Application-defined conversation

Application-defined conversations

In D2000 on system level, there is supported a mechanism that enables to create a conversation between **two** "scriptable" and "addressable" entities of the applications.

In practice, the conversations should be used wherever on the application level, there is some session (or binding) between the instances of ESL scripts or process implemented by D2000 JAPI. In the event that such bindings are made by the conversation, there is possible to resolve this condition where the "other entity" of this session ends.

For example:

A service, which allows clients (other server events, pictures or D2000 JAPI) to be attached to receive the messages, is implemented by server event **E. MAIL_SERVER**. This connection is done by this method:

```
RPC PROCEDURE Attach
....
END Attach
```

The client will subscribe to the messages by calling the appropriate message:

```
CALL [E.MAIL_SERVER] Attach ON SELF.EVH
```

E.MAIL_SERVER implements next service that enables to sent the message to logged in clients:

```
RPC PROCEDURE Send(IN TEXT _msg)
...
END Send
```

When ending the client, it must stop registration by calling this method:

```
RPC PROCEDURE Detach
....
END Detach
```

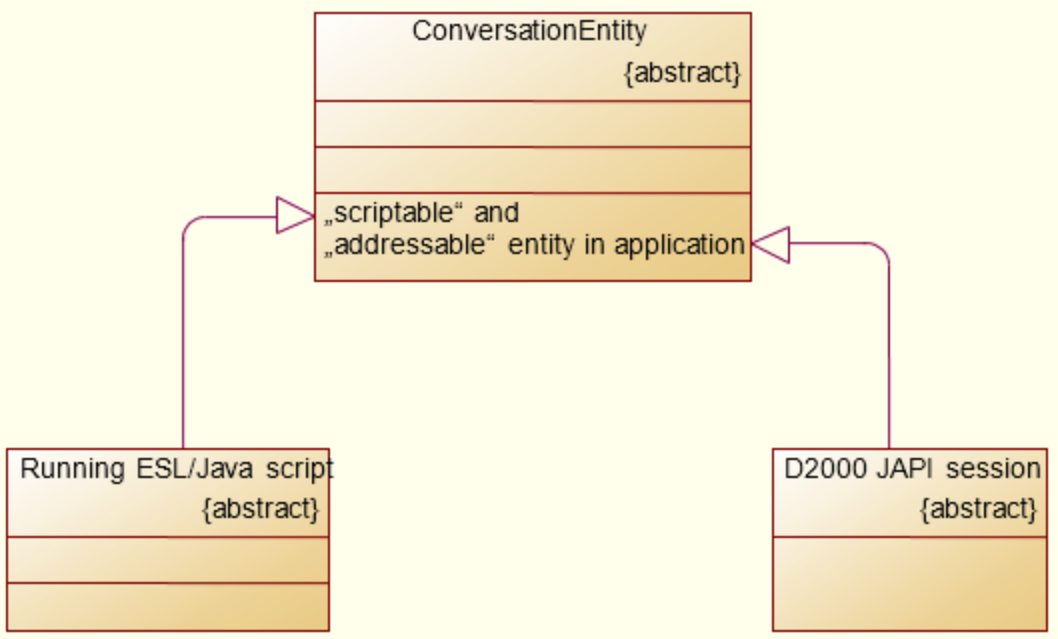
Everything works properly until the moment when a client, who is being ended, do not manage to call the method **Detach**, which causes that **E. MAIL_SERVER** will not be informed that the client no longer exists.
The other error could be that if **E.MAIL_SERVER** was reinitiated for some reason (process restart, new version of service, ...), no client finds out that it must be registered again by calling the method **Attach**.

The conversations help to resolve these problematic situations.

Specification

Below is the list of entities that may be used in the conversations:

- running ESL/Java script (in context of opened object of *Active picture* type or running server event),
- process connected to the application by D2000 JAPI.



The conversation is a unique session between the entities. It is used as a medium (or transport layer) for they mutual **asynchronous** procedural communication (**by calling the RPC procedures and methods of interfaces that they implement**), and regards they life cycle. It means that:

1. Creating of conversation is managed by the application request (calling the RPC procedure). It is created between a requester and entity, which is addressee. Every conversation is in ESL identified by a value of *INT* type (handle of conversation). This value is unique within the instance of event and cannot be shared between other instances of event or picture.
2. Conversation can be closed by any participating entity or automatically by the system providing that any of participating entities were ended (loss of communication due to failure of communication channels, closing the object, saving the object (its configuration) and refresh, ...).

RPC procedures are used to create and close the conversations, and send the messages. Handling of unexpected **interruption** of transaction is done by calling the RPC procedure of script which participates on transaction.

For that purpose, we extended the syntax of declaration of RPC procedures.

```
RPC PROCEDURE [_hTC, TC_B] NewTransaction(parameters)
RPC PROCEDURE [_hTC, TC_E] EndTransaction(parameters)
RPC PROCEDURE [_hTC, TC_BE] QuestionTransaction(parameters)
RPC PROCEDURE [_hTC, TC_C] UserProc(parameters)
RPC PROCEDURE [_hTC, ERROR] TransactionInterrupted
```

Variable *_hTC* is used for identification of conversation (number of conversations is not limited). Within the procedure handling, it is always declared as a local variable of *IN INT* type.

Extended syntax for calling the procedures.

```
 CALL [objIdent] NewConversation(parameters) ASYNC ... TC_B _hTC
CALL [_hTC] EndConversation(parameters) ASYNC ... TC_E
CALL [objIdent] QuestionConversation(parameters) ASYNC ... TC_BE _hTC
CALL [_hTC] UserProc(parameters) ASYNC TC_C
```

Variable *_hTC* is output within the actions **TC_B** and **TC_BE** and input within **TC_C** and **TC_E**. All callings are **asynchronous**.

Key words **TC_B**, **TC_E**, **TC_BE**, **ERROR** are used for procedures that manage the life cycle of conversation (handle or initiate the events that changes its status). Key word **TC_C** is used to indicate the calling that do not change of conversation status.

- **TC_B** – "**T**ransaction **C**onversation - **B**egin"
  Within "**CALL ... TC_B** _hTC", it means the calling of procedure and also creating the conversation. The system expects that the called procedure is declared by the key word **TC_B**. Identifier of new conversation is returned by the value _hTC. It is used for its identification when handling other events (RPC PROCEDURE [_hTC...) or when sending the messages within it (action CALL [_hTC]…). For sending the messages, it is valid until the moment when it is closed for sending (action **CALL ... TC_E**) or it receives the event which interrupts the conversation (**RPC PROCEDURE [_hTC, ERROR]...**).
- **TC_E** – "**T**ransaction **C**onversation - **E**nd"
  Within "**CALL [_hTC] ... T_E**" it means the calling of procedure and also closing the conversation. Conversation must be closed by the appropriate procedure (**RPC PROCEDURE [_hTC, TC_E]…**). Since the closing of transaction by CALL action, _hTC can be used only for the identification of conversation when handling the events to the last event (handle **RPC PROCEDURE [_hTrans, TC_E]…** or **RPC PROCEDURE [_hTC, ERROR]...**).

- **TC_BE** – "**T**ransaction **C**onversation - **B**egin-**E**nd"
  Within "**CALL ... TC_BC** _hTC", it means the calling of procedure and also creating the conversation, which is characterized by a feature that the calling script also closes the conversation and expects only the response(s). Therefore, the value of _hTC cannot be used for changing messages in **CALL** action.
- **ERROR** – Conversation was unexpectedly interrupted – **ERROR**
  The event is generated automatically by the system:
  1. when unexpected crash of conversation (interruption of communication channel, crash of cooperating entity within the conversation),
  2. when calling the non-existent RPC procedure,
     ERROR will be generated for that entity which already knows the conversation.
     When calling:
     - TC_BE only for that entity which calls,
     - TC_C for both,
     - TC_E for both.

     The purpose of such security is that the application server does not support the conversations whose participants do not understand each other.
  3. when the called RPC procedure will be closed by an exception within conversation (ESL RunTimeError). The purpose of such security is that the application server does not support the conversations that have not been implemented correctly.

  Parameter _hTC cannot be used. This procedure may be written in the source code of ESL script only once.

Procedure *UserProc* is a user-defined procedure, which is used to transmit the messages within conversation without change of its status. Name of procedure is always defined by a user. When calling the procedure of ESL Interface, which implements ESL script, a described syntax is applied.

The syntax for calling the procedure which is implemented by a process:

```
CALL [(0)] I.Interface^ProcName(parameters) ON ProcesName.EXT
```

0 – zero

Related pages:

Using conversations
E.MAIL_SERVER example