Persistent State Transaction Concept

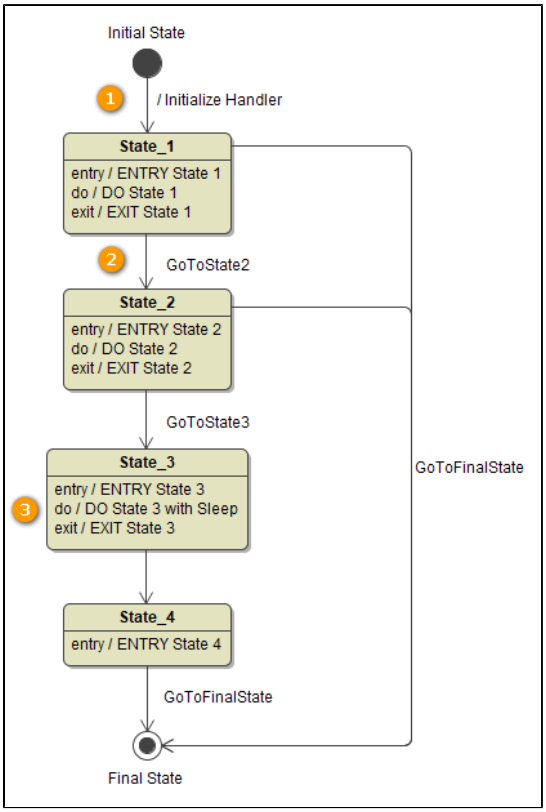
The E2E xUML Runtime works with the concept of **sessions**. Sessions are equivalent to units of work, that can be committed or rolled back depending on the status at the end of the session. Refer to xUML Runtime Transaction Concepts for more details on the transaction concept in general.

On this Page:

Related Pages:

- xUML Runtime Transaction Concepts
- Committing Changes to the Persistent State Database
- Do Activity

Transaction Rollback Behavior

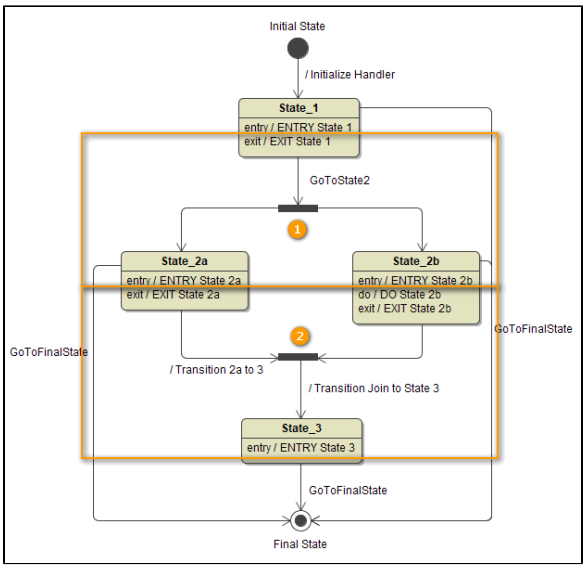


Persistent state transactions will behave on rollback like listed in the tables below:

Simple State Machine

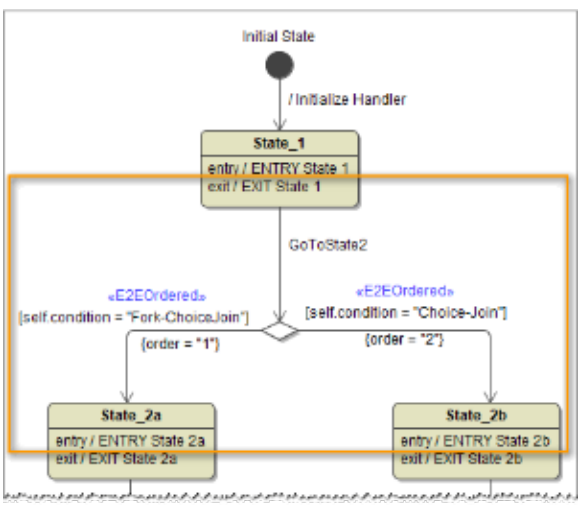
Persistent State Transaction		Error in	Behavior	State
1	**init state - initialize handler - entry action**	initialize handler	object not created	-
		entry action	object not created	-
2	**exit action - transition - entry action**	exit action	rollback of exit behavior changes	source state
		transition	rollback of changes on transition and exit behavior	source state
		entry action	rollback of entry behavior, transition and exit behavior changes	source state
3	**do activity**	do activity	rollback of changes Exit action will wait until do activity is finished.	state machine continues with object lifecycle (independent of error), see also Do Activity Particularities below

State Machine with Fork and Join



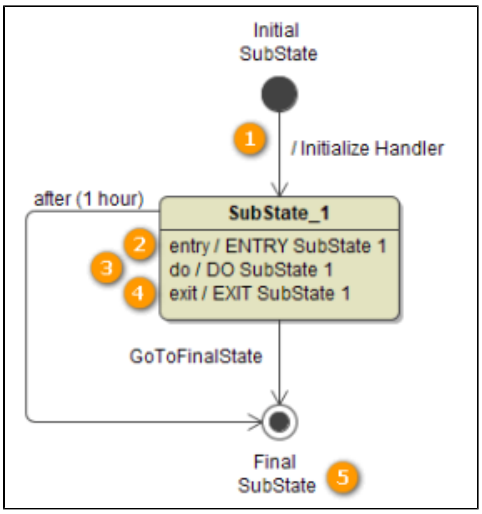
Persistent State Transaction		Error in	Behavior	State
1	**exit behavior + transition + fork + multiple(transition + entry behavior)**	entry action	rollback of all forked transitions, if one fails	state before fork
2	**multiple(exit behavior + transition) + join + entry behavior**	exit action	rollback of exit behavior changes	faulty state and general join state
		transition	rollback of changes on transition and exit behavior of faulty path	faulty state and general join state
		entry action	rollback of entry behavior and transition from join to next state	general join state

State Machine with Choice



Persistent State Transaction	Error in	Behavior	State
multiple(exit behavior + transition) + choice + multiple (transition + entry behavior)	entry action	rollback of changes	source state before choice

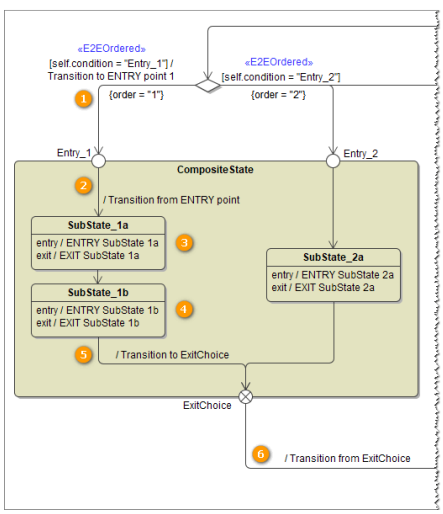
State Machine with Substate Machine

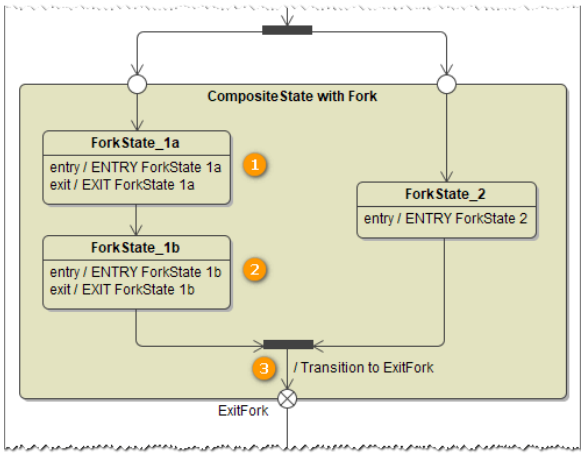


Persistent State Transaction	Error in	Behavior	State
substate machine	1 initialize handler	rollback of changes	source state
	2 entry action	rollback of changes	source state
	3 do activity	rollback of changes Exit action will wait until do activity is finished.	state machine continues with object lifecycle (independent of error), see also Do Activity Particularities below
	4 exit action	rollback of changes	faulty substate and general submachine state
root state machine	5 entry state after substate	rollback of changes of entry action, no rollback of exit substate actions	general submachine state

State Machine with Composite States

Persistent State Transaction	Error in	Behavior	State
composite states (choice + two entry points)	1 transition to entry point	rollback of changes	source state before choice
	2 transition from entry point	rollback of changes	source state before choice
	3 entry action of first substate	rollback of changes	source state before choice
	4 entry action of second substate	rollback of changes	faulty substate and general composite state
	5 transition to exit point	rollback of changes	source state and general composite state
	6 transition from exit point	rollback of changes	general composite state



composite states (fork + two entry points + join within composite)  	1 entry action	rollback of all forked transitions, if one fails	state before fork
	2 entry action within composite	rollback of actions related to faulty branch	• composite state branch 1 • faulty state • composite state branch 2 • join
	3 transition to exit point		• composite state branch 1 • composite state branch 2 • join

Do Activity Particularities

Persistent state do activities are handled by the E2E Runtime different than other persistent state activities - they are processed asynchronously in a separate session and the persistent state object is not locked during execution of the do activity.
This may lead to the following issues:

- **Concurrent Updates on the Persistent State Object**
  While the do activity is still processed by the E2E Runtime, the persistent state object can be changed by other processes. The Runtime will try to merge the changes in this case. If the do activity changes **self.a** and another process triggers a change of **self.b**, the final persistent state object will contain both changes. In case that both change the same attribute, the last change will win.
- **Error in Do Activity (1)**
  If an error occurs during execution of a do activity, the E2E Runtime will throw `[PSADSM][29][Fatal error while executing doActivity` (if no error handler is implemented) and rollback all changes of the do activity. Nevertheless, the persistent state object is not stalled but continues with its lifecycle.
  In this case, it is best practice to implement an error handler that catches the mentioned error and does the necessary handling.
- **Error in Do Activity (2)**
  After a do activity, changes to the self-object are not automatically persisted. To modify contents of the persisted object, the do activity has to return the corresponding attributes as output parameters. In case of error (and rollback), this leads to the fact that the output parameters are `NULL`. As a consequence, the related persistent state attributes are set to `NULL`, too, and not rolled back to the previous value.
  In this case, it is best practice to implement an error handler that catches the above mentioned error and does the necessary handling.

> It is strongly recommended to use error handlers together with do activities.

Apart from this, the do activity is the only activity within an persistent state object's lifecycle the modeler can be sure that the object is **really in** the desired state. So, for example, if you want to inform other parties that a persistent state object is in a specific state, you should always do this in the do activity. Otherwise, you can run into race conditions (e.g. if you use the entry action for this and the other party reacts very fast).

Implicit and Explicit Commits

Changes to persistent state objects are not saved to the persistent state database immediately, but on transaction end. This is called **implicit commit**. If an error occurs during a transaction, transaction changes will be rolled back to the beginning of the transaction (see xUML Runtime Transaction Concepts and Transaction Rollback Behavior).
Additionally, the following action types are part of the transaction and affected on commit / roll back if they were used in persistent state context:

Action Type	Example
database access	insertion or deletion of database records
persistent state handling	creation of a persistent state object, sending of a persistent state signal, sending of conversation signals
JMS activities	sending or receiving of JMS messages with acknowledge mode **transacted**
POP3 activities	deletion of mails from POP3 server

All other actions or adapter calls (e.g. SOAP call, REST call, SAP access, ...) have to be rolled back manually in case of error.

User can force the persistent state machine to commit changes to the database by calling the Persistent State Adapter with action **commit** (see Committing Changes to the Persistent State Database). This is called **explicit commit**. Changes that have been explicitly committed will not been rolled back on transaction error.
Use explicit commits if you do not want certain actions to be rolled back on error, e.g. persistent state signals in case of conversations.

Regarding service robustness, it is not recommended to model too complex transactions, as rollback on error may lead to unwanted results. Better implement intermediate states that can serve as rollback-points in case of error. See also Do Activity for more information on do activities.