Message Management

Task

In all levels of a software application there are messages and error messages. These are active and unrequested sent out messages of the software application to a user or a technical component, which will perform a reaction upon receiving a message. They have different content and different semantics and can occur in all levels of the application. In CBA messages and error messages are grouped together under messages and are treated uniformly.

Error handling, logging and monitoring present special reactions to messages, whereby the same message can trigger mutliple reactions. While normal messages are normally are shown to the inactive user on the userinterface, logging information is saved into a logging database to be displayed to a adminstrator on request. A monitoring system evaluates a specific section of the messages to get a permanent overview of the systems's work. Messages can also present questions to a user or inititate a workflow.

In CBA a message carries besides the actual content a timestamp and often additional information like session, transaction, user etc. This way e.g. invoking capability or invoking duration among others can be derived from messages with targeted use. The absence of messages can also be evaluated.

The CBA message management is client and mulitlanguageable and can generate parameterised messages. These can also be besides errors every kind of message and text blocks. This can be e.g. requests for userinterfaces, formatted messages, text blocks for emails or protocol entries.

Requirements

CBA implements a uniform message management, which also considers requirements for error messages, logging and monitoring:

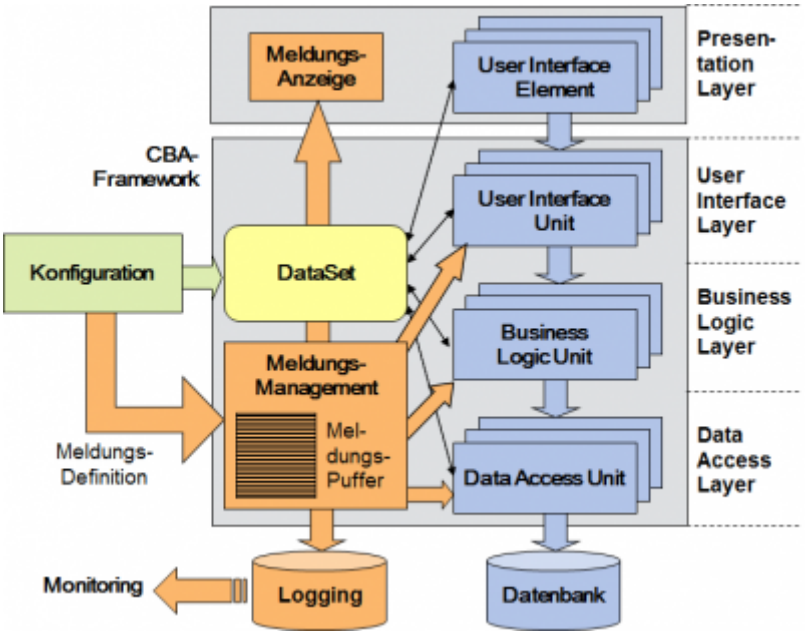
- Messages are classified under their meaning.
- Messages explcitly define together with system settings the necessary reactions. The reaction is also parameterizable.
- When sending a message (e.g. an error) the executed function can optionally be cancelled via a CbaException or all messages can be collected within a logical transaction bracket (e.g. within a service call) and sent as one. This way a error tolerant operation is possible.
- The messages can have one or more parameters, which are handed over via functions, e.g. the filename of a not found file.
- The message management generates a readable message text from the parameterised message.
- Besides errors other kinds of messages and text blocks can be managed like e.g. Requests for Userinterfaces or formatted messages.
- Messages of underlying program sysmtems like e.g. database systems or file systems are being transferred into the uniform message management.
- The CBA message management is mulitlingual, that means the messages will be generated in the user's language.

- In distributed applications messages are transparent, that means they are communicated between service call and service, so that a uniform behaviour independent from the place ot the message creation is realized.

Architecture

The message management is embedded into the CBA-Framework. It consists of

- the message definition
- the temporary message buffer
- the message functions
- prefabricated elements for message display
- the logging database
- the possibilty of linking a monitoring system



Configuration

The potential messages are configurated with a message number. In the message definition placholders can be defined for the paramterers that are to be included later. The message texts are mutlilingual.

Functions

The CBA-Framework offers a set of funtcions for message management:

- **clearMessage** Deleting the message buffer
- **setMessage** Inserting a message into the message buffer

- **getMessageText** Conversion of a message into a text
- **getMessages** Returning the message buffer
- **saveMessage** Saving a message in the logging database
- **removeMessage** Deleting the most recently set message from the message buffer

The fucntion **setMessage** will be handed over a message number, possibly message parameters (e.g. inforamtion about the faulty object or similar things), the classification and a flag for behaviour specification. More parameters can specify the message more strongly and are particularly important for error analysis.

With the help of the function **getMessagetext** and the messsage number a character string is determined, which is formatted with the given message parameters. {0} is a placeholder for the first message parameter, {1} for the second and so on. The message parameters are a string, in which an abitrary number of parameters can be given and are must be seperated by â€š|â€™. The message automatically gets a timestamp, the session assignment and (if available) the transaction number, so that messages can be put into context.

example:

- **message number**: 1234
- **message text**: 'The file '{0}' caused the following error: {1}
- **call**: setMessage (1234, “pic.gif|file read-only”, “E”, “e6”, myTable, null, 4711);
- **message**: The file 'pic.gif' caused the following error: file read-only

The function **removeMessage** can be used to delete the most recently set message from the message buffer. This way errors can be intercepted program-technically and the effects of a message can be reverted.

The fucntion **getMessage** can also be called without writing a message into the messafe buffer. E.g. multilingual requests, message and similar things can therefore be determined in this userinterface.

getMessages reads the message buffer to evaluate all messages of the transaction.

With **saveMessage** the relevant messages for the purpose of logging and monitoring are saved into the database. The logging includes messages, where the flag 'l' is set and where their level is less than or equal to the logging level, which is set in the system. They can also be read from the database with the help of system tools. The **Monitoring** takes place with an application, which evaluates the in the databased save logging information with its own logic. With the help of the throught the BLU configurated message buffers the monitoring applictaion can be informed about the availabilty of new logging information, so that it can react promptly.

Message Classification

Messages underlie the following classification in CBA:

F .. fatal error	errors, that prevent the application's further work or significantly impair it
E .. error	errors, that can happen in an application's work, only have local effect and which the user can react to
W .. warnings	system's indications on inconsistencies, unfavourable conditions and similar things

I .. information	information to the user
Q .. questions	questions to the user, which can be answered with Yes or No

The classification is handed over when calling the function **setMessage**.

The behaviour of a **setMessage** call is determined by the flag. It can include the following values:

e .. Exception	The message triggers a CbaException, which can be intercepted in any call hierarchy level, after its handling.
1 .. Logging-Level 1	logging and monitoring information for fatal system errrors
2 .. Logging-Level 2	logging and monitoring information for fatal configuration errors
3 .. Logging-Level 3	logging and monitoring information for fatal application errors
4 .. Logging-Level 4	logging and monitoring information for system errors
5 .. Logging-Level 5	logging and monitoring information for configuration errors
6 .. Logging-Level 6	logging and monitoring information for application errors
7 .. Logging-Level 7	application-specific logging and monitoring information
8 .. Logging-Level 8	application-specific logging and monitoring information
9 .. Logging-Level 9	application-specific logging and monitoring information

The messages are being forwarded to the logging and monitoring, when the given level is less than or equal to the level given when calling **saveMessage**.

Behaviour

The message buffer is automatically being cleared at transaction start by calling **clearMessage**. The transaction start is implicitly given via a service call, but can also be given explicitly bythe user calling the function **BlTransactionBeginn**. All messages of the transaction are being collected in the message buffer until the next transaction starts. This way with the help of **getMessages** the list of accumulated messages from the last transaction can be returned after the transaction ended.

Through the targeted control of the exceptions picking up messages are possible without cancelling the program sequence. Error tolerant services can be implemented, which e.g. edit error free information and reject faulty information. With the help of messages the rejected information and the cause for rejection can clearly be identified.

In distributed applications the CBA-Framework ensures that the messages of the called services are being transferred into the message list of the called application and possibly trigger a exception. This way the behaviour is uniform beyond the domain boundaries.

System Messages

System Messages are generated automatically via the CBA-Framework when the following happens:

- detected errors in the CBA-Framework
- error of underlying systems (e.g. file management, databases, ...)

- start and end of a service call
- instantiation of a unit

All system messages of the CBA-Framework are generated via the setMessage fucntion.

Application

Messages are genereated in the source code at different places. During a transaction all messages are cached in a message buffer, so that a sequence of messages can also be created. An important application example for this are validations. The validation is not cancelled on the first detected error, but is continued. Thus a better ease of use can be achieved, because multiple errors can be cleared in one cycle. Through the targeted control of the exceptions picking up messages are possible without cancelling the program sequence.

Subsequent an example for application of message functions:

```
      string MyFunction (...)
        {
            string result = null;
            try
            {
                // ... desiredfunctionality

                // special error
                if (...) setMessage(1001, [par0]+"|"+[par1], "E", "e", null,
null, 0);
                   // [par0] replaces the placeholder {0} and [par1] the
placeholder {1}
        // Flag  'e' => after error handling a CbaException is called

                // ... more functionality

                // ... return of a language dependent message
                result = getMessageText (1002, ...+"|"+...+"|"+...);
            }
            catch (CbaException se)
            {
                // forwarding of an already handled exception
                throw se;
            }
            catch (Exception e)
            {
                // handling of a new exception
                setMessage(1000, e.Message, "E", "eâ€");
            }
            finally
            {
                // ... final measures
            }
```

```
            return result;
        }
```