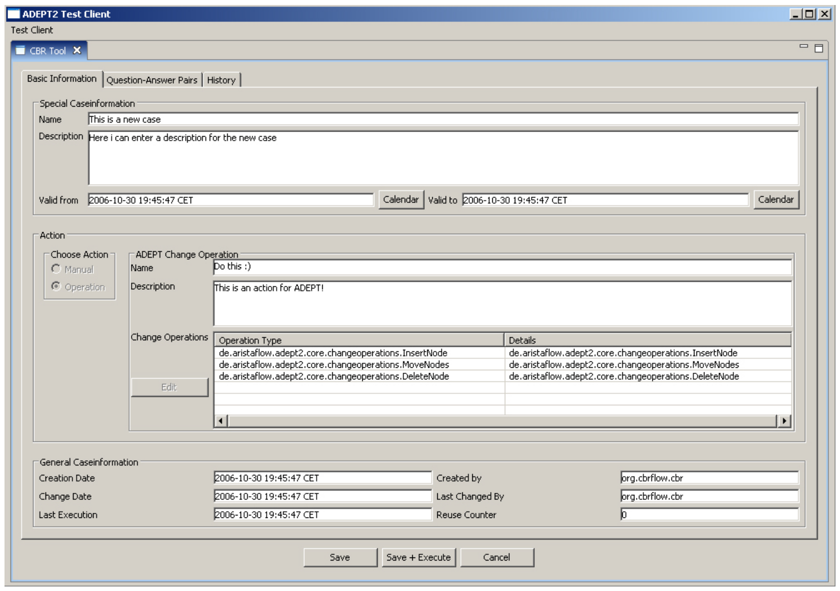
```
RATING: HIGHLY_POSITIVE
COMMENT: /* no comment */
```

Abbildung 6. Feedbacks for the case


Create Case

Do you want to create a new case?

Abbildung 7. Start CBR Tool in *case creation* mode


ADEPT2 Test Client

Test Client

Special Caseinformation

Name: This is a new case

Description: Here i can enter a description for the new case

Valid from: 2006-10-30 19:45:47 CET Valid to: 2006-10-30 19:45:47 CET

Action

Choose Action: Manual [✓ Operation]

ADEPT Change Operation

Name: Do this :)

Description: This is an action for ADEPT!

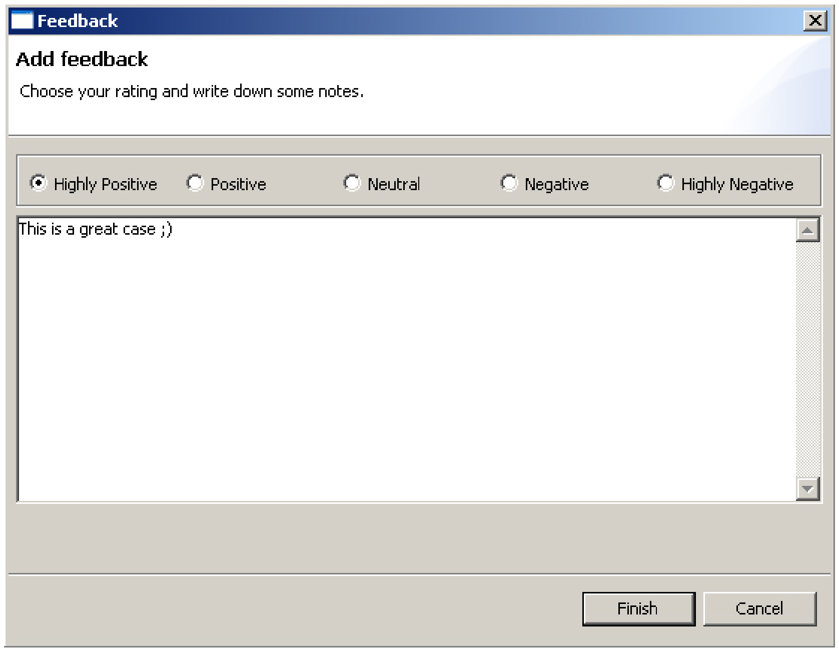
Change Operations

Operation Type	Details
de.aristaflow.adept2.core.changeoperations.InsertNode	de.aristaflow.adept2.core.changeoperations.InsertNode
de.aristaflow.adept2.core.changeoperations.MoveNodes	de.aristaflow.adept2.core.changeoperations.MoveNodes
de.aristaflow.adept2.core.changeoperations.DeleteNode	de.aristaflow.adept2.core.changeoperations.DeleteNode

General Caseinformation

Creation Date	2006-10-30 19:45:47 CET	Created by	org.cbrflow.cbr
Change Date	2006-10-30 19:45:47 CET	Last Changed By	org.cbrflow.cbr
Last Execution	2006-10-30 19:45:47 CET	Reuse Counter	0

Abbildung 8. Create a new case



Choose your rating and write down some notes.

[✓ Highly Positive] Positive Neutral Negative Highly Negative

This is a great case ;)

Abbildung 9. Add Feedback

3 Developer Manual

The developer manual provides more detailed information about the implementation of CCBR Tool. It is intended for readers who want to integrate CCBR Tool in some 3rd party application or extend its functionality.

3.1 Naming conventions

Some frequently used names became quite long in order to follow the plugin naming conventions of Eclipse [1]. To make life easier for the reader, some abbreviations have been introduced.
The prefix of a package is always the same as the name of the plugin where it can be found. The `org.cbrflow.core` plugin contains for example the package `org.cbrflow.core.coreObjects.actions`. Although CCBR Tool consists of quite a number of plugins, one should easily find the desired classes. Plugins comprising CCBR Tool are generally prefixed with "org.cbrflow.cbr". For brevity's sake the prefix is omitted, if it is clear, which plugin is meant.

3.2 CCBR Tool Structure

Figure 10 gives an overview of the structure of CCBR Tool and its integration into ADEPT2.
ADEPT2 consists among others of the `WorklistVisualizationPlugin` plugin, which is used to invoke CCBR Tool. This is done by calling the offered functionalities of CCBR Server, which is seamlessly integrated into ADEPT. For more detailed information about the communication between ADEPT and CCBR Server take a look at Chapter 3.5.
On the right hand side CCBR Tool is shown, which contains CCBR Server and CCBR Client. Note that there is no direct connection between ADEPT and CCBR Client. The communication between the CCBR Server and the CCBR Client is described in Chapter 3.6.
At the bottom of Figure 10 all included plugins are listed and assigned to the specific parts of CCBR Tool. Detailed descriptions of all plugins can be obtained from Section 3.4 on page 14.

3.3 Three-tier architecture

CCBR Tool implements the three tier architecture (cf. Fig. 11). This section deals with each layer in more detail and discusses design decisions.

Presentation Tier The presentation tier is responsible for presenting data in a way the user can convienently work with. According to the three tier architecture model (cf. Fig. 11), the presentation layer is not allowed to know any business functionality. In addition, direct communication with the data tier is

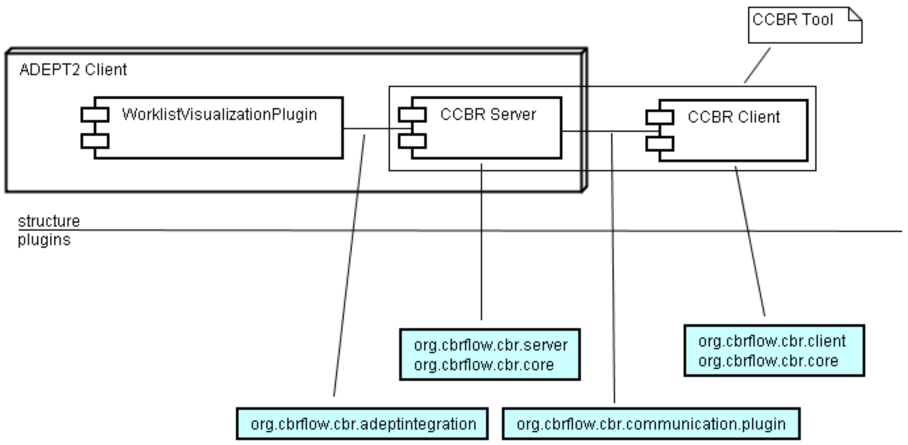


Abbildung 10. Structure of CCBR Tool

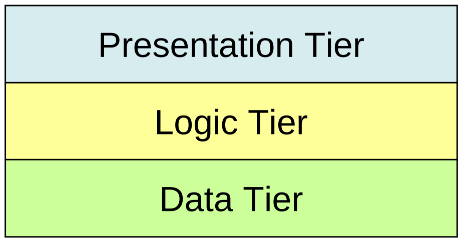


Abbildung 11. Three-tier architecture

prohibited.

CCBR Tool deploys the technology of SWT, in particular it makes use of the JFace framework [1]. Besides eclipse offers user interface components, like Views and Perspectives, which were also frequently used during the development of CCBR Tool.

Logic Tier The underlying logic tier is responsible for coordinating the behavior of the application as well as the evaluation of data.

Core Model The main entry point to the core model is `CaseBase` which is used to store a various amount of `Cases`. Each `CaseBase` has a unique schema id, which indicates to which process schema the `CaseBase` is attached.
Every `Case` in the `CaseBase` consists of three parts.

– QAPair A `Case` contains a set of observations, which are simply a mapping between `Questions` and `Answers`. These `Question` and `Answer` pairs

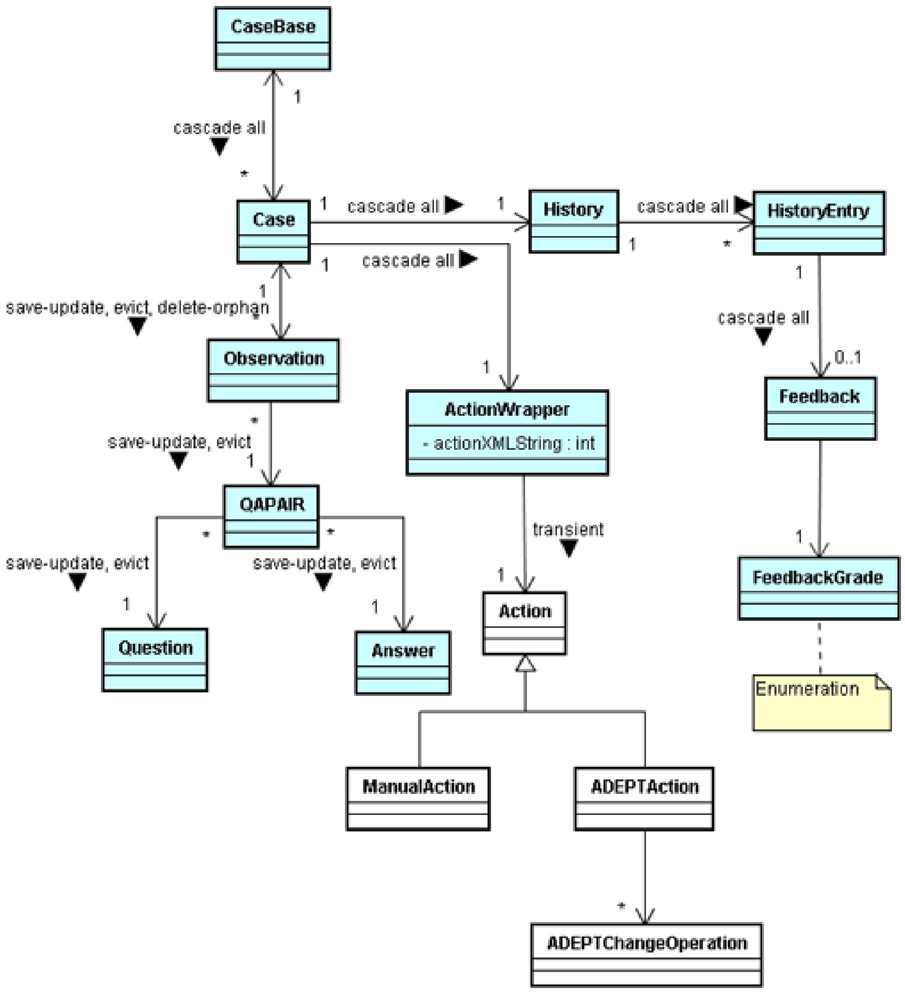


Abbildung 12. CCBR Core Model

(`QAPair`) are used during case retrieval to find `Case`s which are relevant for the user in his actual situation.

– Action On the other hand the `Case` class contains an `Action`, at the moment there are two `Action` types implemented. On the one hand manual `Action`, which simply contains a textual description and a name, and on the other hand ADEPT2 `Action`, which are used by ADEPT2 to store the so called `ChangeOperation`s.
– History Additionally we implemented a `Case` history for each case, which can be used to evaluate `Case`s and which contains entries for facts like `Case` creation. Evaluation entries additionally contain a `FeedbackGrade`, which allows the user to rate the `Case`.

When implementing the logic tier it has been of paramount importance to develop objects, which do not depend on the database or their graphical representation. By obeying this design guideline, core classes can be used in the `client` or `server` plugin as well as for the XML communication (cf. Section 3.6). Furthermore this allows the core classes to focus on their business rules they must implement, in turn leading to code which is not "contaminated" with GUI or database specific code. Additionally the communication between CCBR Client and Server belongs to the logic tier. Section 3.6 discusses this topic in detail.

Data Tier All functionality regarded to data persistency can be found in the data tier. CCBR Tool deploys databases for storing information. Since quite a number of different databases are available and each one has its own peculiarities, Hibernate [2] has been used to introduce a uniform storing mechanism.

Hibernate simplifies the support of a multiplicity of databases (CCBR Tool actually supports MySQL [5] and HSQLDB [3]), since supporting a new database merely requires adding the corresponding jdbc driver and changing the connection URL. In addition, hibernate works as a O/R (object to relational) mapping framework. In order to save a business class in a relational database, XML files must be defined. These mapping files specify, which fields of the class need to be saved or what is the primary key of the class. The time which is lost due to definition of mapping files is nevertheless well invested because in turn Hibernate manages tedious work like caching of objects and cascading.

3.4 Plugins

This Section describes all plugins used in CCBR Tool and how they fit to the three-tier application model.

Plugin architecture As described in Section 3.3, CCBR Tool is designed to fit the three-tier application model. According to this architecture, the plugins can be grouped in the following way:

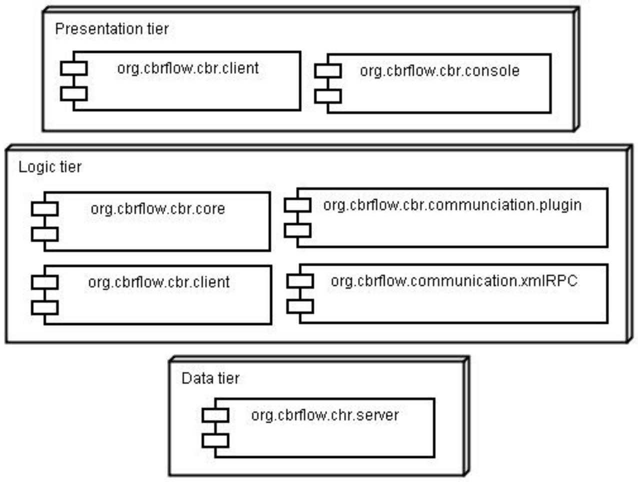


Abbildung 13. Three - tier architecture

- Presentation Tier The `client` and `console` plugins are associated with the presentation tier.
- Logic Tier The `core` plugin belongs to this layer, representing the object model of CCBR Tool. Parts of the `core` plugin have been extracted to the plugin `defaultSimilarityCalculator` to allow a more flexible calculation of the case similarity. Furthermore plugins `communication.plugin` and `communication.xmlRpc` realize the communication between CCBR Server and Client, thus also belonging to the logic tier. Finally the `client` plugin should be mentioned, since it also contains application logic.
- Data Tier The `server` plugins realizes persistency demands.

Figure 13 depicts this grouping. Plugins `serverView`, `adeptintegration` and `cbrToolRCP` do not belong to any group - they can be considered as supplementary plugins: `serverView` represents a 3rd party application, `adeptintegration` contains all classes needed for integrating CCBR Tool in ADEPT and `cbrToolRCP` depicts CCBR Tool, already integrated in a host application (the `serverView` plugin). Consequently these plugins do not constitute CCBR Tool, they merely serve demonstration and integration purposes.

The current implementation of CCBR Tool consists of several plugins. Basically there are two groups of plugins - the ones which were implemented by the developers of CCBR Tool and the assisting plugins (like Hibernate [2] or JDOM [4]). The latter ones are merely extracted .jar files, which were transformed to plugins to better fit the plugin model of Eclipse [1].
Table 3.4 contains a complete list of all plugins constituting CCBR Tool.

Name of plugin	impl.	assis
org.cbrflow.cbr.client	x	
org.cbrflow.cbr.core	x	
org.cbrflow.cbr.core.defaultSimilarityCalculator	x	
org.cbrflow.cbr.communication.plugin	x	
org.cbrflow.cbr.communication.xmlRpc	x	
org.cbrflow.cbr.console	x	
org.cbrflow.cbr.server	x	
org.cbrflow.cbr.serverView	x	
org.cbrflow.cbr.adeptintegration	x	
org.cbrflow.cbr.cbrToolRCP	x	
com.mysql.jdbc		x
org.apache.commons.logging		x
org.apache.xmlRPC		x
org.hibernate.eclipse		x
org.hsqldb		x
org.jdom		x
org.log4j		x
org.vafada.swtcalendar		x

Tabelle 1. All plugins

org.cbrflow.cbr.client The graphical user interface of CCBR Tool is placed in this plugin. If you want to extend CCBR Tool with your own extensions (cf. Section 3.7), this plugin is the right place. Of course, if your new functionality require persistency support, the `server` plugin must be adapted too.

org.cbrflow.cbr.core The business classes of CCBR Tool are separated into this plugin. They represent the notion of cases, casebases, questions and more. For detailed information see Chapter 3.3.

org.cbrflow.cbr.server The `server` represents a reference implementation of CCBR Server, which is integrated into ADEPT2 (or any other 3rd party application which wants to use CBR functionality). The server is responsible for persisting data as well as the communication with ADEPT and CCBR Client.

org.cbrflow.communication.plugin This plugin is used for the communication between CCBR Server and CCBR Client assuming that both parts run within the same Java Virtual Machine. It provides a well defined interface for data exchange and triggering of CCBR Tool's functionality (retrieving cases, creating a case, . . .). More information about the communication architecture can be found in Section 3.6.

org.cbrflow.communication.xmlRpc Plugin `communication.xmlRpc` is used for the communication between CCBR Server and CCBR Client assuming that both parts do not run within the same Java Virtual Machine. The plugin demonstrates the usage of XML RPC for data exchange. If you are interested to implement your own communication protocol, read Section 3.6.

org.cbrflow.cbr.console The `org.cbrflow.cbr.console` plugin provides a console view to display additional debugging information during the execution of CCBR Tool.

org.cbrflow.server.serverView The `serverView` plugin adds a graphical user interface to the server, which makes it possible to test CCBR Client with the provided server, simulating a 3rd party application.

org.cbrflow.cbr.adeptintegration The `adeptintegration` plugin offers an interface for the interaction between ADEPT2 and the server plugin, in order to enable ADEPT2 to invoke the functionalities of CCBR Tool. Basically the incoming requests are simply forwarded to the server plugin. Additionally the `adeptintegration` plugin saves information, which cannot be processed by the CCBR Tool (actually the process template id). This information is returned afterwards to ADEPT2 with CCBR Tool's reply. For further information about the communication with the server take a look at Chapter 3.5.

org.cbrflow.cbrToolRCP All plugins of CBR Tool are bundled in `cbrTool-RCP`: The `client` constituting the user interface, plugins `core`, `communicati-on.plugin` and `communication.xmlRPC` comprising the logical tier and plugin `server` for data persistency. Additionally plugin `serverView` is added to simulate a 3rd party application, facilitating to run CBR Tool without ADEPT.

3.5 Communication with CCBR Server

In order to communicate with CCBR Server the offered methods by the `Server-Controller` have to be invoked and the `IServerControllerCallback` interface implemented to treat the callbacks from the server plugin. The methods defined in this interface are invoked on case execution, case creation, case adaption, after editing a case or after deleting a case. Additionally there is a callback if the user cancels the interaction with CCBR Tool.

Retrieve in Detail The sequence diagram in Figure 14 shows the execution of a case launched by ADEPT2 (actually launched from the `CallCCBRToolAction` class).

Call CCBR Tool First ADEPT2 has to create a `RetrieveCCBRCallback` which methods are invoked after execution of the CCBR Tool. This callback has to be passed together with the `templateId`, which will be called also schemaId in further processing, and the `instanceId` to the `AdeptServerCommunicator`. This

class is part of a plugin, called `org.cbrflow.cbr.adeptintegration`, which works as an adapter between ADEPT2 and the CCBR Server. It is used to save the `instanceId` for returning it to ADEPT2 after execution of CCBR Tool. This is necessary, because the `instanceId` is not saved on the server and the CCBR Client is not able to receive and return the `instanceId` in order to pass it to ADEPT2 after invocation.
The ADEPT2 integration plugin creates now an instance of `ServerControllerCallback` and passes it together with the schemaId and `instanceId` to the CCBR server, which invokes the CCBR Client. Afterwards the server executes `actionPerformed` on the given callback. Note that there is only a schemaId, the caseId and a XML string containing the action returned. The `ServerControllerCallback` now adds the `instanceId` and passes it together with the obtained results from the CCBR Server to the `RetrieveCCBRCallback`, which was created by ADEPT2 at the beginning of the invocation process.

Execute Change Operations The above described `RetrieveCCBRCallback` calls `executeChangeOperations` in the `CallCCBRToolAction`, which receives the `instanceId` and a list containing `ChangeOperationXMLWrappers`. At this point ADEPT2 executes the given change operations, and returns true, if successful, to the `RetrieveCCBRCallback`, which triggers the increase of the reuse counter. This is done by invoking `increaseReuseCounter(long schemaId, long caseId)` in the `AdeptServerCommunicator`, which simply forwards the request to the `ServerController`. Note that there is no callback for increasing the reuse counter, thus ADEPT2 does not know if the counter was really increased.

Add Feedback After executing a case the user has the possibility to add a feedback to the case. If the user decides to do so, the `addFeedbackToCase` method in the `CallCCBRToolAction` is invoked, which needs two parameters, the `schemaId` and the `caseId`. The feedback request is forwarded to the `AdeptServerCommunicator` by executing `addFeedback` with the same parameters as given before and additionally the username for the current user. The `AdeptServerCommunicator` opens now a feedback dialog, where the user could enter a comment and a grade for the case, which indicates its quality. Afterwards a history entry is created and forwarded, together with the `schemaId` and the `caseId`, to the `ServerController`, which saves the history entry to database. Note that there is no callback to ADEPT for adding a feedback.

New Case in Detail If the user is in retrieval mode and cancels the interaction with CCBR Tool by clicking the Cancel button in CCBR Client, one has the possibility to create a new case (cf. Fig. 15).

Call CCBR Tool Until the user presses the Cancel button, the execution stack is identical to the one described in Chapter 3.5. The CCBR server invokes instead of the `actionPerformed` method the `canceled(long schemaId)` method. The adept integration plugin, in fact the `ServerControllerCallback` adds the

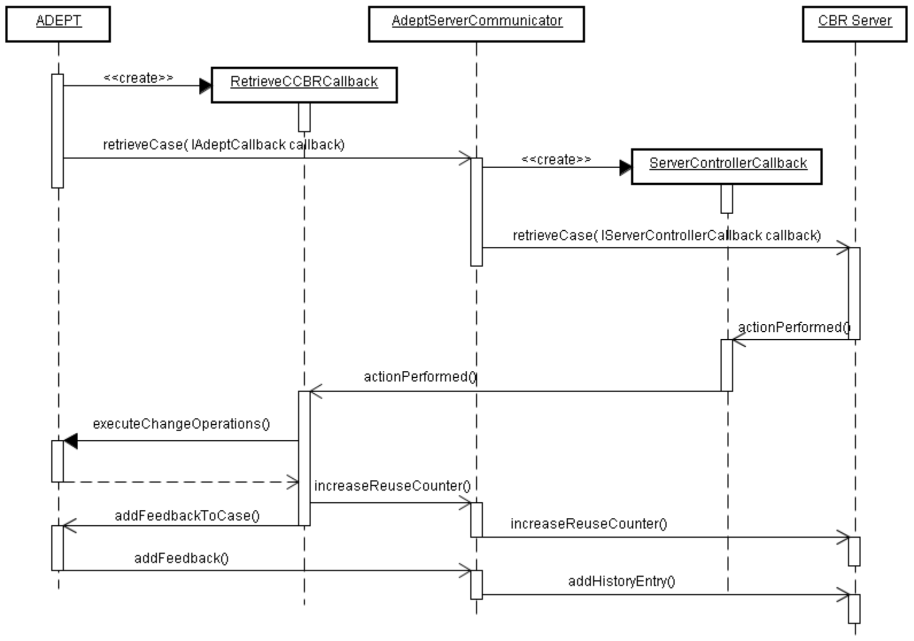


Abbildung 14. Retrieve Case from ADEPT

`instanceId` to the result and triggers the canceled method in the `RetrieveCC-BRCallback`, which asks the user if he likes to create a new case. If desired the CCBR Tool is restarted in *Case Creation* mode.

Case Creation Mode The `RetrieveCCBRCallback` invokes the `newCase(long processTypeTemplateId, long instanceId)` method in the `CallCCBRAction`. Note that the `processTypeTemplateId` equals the `schemaId` returned by the CCBR server.
The `CallCCBRToolAction` creates an instance of `NewCaseCCBRCallback` and passes it together with the given `processTypeTemplateId` and the `instanceId` to the `AdeptServerCommunicator`. Additionally ADEPT2 has to add change operations to the request, which have to be encoded as a JDOM Element in the following format.

```
<ACTION_TO_PERFORM ID="0" TYPE="ADEPTAction">
  <ACTION_NAME />
  <ACTION_DESCRIPTION />
  <CHANGEOPERATIONS>
    <CHANGEOPERATIONS type="InsertXORBlockOperation"
    description="Insert XOR block (a, b)
    between Start and End" />
    <CHANGEOPERATIONS type="InsertANDBlockOperation"
    description="Insert AND block (c, d)
    between a and Default" />
  </CHANGEOPERATIONS>
</ACTION_TO_PERFORM>
```

The action name and description will be entered by the user in CCBR Client and the id of the action will be changed when the action is saved to the CCBR database. At this point of development change operations are hard coded in the `CallCCBRToolAction`.

The `AdeptServerCommunicator` creates, like in the retrieve request, an instance of `ServerControllerCallback` and passes it with the `schemaId`, the `instanceId` and the change operations, contained in the XML described above, to the `ServerController`. The server invokes CCBR Client, where the user enters all mandatory data about the new case. CCBR Client sends this created case back to the server, which triggers `caseCreated(long schemaId, long caseId)` in the `ServerControllerCallback`. The callback executes the operation `caseCreated` in the `NewCaseCCBRCallback` with the given parameters and the saved `instanceId`. The callback informs ADEPT2, by calling `caseSaved(long templateId, long instanceId, long caseId)`.
Please note that `schemaId` is called `templateId` in this context.
Additionally it is possible that the user likes to execute the newly created case, which leads to an additional `performAction` callback and the possibility to add a feedback to the case (cf. Section 3.5).

3.6 Communication with CCBR Client

Basically the communication with CCBR Client is defined by a XML document, described in Chapter 3.6, which has to be passed when invoking one of its functionalities.

Plugin Layer In order to support several communication technologies like XML-RPC [15] we decided to introduce a layer of plugins between the server and CCBR Client. This layer contains a plugin for each supported technology. By default the communication over XMLRPC and the direct communication within the same Eclipse [1] runtime is implemented.

Request to CCBR Client Figure 16 shows the communication of CCBR Server with CCBR Client in case of a retrieve request. The `ServerController` creates a new instance of `ServerControllerCallback`, which is passed together with the request, described in chapter 3.6, as arguments to the `PluginServerCommunicator`. These classes are all located in the `org.cbrflow.cbr.server` plugin. The `PluginServerCommunicator` creates a `CBRListener` instance and forwards it together with the XML request to the `Communicator` class, which is located in the `org.cbrflow.cbr.communication.plugin` plugin, by calling `retrieve`. This plugin is one member of the above mentioned layer between CCBR Server and CCBR Client. In this case the server and CCBR Client run in

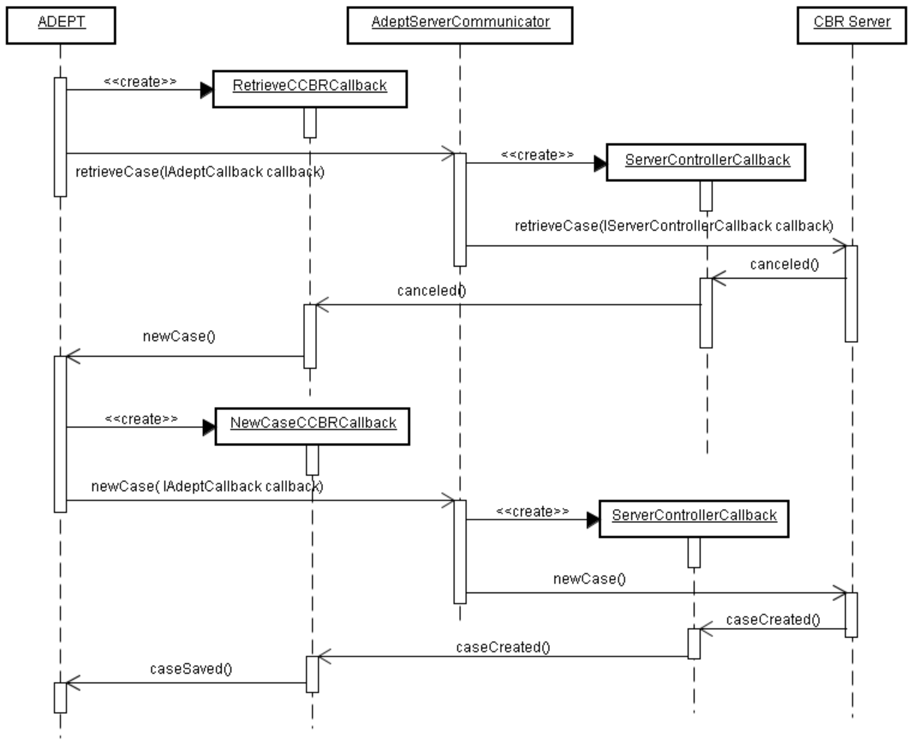


Abbildung 15. New Case from ADEPT

the same Eclipse [1] runtime environment. As a result the plugin simply forwards the request to the `CBRCommunicator`, which is part of CCBR Client plugin. CC-BR Client opens the graphical user interface, where the user can select a case, which fits his requirements. For a more detailed description of the user interface take a look at Section 2.3.

Response from CCBR Client When the user has found a appropriate case he presses the execute button, which invokes the `response(String xmlResponse)` method of the `CBRListener`, which contains the formerly created instance of the `ServerControllerCallback`. This callback will be later used to inform users of CCBR Server about the actions, like described in Chapter 3.5.
This callback is passed together with the XML response to the `ServerController` by calling `processResponse(String xmlResponse, IServerControllerCallback callback)`. The commands encoded in the XML reply are perfomed by the `ServerController`, which invokes the appropriate methods in the `ServerControllerCallback`.
Note that there is no fixed interface for calling the communication plugins. Everybody is free to implement an own protocol in order to communicate with

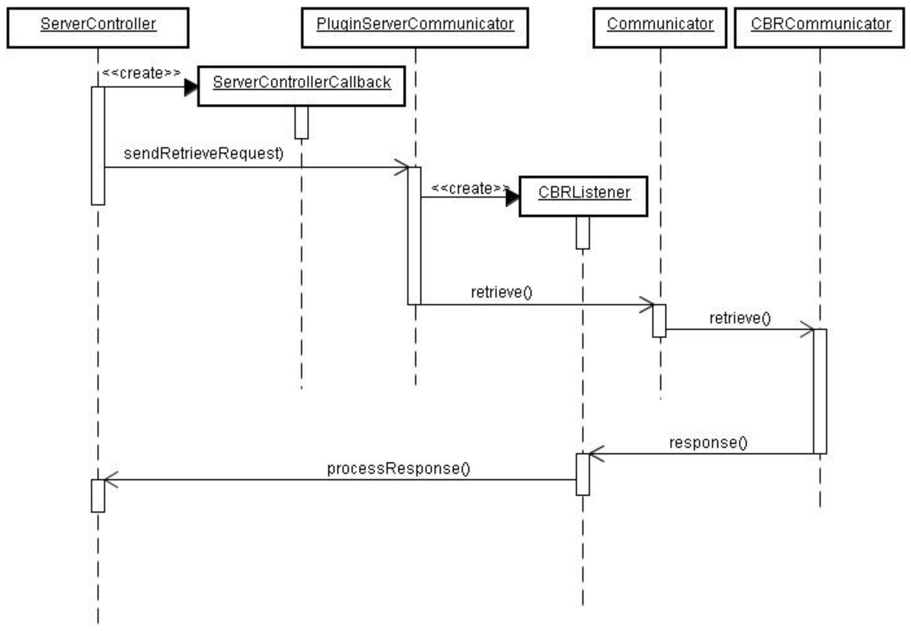


Abbildung 16. Communication with CCBR Client - Sequence Chart

CCBR Client. Of course it is possible to communicate with classes from CCBR Client directly. Although it is recommended to use the `communication.plugin` plugin as intermediate layer, because it isolates changes in the `client` plugin communication classes. An example is shown in Figure 17.

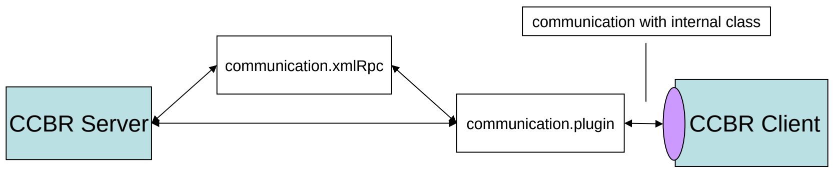


Abbildung 17. Communication with CCBR Client - Plugin Structure

XML Schema This section describes the XML schema, which are used to communicate between the server and CCBR Client. There exist basically two types of schema. On the one hand for sending a request to CCBR Client, and on the other hand the response, which is returned by the CCBR Client.

XML Request The XML request, which is send to CCBR Client, is surrounded by a `REQUEST` element as root. This item contains exactly one casebase, which has a name, a description and a locale. Besides every `CASEBASE` element consists of an `ID` and `SCHEMA_ID` attribute and a `CASES` element which contains an arbitrary number of cases. The request shown in the following example contains only one case in order to retrieve a compact representation.

```
<?xml version="1.0" encoding="UTF-8"?>
<REQUEST>
<CASEBASE ID="1" SCHEMA_ID="123">
<LOCALE>de_AT</LOCALE>
<DESCRIPTION>The basic casebase.</DESCRIPTION>
<NAME>Casebase basic</NAME>
  <CASES>
    <CASE ID="1">
          <NAME>No hotel needed</NAME>
          <DESCRIPTION>No hotel needed because private
           accommodation is available.</DESCRIPTION>
          <CREATEDDATE>2006-11-06 20:19:18 UTC</CREATEDDATE>
          <CREATEDUSER>Chuck</CREATEDUSER>
          <MODIFIEDUSER>John</MODIFIEDUSER>
          <MODIFIEDDATE>2006-11-06 20:19:18 UTC</MODIFIEDDATE>
          <VALIDFROM>2006-11-06 20:19:18 UTC</VALIDFROM>
          <VALIDTO>2007-11-06 20:19:18 UTC</VALIDTO>
          <EXECDATE>2006-11-06 20:19:18 UTC</EXECDATE>
          <EXECCOUNTER>1000</EXECCOUNTER>
          <RATING>0</RATING>
          <QAPAIRS>
           <QAPAIR ID="1">
                 <QUESTION ID="1">Night events ?</QUESTION>
                 <ANSWER ID="1">Many</ANSWER>
                 <TYPE>CAN</TYPE>
           </QAPAIR>
                 <QAPAIR ID="2">
                 <QUESTION ID="2">Distance from conference center?
                 </QUESTION>
                 <ANSWER ID="2">Very Short</ANSWER>
                 <TYPE>CAN</TYPE>
           </QAPAIR>
          </QAPAIRS>
          <ACTION_TO_PERFORM ID="0" TYPE="ADEPTAction">
           <ACTION_NAME>Book no hotel</ACTION_NAME>
           <ACTION_DESCRIPTION>Book no hotel, because it
                  is not necessary.</ACTION_DESCRIPTION>
                 <CHANGEOPERATIONS>
                   <InsertXORBlockOperation
```

```
                     type="InsertXORBlockOperation"
                         description="Insert XOR block (a, b)
                         between Start and End"/>
                   <InsertANDBlockOperation
                        type="InsertANDBlockOperation"
                        description="Insert AND block (c, d)
                        between a and Default"/>
                   <InsertXORBranchOperation
                        type="InsertXORBranchOperation"
                        description="Add branch with condition=(5)
                        to the XOR block (a, b)"/>
                 </CHANGEOPERATIONS>
          </ACTION_TO_PERFORM>
          <HISTORY>
          <HISTORY_ENTRY ID="2" TYPE="EXECUTE">
                <TIMESTAMP>2006-11-07 16:21:18 UTC</TIMESTAMP>
                <USER>John</USER>
                <FEEDBACK>
                 <GRADE>HIGHLY_POSITIVE</GRADE>
                 <COMMENT>/* no comment */</COMMENT>
                </FEEDBACK>
                 </HISTORY_ENTRY>
                <HISTORY_ENTRY ID="1" TYPE="CREATE">
                 <TIMESTAMP>2006-11-06 20:19:18 UTC</TIMESTAMP>
                 <USER>Chuck</USER>
                </HISTORY_ENTRY>
          </HISTORY>
        </CASE>
   </CASES>
</CASEBASE>
</REQUEST>
```

The most important part of the request is the `CASE` element, which has similar to the `CASEBASE` item an `ID` attribute. Additionally every case contains several data like name, description, the last execution date or the execution counter. For a full list of elements of the casebase take a look at the above shown XML. Additionally a Case consists of the following parts:

– QAPAIRS This element consists of all question and answer pairs used by the case. A question and answer pair contains of a question, an answer, both with a unique id and an `ID` attribute for the question and answer pair itself. The `QAPAIRS` item can contain an arbitrary number of pairs.
– ACTION_TO_PERFORM On the other hand the case contains a `ACTION_TO_PERFORM` element, which includes the action of the case. There are two attributes for an action, the `ID` and the `TYPE`. In connection with ADEPT2 [8] only `ADEPTAction`s are used. Every action consists of an `ACTION_DESCRIPTION`,

an `ACTION_NAME` and the ADEPT2 change operations, which are located in the `CHANGEOPERATIONS` item. This element can contain an arbitrary number of change operations, which contain of a type and a description attribute.
– HISTORY The `HISTORY` element consists of `HISTORY_ENTRY` items, which are defined by a unique `ID` and a `TYPE` attribute. Additionally a timestamp and the user, who created the history entry, either by creating or evaluating the case, is part of the entry. In case of an evaluation entry the user has the possibility to add a feedback, which is saved in a `FEEDBACK` item and consists of a `GRADE` and a `COMMENT` element.

XML Response The response by CCBR Client is designed quite similar to the request, except that the `REQUEST` item is replaced by a `RESPONSE` element as root, which contains the casebase, like described in request part of Chapter 3.6. Note that in most circumstances only the case to handle (save or execute) is sent back with the response. Only if the user canceled the interaction with CCBR Client the whole casebase is returned. The following example shows such a response.

```
<?xml version="1.0" encoding="UTF-8"?>
<RESPONSE>
<CASEBASE ID="1" SCHEMA_ID="123">
  <LOCALE>de_at</LOCALE>
  <DESCRIPTION>The basic casebase.</DESCRIPTION>
  <NAME>Casebase Basic</NAME>
  <CASES>
    <CASE ID="0">
      <NAME>Cancel travelling.</NAME>
      <DESCRIPTION>Skip the booking and the payment, because
      nothing appropriate was found.</DESCRIPTION>
      <CREATEDDATE>2006-11-06 20:49:52 UTC</CREATEDDATE>
      <CREATEDUSER>Chuck</CREATEDUSER>
      <MODIFIEDUSER>Chuck</MODIFIEDUSER>
      <MODIFIEDDATE>2006-11-06 20:49:52 UTC</MODIFIEDDATE>
      <VALIDFROM>2006-11-06 20:49:52 UTC</VALIDFROM>
      <VALIDTO>2006-11-06 20:49:52 UTC</VALIDTO>
      <EXECDATE>2006-11-06 20:49:52 UTC</EXECDATE>
      <EXECCOUNTER>0</EXECCOUNTER>
      <RATING>0</RATING>
      <QAPAIRS>
        <QAPAIR ID="0">
          <QUESTION ID="0">Anything found?</QUESTION>
          <ANSWER ID="0">No</ANSWER>
          <TYPE>CAN</TYPE>
        </QAPAIR>
      </QAPAIRS>
      <ACTION_TO_PERFORM ID="0" TYPE="ADEPTAction">
        <ACTION_NAME>Skip booking and payment.</ACTION_NAME>
        <ACTION_DESCRIPTION>Skip the booking and payment
        nodes.</ACTION_DESCRIPTION>
        <CHANGEOPERATIONS>
          <InsertANDBlockOperation
          type="InsertANDBlockOperation"
          description="Insert AND block (c, d) between
          a and Default" />
          <InsertXORBlockOperation
          type="InsertXORBlockOperation"
          description="Insert XOR block (a, b) between
          Start and End" />
          <InsertXORBranchOperation
          type="InsertXORBranchOperation"
          description="Add branch with condition=(5)
          to the XOR block (a, b)" />
        </CHANGEOPERATIONS>
      </ACTION_TO_PERFORM>
      <HISTORY />
    </CASE>
  </CASES>
</CASEBASE>
<SERVER_COMMANDS>
  <BY_USER USERNAME="Chuck">
    <NEW_CASE ID="0" />
    <PERFORM_ACTION ID="0" />
  </BY_USER>
</SERVER_COMMANDS>
</RESPONSE>
```

Additionally to the casebase a `SERVER_COMMANDS` item can be found in the response, which indicates what the server has to do, and who wants the server to perform this commands. In the above shown example the user, who triggers the server commands, is called Chuck. This information can be obtained by the `USERNAME` attribute in the `BY_USER` item. Basically it is possible, that several users trigger commands in one response, but in fact CCBR Client supports only one user per response. The `BY_USER` item includes an arbitrary number of server commands. In connection with ADEPT2 [8] only `NEW_CASE` and `PERFORM_ACTION` are used, like shown in the above example. Every server command has an `ID` attribute which indicates on which case the action should be performed.

3.7 Extensions for CCBR Tool

CCBR Tool provides several extension mechanisms, which enable other plugins to react on events during the execution of CCBR Tool.

XML Converter Extension On the one hand CCBR Client offers an extension mechanism, which makes it possible for additional plugins to interfere with the XML request handling and the creation of the XML reply. This technique allows to augment the XML with additional data, which can be used by plugins extending CCBR Client.
In order to write a new XML Converter the `XMLConverter` extension point offered by the `client` plugin has to be implemented. This extension point contains the `IXMLConverter` interface, which is located in the package `org.cbrflow.cbr.client.extension`. This interface defines two methods. On the one hand `fromXML(Element root)`, which is called whenever a XML request is send to CCBR Client. The additional plugin should extract needed information from the request in this method. On the other hand the `toXML(Response response)` method is defined, which is the counterpart to the fromXML operation. CCBR Client calls it on every registered plugin before sending the reply back to the server. The plugin has now the possibility to add information to the response.

EventHandler Extension The second offered extension point of CCBR Client is the EventHandler extension point, which informs registered plugins whenever an event occurs. At the moment only the execution of cases is supported.
Plugins which like to implement this extension have to register at the `org.cbrflow.cbr.client.eventHandlers` extension point. This contains the `IExecuteHandler` interface which defines the operation `handleExecute(Case caseToExecute)`. This method is called by CCBR Client whenever a case is executed in order to enable specific reactions of the registered plugins.

Similarity Calculation Extension The Similarity extension point can be used to customize the similarity calculation of CCBR Client. For this purpose one has to register at the `org.cbrflow.cbr.core.Similarity` extension point, which enforces the programmer to implement the `ISimilarityCalculator` interface in the `org.cbrflow.cbr.core.extension` package. This interface contains a single method, which is called for every case when the similarity score is calculated. Additionally a set containing all `QAPair`s, which are currently selected by the user, is passed to the method, which has to calculate the score for the given case and return it as a double value.
It is important that there is exactly one plugin registered at this extension point at runtime. Otherwise the similarity score can not be calculated and an `IllegalSimilarityCalculatorAmountException` is thrown.

3.8 Additional Database Information

This section contains additional information about database decisions we made, which can be useful for setting up CCBR Tool on another database.

– Because of the fact that Date objects are saved differently on a MySQL database and HSQLDB (HSQLDB saves millis whereas MySQL does not)

we decided to save all Date objects as Strings in the database. This should not be a problem on any database.
– At the moment QAPairs, Questions and Answers remain in the database, when the last parent element in the core hierarchy is deleted. For instance if there are no more Observations in the database with a specific QAPair, the QAPair still exists in the database. For more information about the core see Section 3.3.

Literatur

1. Eclipse. http://www.eclipse.org, 2006.
2. Hibernate. http://www.hibernate.org, 2006.
3. HSQLDB. http://www.hsqldb.org, 2006.
4. JDOM. http://www.jdom.org/, 2006.
5. MySQL. http://www.mysql.com, 2006.
6. OpenOffice. http://www.openoffice.org, 2006.
7. M. Reichert and P. Dadam. ADEPT$_{flex}$ - Supporting dynamic changes of workflows without losing control. *Journal of Intelligent Inf. Systems*, 10(2):93–129, 1998.
8. M. Reichert, P. Dadam, M. Juritsch, U. Kreher, K. Göser, and M.Lauer. Architectural design of flexible process management technology. Technical Report 2008-02, Univ. of Ulm, 2008.
9. M. Reichert, S. Rinderle, U. Kreher, and P. Dadam. Adaptive process management with ADEPT2. In *Proc. ICDE'05*, pages 1113–1114, 2005.
10. Manfred Reichert and Peter Dadam. ADEPTflex-supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
11. S. Rinderle, M. Reichert, and P. Dadam. Flexible support of team processes by adaptive workflow systems. *Distributed and Parallel Databases*, 16(1):91–116, 2004.
12. S. Rinderle, B. Weber, M. Reichert, and W. Wild. Integrating process learning and process evolution - a semantics based approach. In *Proc. BPM'05*, pages 252–267, 2005.
13. B. Weber, S. Rinderle, and M. Reichert. Change patterns and change support features in process-aware information systems. In *Proc. CAiSE'07*, pages 574–588, 2007.
14. B. Weber, S. Rinderle, W. Wild, and M. Reichert. CCBR–driven business process evolution. In *ICCBR'05*, Chicago, 2005.
15. XMLRPC. http://ws.apache.org/xmlrpc, 2006.