Frameworks

Activity Framework

Object Behavior Framework

Persistence Framework

Benutzerhandbuch

V40 2000.10.17 (Oktober 17)

PLEASE NOTE:

This document is being written by Terry Taylor (MYND Deutschland, terrytaylor@mynd.com, +49-173-2845-784) for DEKRA Stuttgart.

RECOMMENDED STARTING POINT: Chapter 5 (Installation) and then Chapter 6 (Tutorial).

The document is not finished and is being updated daily. The primary concern at the current time is content and structure (not the document format).

The document contains english text (written by me) and german text from old documents (not written by me). The English text will eventually be translated to German.

Files (repositories, etc) for the tutorial are stored in a lotus notes database. Contact me to get these files.

Inhaltsverzeichnis

Inhaltsverzeichnis **2**
1. Copyrights **7**
2. Dokumentübersicht **8**
3. Release notes **9**
4. Die Vorteile von Frameworks **10**
5. Installation **11**
5.1. Requirements 11
5.2. Step-by-step 11
5.2.1. Load VA projects 11
5.2.2. Load Mynd DSE 11
5.2.3. Install Frameworks tools 12
5.2.3.1. Install files 12
5.2.3.2. Configure Object Model Browser 12
6. Tutorial **14**
6.1. Overview 15
6.2. Using the supplied code 16
6.3. MVC Design 17
6.3.1. UseCases 17
6.3.2. Activities 17
6.3.2.1. Activities Overview 17
6.3.2.2. Suchen 18
6.3.2.3. KundeBearbeiten 19
6.3.2.4. LkwBearbeiten 20
6.3.3. Persistent Objects 21
6.3.3.1. Objects 21
6.3.3.2. Object accessors (created with OBF) 21
6.3.3.3. DatabaseTables 21
6.3.3.4. RT POM (Object <-> DatabaseTable interface) 22
6.3.4. Views 22
6.3.4.1. Suchen 22
6.3.4.2. KundeBearbeiten 23
6.3.4.3. LkwBearbeiten 23
6.3.5. MVC Components/Interactions 24
6.3.5.1. Components 24
6.3.5.2. Interactions: Actor and MVC 25
6.4. MVC Implementation 26
6.4.1. Import required files 26
6.4.2. Configuration files 26
6.4.3. Model: Persistent Objects 30
6.4.3.1. Overview 30
6.4.3.2. Create Object Model 30
6.4.3.3. Create VA object classes 39
6.4.3.4. Create POM Generator 39
6.4.3.5. Create POM 43
6.4.3.6. Create Runtime POM 46
6.4.3.7. Modify POM-releated classes 48
6.4.3.8. Create database 49
6.4.3.9. Create database tables 51
6.4.4. Model: Activities (mit fachlicher Logik) 54

6.4.4.1. ZZZActivitySuchen.java 55
6.4.4.2. ZZZActivityKundeBearbeiten.java 57
6.4.4.3. ZZZActivityLkwBearbeiten.java 59
6.4.5. Model: Transitions 60
6.4.5.1. Configuration files (XML file) 60
6.4.6. Views 61
6.4.6.1. de.dekra.tutorial.view.ZZZPanelSuchen 61
6.4.6.2. de.dekra.tutorial.view.ZZZPanelKundeBearbeiten 65
6.4.6.3. de.dekra.tutorial.view.ZZZPanelLkwBearbeiten 66
6.4.6.4. de.dekra.tutorial.view.ZZZFrame 66
6.4.7. Controller: GUIController class (ZZZGC) 68
6.4.8. Main class (ZZZMain) 70
6.5. Test (manually) 72
6.5.1. Requirements 72
6.5.2. Compute class path 72
6.5.3. Start 72
6.5.4. Kunden suchen 73
6.5.5. Go to activity specified by XML file and canStart 73
6.5.6. Save changes to the database 74
6.5.7. Go direct to an activity 74
6.6. Test (with TestSuite) 75
6.6.1. Requirements 75
6.6.2. Create ActivityManagerTestSuite 75
6.6.3. Run single test 77
6.6.4. Batch test (run multiple tests) 78
6.6.5. Failing test 78
6.6.6. Failing test case 79
6.7. Modify the object net 80
6.7.1. Add variable name2 to Kunde 80
6.7.1.1. Modify object net 80
6.7.1.2. Create VA classes 80
6.7.2. Create new Runtime-POM 81
6.7.2.1. Create POMG 81
6.7.2.2. Generate POM 81
6.7.2.3. Generate RT-POM 81
6.7.2.4. Import into VA RT-POM 81
6.7.2.5. Modify RTPOM spec 81
6.7.3. Add table column NAME2 to KUNDE 82
6.7.3.1. Generate SQL 82
6.7.3.2. Execute SQL against DB 82
6.7.4. Modify views 82
6.7.4.1. Modify ZZZPanelSuchen 82
6.7.4.2. Modify ZZZPanelKundeBearbeiten 83
6.7.5. Test 83
6.8. ViewPorts 85
6.8.1. Modify ZZZPanelKundeBearbeiten 85
6.8.2. Modify GUIController 85
6.8.3. Create activity viewport 85
6.8.4. Create object viewport 86
6.8.5. Test 86
6.9. Transactions 88
6.9.1. Why transactions 88
6.9.2. Transactions: Non-isolated 88
6.9.3. Transactions: Isolated 88

6.10. Adding/deleting objects (from database) ... 89
6.11. Enable/disable view components ... 90
6.12. Validating input ... 91
6.13. Implementing tooltips ... 92
7. Anwendungsszenarien (Komplette Beispiele) ... 93
7.1. Einbindung in die DSE Gesamtarchitektur ... 94
7.2. Einbindung in den Kommunikationsfluss einer Gesamtanwendung ... 95
7.3. Fat-client (mvc) ... 96
7.4. EJB-Container ... 97
7.5. Web-Server ... 98
7.6. Workflow-System (DSE-OFC) ... 99
8. Concepts ... 100
8.1. Triade (Manager, Konfiguration, Creator) ... 101
8.1.1. Überblick ... 101
8.1.2. Begriffsdefinition ... 101
8.1.3. Die Ablaufumgebung ... 103
8.1.4. Die Verwaltung der Nachrichten ... 103
8.1.5. Das „Innen und Aussen“ ... 103
8.2. Manager ... 105
8.2.1. Aufgaben des Managers : Aussensicht ... 105
8.2.2. Aufgaben des Managers : Innensicht ... 105
8.2.3. Aufgaben des Managers : Kernelsicht ... 105
8.2.4. Aufgaben des Managers : Usersicht ... 105
8.2.5. Die Kontrakte : ... 105
8.3. Configuration ... 106
8.3.1. Aufgaben der Konfiguration : Abstraktion der Hierarchischen Abbildung eines Geschäftsprozesses ... 106
8.3.2. Aufgaben der Konfiguration : Grundlage der Instrumentierung einer Aktivität. ... 106
8.3.3. Ablage der Informationen ... 106
8.3.4. Konfigurierbarkeit ... 106
8.4. Creator ... 107
8.4.1. Aufgaben des Creators: Schnittstelle Kernel/User ... 107
8.4.2. Aufgaben des Creators: Workspace-Management ... 107
8.5. Activities ... 108
8.5.1. Beschreibung der fachlichen Logik ... 108
8.5.2. Fachliche Ablauflogik ... 108
8.5.3. Datenhaltung ... 108
8.5.4. Passive Ablaufsteuerung ... 108
8.5.5. Implementation der Vorbedingungen und Nachbedingungen , Datenzugriffe ... 109
8.5.6. Implementation der Geschäftsregeln ... 109
8.5.7. Design ... 109
8.5.8. Implementation ... 109
8.5.9. Implementation einer einzelnen Activity ... 110
8.6. Kommunikation Manager – Activity ... 114
8.6.1. Manager -> Activity ... 114
8.6.2. Activity -> Manager ... 115
8.7. Transitions ... 117
8.8. Beschreibung der XML-Konfiguration ... 118
8.8.1. Die DTD (Document Type Definition) ... 118
8.8.2. Die Elemente ... 118

8.8.3. Beschreibung einzelner Aktivitäten ... 118
8.8.4. Beschreibung der Transitionen , Zustandsübergänge ... 120
8.8.5. Beschreibung der Domainobjekte ... 121
8.8.6. Beschreibung der Userinterface-Elemente ... 122
8.9. Workspace ... 123
8.9.1. Definition ... 123
8.9.2. API (von Creator) ... 124
8.9.3. API (von Activity) ... 124
8.10. Slots ... 125
8.11. Transactions (committing, persistence, etc.) ... 126
8.12. SDI ... 127
8.13. Services ... 128
9. Tools ... 129
9.1. Tool Center ... 130
9.2. Object Model Browser ... 131
9.3. Model->POM Generator ... 132
9.4. Model DB Mapping Tool ... 133
9.5. Interactive SQL ... 134
9.6. Runtime POM Generator ... 135
9.7. Logger ... 136
10. API ... 137
10.1. Activity ... 138
10.1.1. Overview ... 138
10.1.2. public void addBusinessRule(String objkey, String attributeKey, PropertyChangeListener pcl) ... 138
10.1.3. public void addActionListener(String objkey, ActionListener acl) ... 139
10.1.4. public void addVetoableBusinessRule(String objkey, String attributeKey, VetoableChangeListener vcl) ... 139
10.1.5. public Iterator iterator() ... 140
10.1.6. public Object get(String key) ... 140
10.1.7. public boolean hasObject(String key) ... 140
10.1.8. public boolean hasSlot(String key) ... 140
10.1.9. public void put(String key, Object item) ... 141
10.1.10. public boolean canComplete() ... 141
10.1.11. public boolean canComplete(int nPartialMode) ... 141
10.1.12. public void doStart() ... 141
10.1.13. public boolean canStart() ... 142
10.1.14. public void done() ... 142
10.1.15. public void failed() ... 142
10.1.16. public void failed(String msg) ... 142
10.1.17. public void failed(Exception exc) ... 143
10.2. Manager ... 144
10.2.1. Interface ActivityGuiEventSupport ... 144
10.2.2. Interface ActivityNotificationSupport ... 144
10.2.3. ActivityWorkflowSupport ... 144
10.2.4. Interface ActivityTransactionalSupport ... 144
10.3. Configuration ... 145
10.3.1. Interface Configuration ... 145
10.4. Creator ... 146
10.4.1. Interface Creator ... 146

10.5. Workspace 147
10.5.1. Interface Workspace 147
10.6. Context 148
10.6.1. Interface Context 148
10.7. Technische Beschreibung der Implementation des ActFW (API) 149
10.7.1. Manager 149
10.7.2. Configuration 149
10.7.3. Creator 149
10.7.4. Activity 149
10.7.5. Workspace 149
10.7.6. Context 149
10.8. API reference (ask, alert, tell, prompt, select) ?? 150
11. Trouble shooting 151
12. FAQ 152
13. Glossary 153
14. Index 154
15. Abbildungverzeichnis 155

1. Copyrights

2. Dokumentübersicht

Dokumentübersicht:

- Release notes. Beschreibt was neues gibt’s in jeder version von Frameworks und diesem Handbuch.
- Die Vorteile von Frameworks. Ein Übersicht der Vorteilen, die man von der Benutzung Frameworks geniessen kann.
- Installation. Eine ausführliche schritt-für-schritt Beschreibung, wie man alles, wann man für Frameworks braucht, auf einen Rechner installieren kann.
- Tutorial. Eine ausführliche schritt-für-schritt Einleitung zu Framworks. Empfehlenswert für den, der sofort „hands-on“ Erfahrung mit Frameworks bekommen will. Hinweis: Es geht davon aus, dass die Installation ist schon fertig.
- Anwendungsszenarien. Komplette, komplexe Beispielanwendungen. Besonders interessant für den, der sofort sehen will, was für echte Anwendungen man mit Frameworks bauen kann.
- Concepts. Eine umfassende Darstellung der grundlegenden Konzepte von Frameworks. Besonders empfehlenswert für den, der schon „hands-on“ Erfahrung mit Frameworks hat und Frameworks besser verstehen will.
- Tools. Beschreibt den Umgang mit Frameworks tools.
- API. Detaillierte der Frameworks API.
- Troubleshooting. Wie kann man typische Probleme aufheben.
- FAQ. Antworte auf häufig gestellten Fragen.
- Glossary.
- Index.
- Abbildungsverzeichnis.

3. Release notes

3.1. Activity Framework

3.2. Benutzerhandbuch

4. Die Vorteile von Frameworks

Warum brauchen Sie Frameworks? Es gibt eine ganze Reihe von Gründen, warum man sein Produktivität mit Frameworks drastisch steigern kann. In diesem Kapitel wird in einer insofern wie möglich kurzen Darstellung klar gemacht, was man mit Frameworks verwirklichen kann.

NOTE: This chapter should show typical scenarios in which framworks provides big advantages. This is NOT a concepts or tutorial chapter, but rather a demonstration chapter.

5. Installation

Note that the installation can have various options. For example, a shared library instead of local. In any case, this section describes the installation that was used for the tutorial. Try to follow this installation instructions as closely as possible if you are planning on doing the step-by-step tutorial.

5.1. Requirements

1. Windows NT/2000. The tutorial was written using NT 4.0 German.
2. VA 3.02.

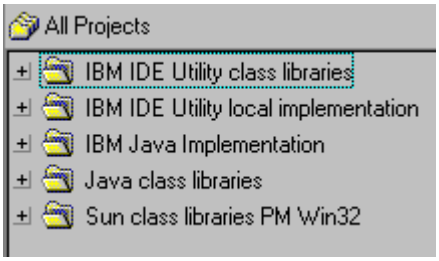


Abbildung 1. VA classes

3. Oracle 8i Lite. (tutorial assumes installed to **\orant**).

5.2. Step-by-step

5.2.1. Load VA projects

4. Import from **ETC.dat**:

- Javax rmi
- JAXP
- JNDI
- Old stuff

5.2.2. Load Mynd DSE

5. Import from **DSE.dat**:

- Mynd DSE Activity
- Mynd DSE Application
- Mynd DSE Application AWT Support
- Mynd DSE Application Swing Support
- Mynd DSE Base
- Mynd DSE Object Behavior Base
- Mynd DSE Persistence
- Mynd DSE Services
- Mynd DSE TestSuite

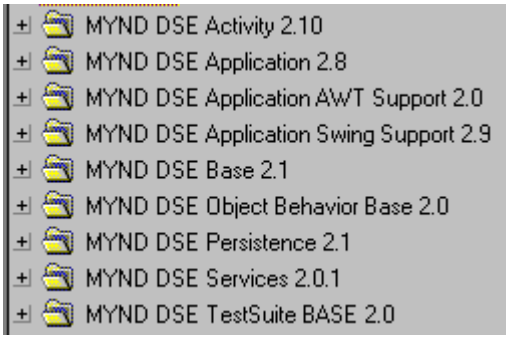


Abbildung 2. DSE classes

5.2.3. Install Frameworks tools

5.2.3.1. Install files

6. Install the ToolCenter.

- Import from **DSETOOLS.dat**.
- Unzip **IDE.zip** to **d:\IBMVJava\ide** (with directories).
- Copy **uml.dtd** to **d:**.
- Select Window / Options.
- Select Resources.
- For Workspace class path: Enter: **D:\IBMVJava2\ide\tools\MYND\;**.

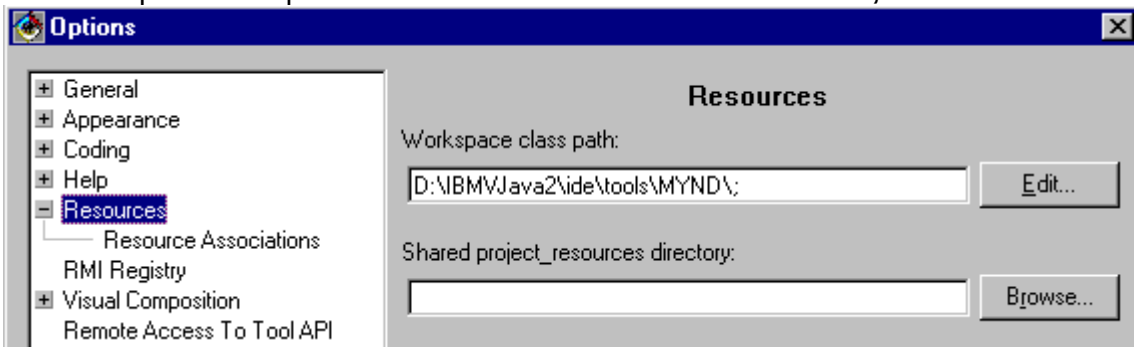


Abbildung 3. Options / Resources

- Click OK.

5.2.3.2. Configure Object Model Browser

7. Start VA.

8. From VAJ menu: Select: Workspace / Tools / Mynd environment / Open ToolCenter. The ModelIntegrator appears:

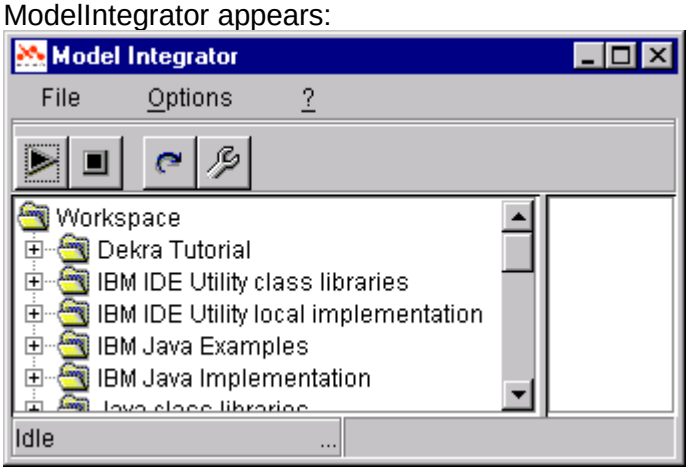


Abbildung 4. Model Integrator dialog (117)

9. Select Options / Preferences. The preferences dialog appears:

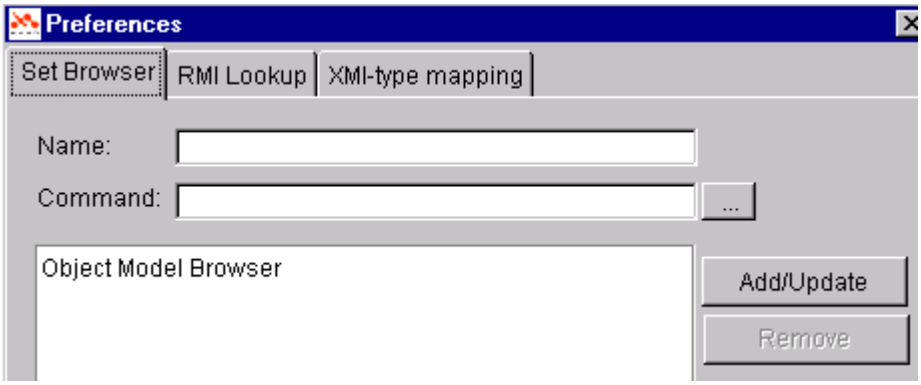


Abbildung 5. Model Integrator Preferences dialog (185)

10. Select Object Model Browser.
11. Click on the button besides the Command text field. The Open File dialog appears.
12. Double-click on the [VA drive : directory] \ide\tools\MYND\browser\dsetools.exe.

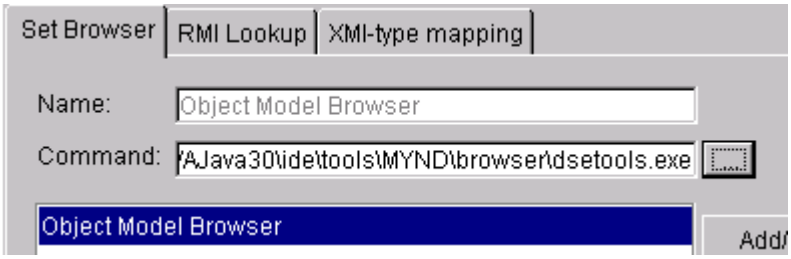


Abbildung 6. Browser command settings in the Model Integrator Preferences dialog (186)

13. Click OK.
14. Close the model integrator.

6. Tutorial

6.1. Overview

This tutorial contains the following:

- Using supplied code. Describes how to load code (allows you to start at a chapter without having completed the previous chapters).
- MVC Design. Describes the design of a simple system.
- MVC Implementation. Describes the implementation of the design.
- Testing (manual).
- Testing with TestSuiteBrowser.
- Modifying the object net.
- Viewports
- Transactions
- Adding/deleting objects from database
- Enable/disable viewcomponents
- Validating input.
- Implementing tooltips.

6.2. Using the supplied code

The completed code for the examples described in this tutorial are supplied in **TUTORIAL.dat**. Other files, such as the configuration files, are also supplied.

Example

Assume that you want to start immediately at section „6.5 Testing“, and do not want to complete „6.4 MVC Implementation“.

Simply follow the instructions at the beginning of „6.5 Testing“, which explain what is required to use the supplied code.

6.3. MVC Design

In this section you will design a system to solve a simple problem.

Note that the example to be implemented is very simple. Remember, however, that the goal of this part of the tutorial is to become familiar with the tools.

6.3.1. UseCases

You want to create an application that allows you to search for one of the following:

- Kunde id
- Lkw id

in a database.

If a Kunde or Lkw is found in the database, then a dialog allows you to edit the info and save in the database.

A Kunde will be assigned an Lkw, and vice versa.

Use cases:

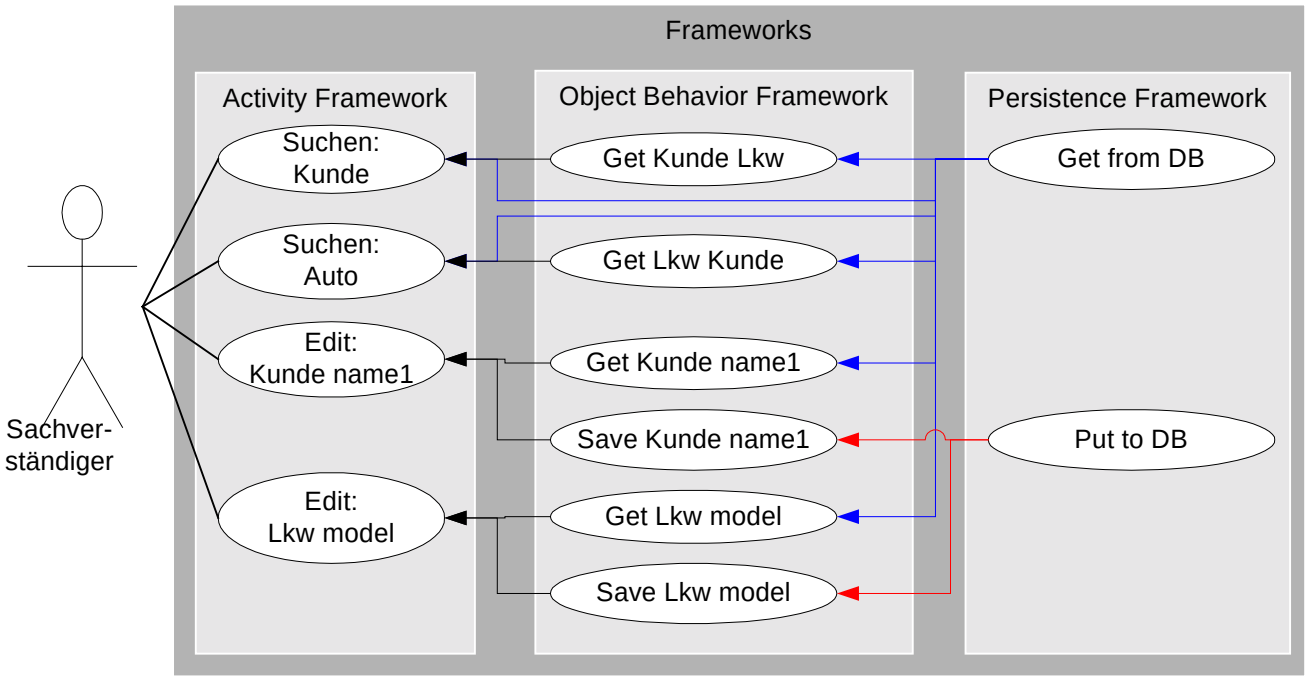


Abbildung 7. Use cases (911)

6.3.2. Activities

6.3.2.1. Activities Overview

The following diagram shows the activities:

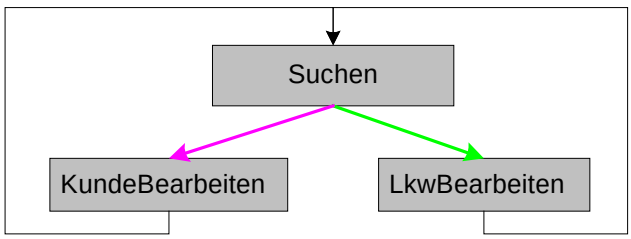


Abbildung 8. Activities (913)

6.3.2.2. Suchen

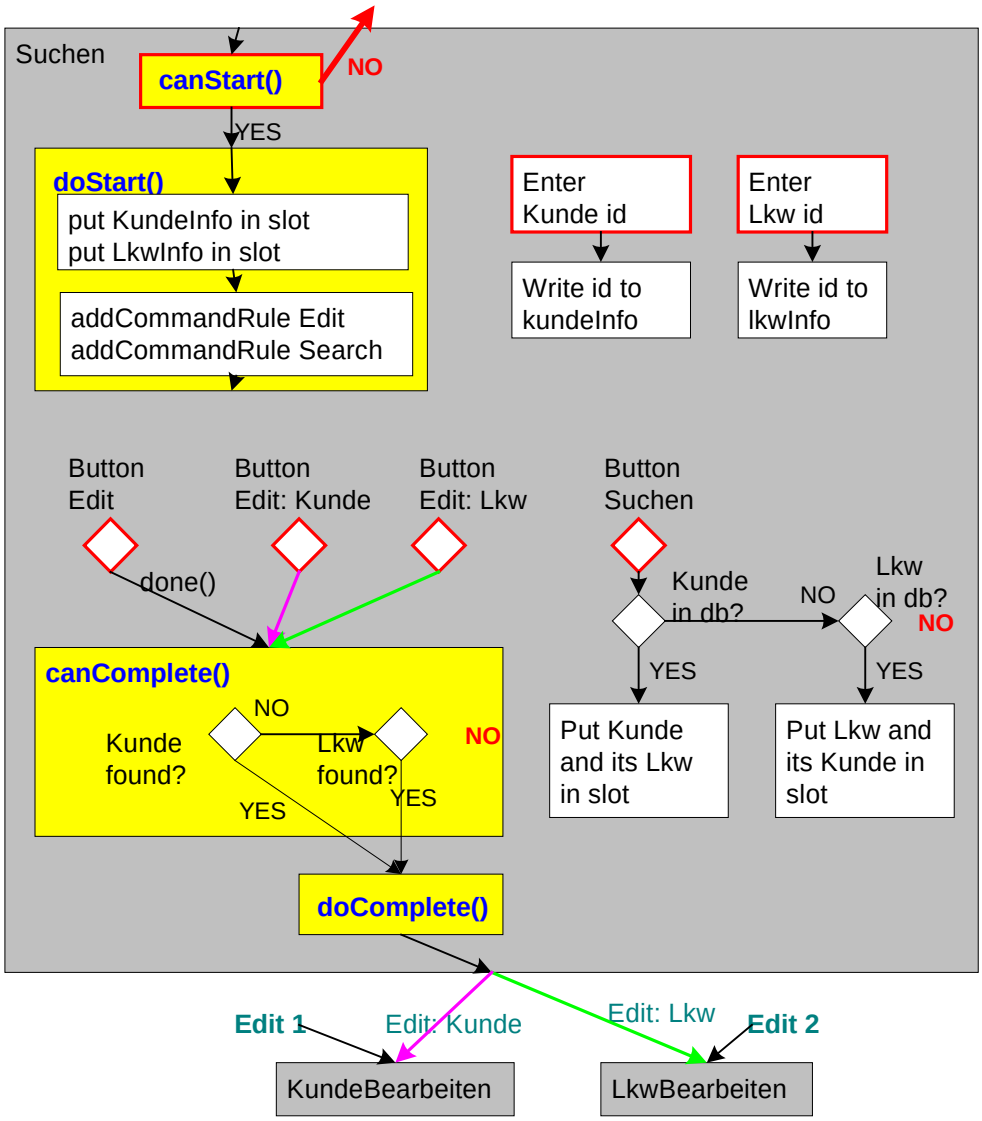


Abbildung 9. Activity Suchen diagram (912)

6.3.2.3. KundeBearbeiten

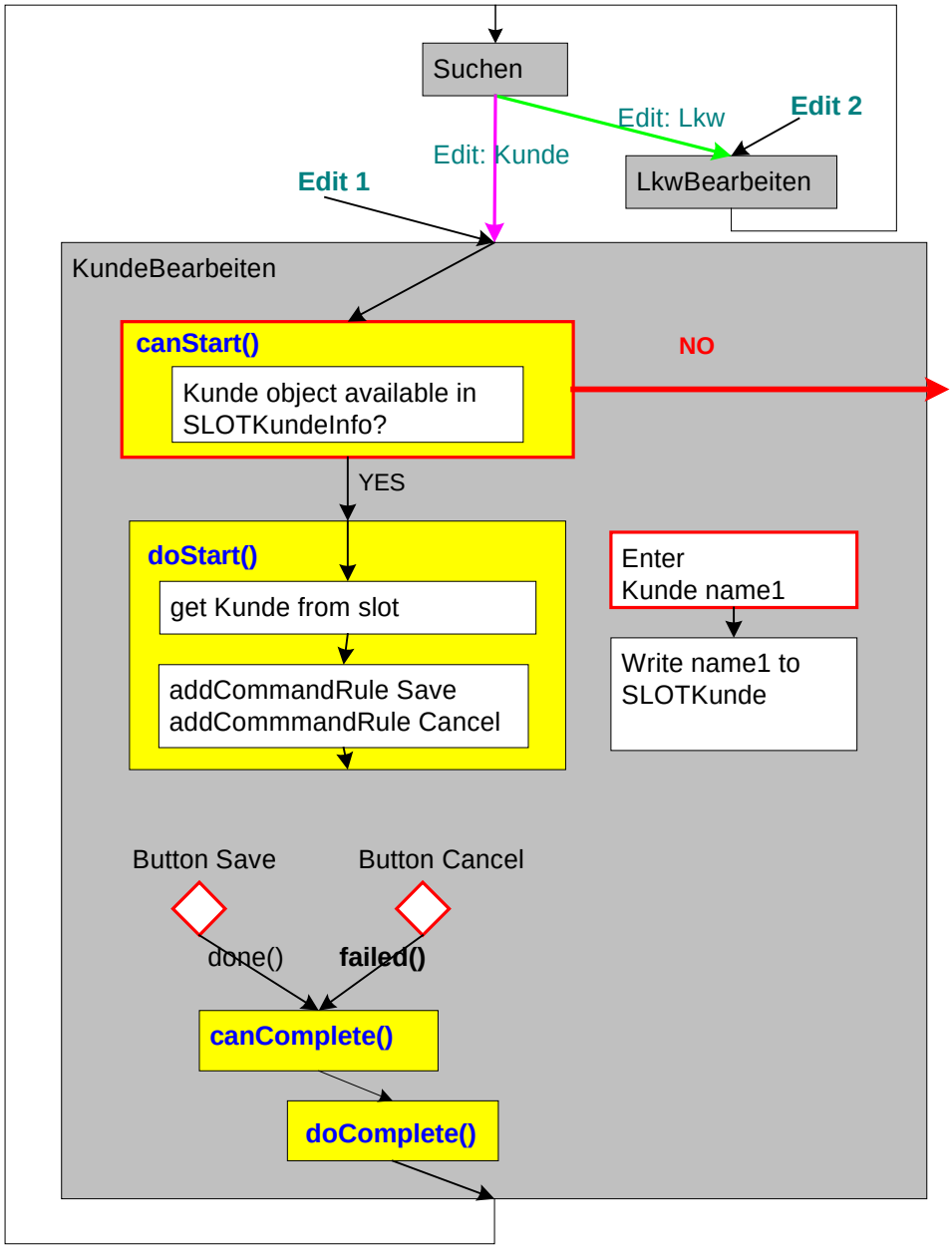


Abbildung 10. Activity KundeBearbeiten diagram (914)

6.3.2.4. LkwBearbeiten

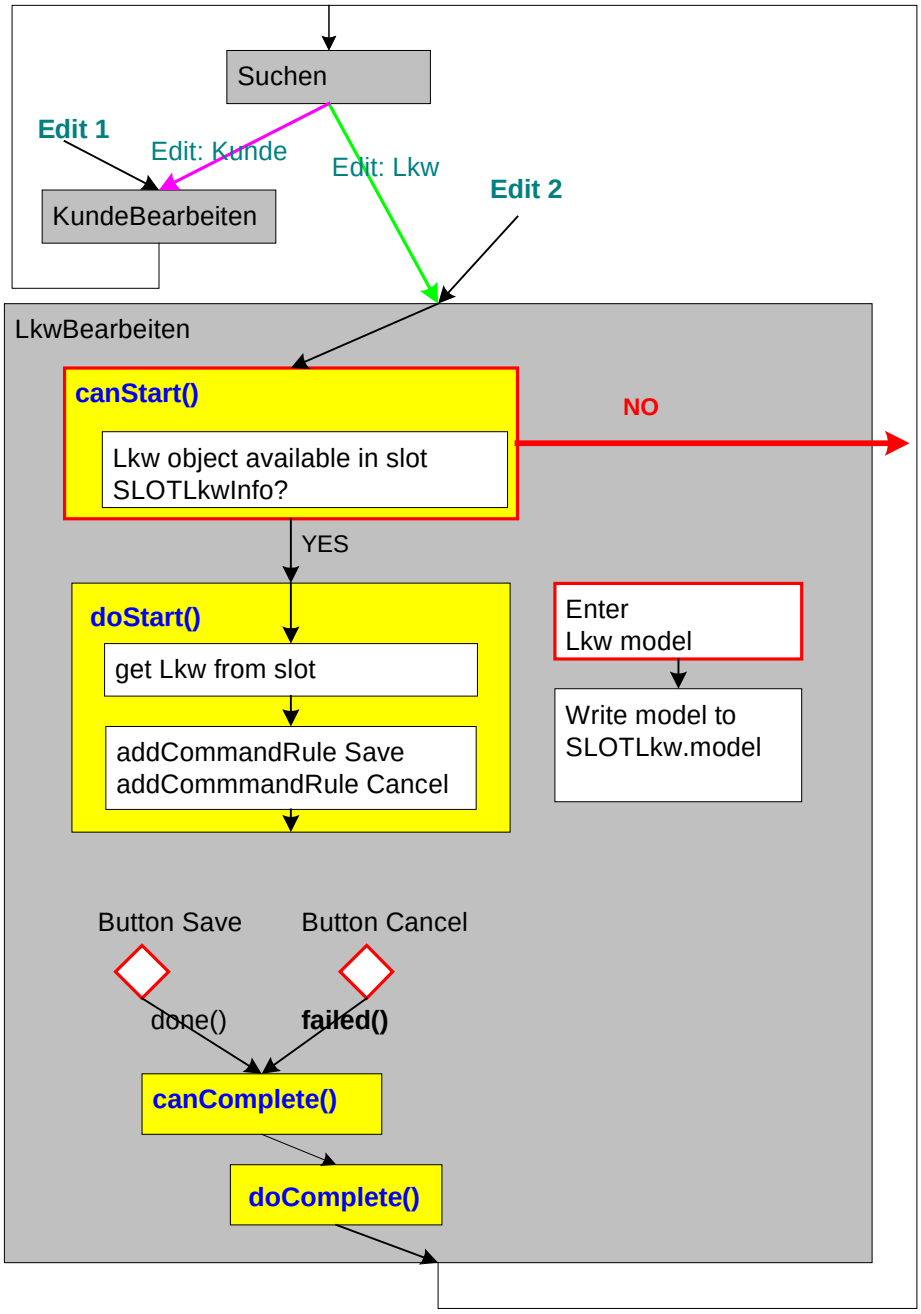


Abbildung 11. Activity LkwBearbeiten diagram (915)

6.3.3. Persistent Objects

6.3.3.1. Objects

The DO's are the actual objects accessed and modified by the activities.

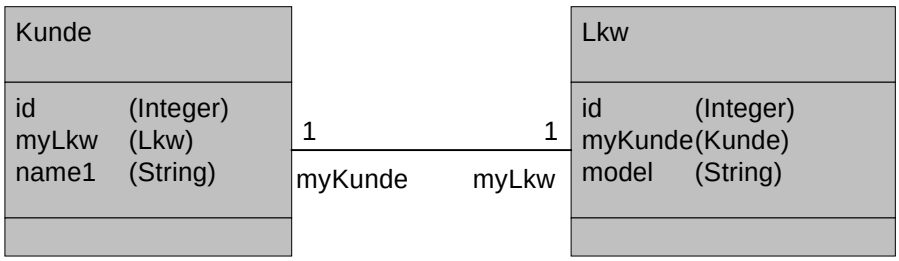


Abbildung 12. Domain objects (907)

6.3.3.2. Object accessors (created with OBF)

Created with OMB.

Includes special get, set accessors that support:

- Typing (including association (relationship) verification).
- Persistence.
- Transactions.

OBF Accessors diagram:

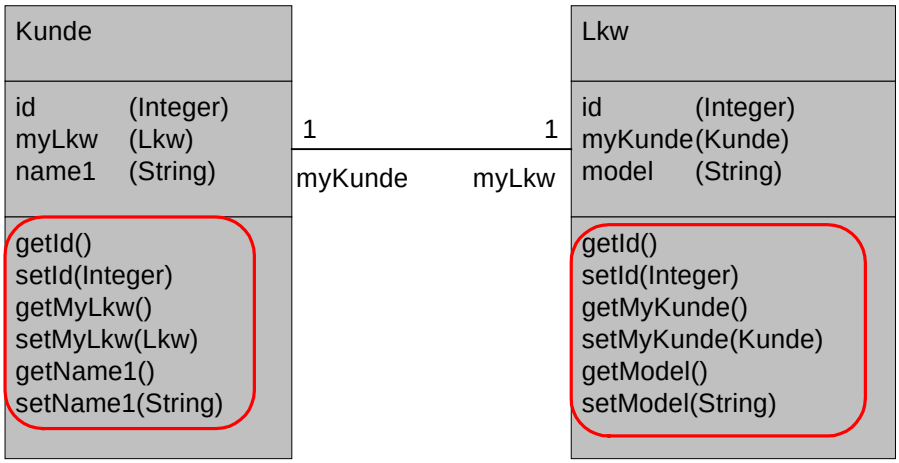


Abbildung 13. Object accessors created with OMB (906)

6.3.3.3. DatabaseTables

Generated with POM tools (using info from OBF model).

The DO's are stored in a database.

Tables:

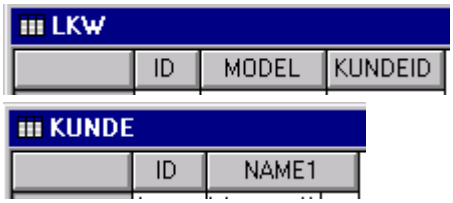


Abbildung 14. Database tables for domain objects (181, 182)

6.3.3.4. RT POM (Object <-> DatabaseTable interface)

Interface between OBF class accessors and database.

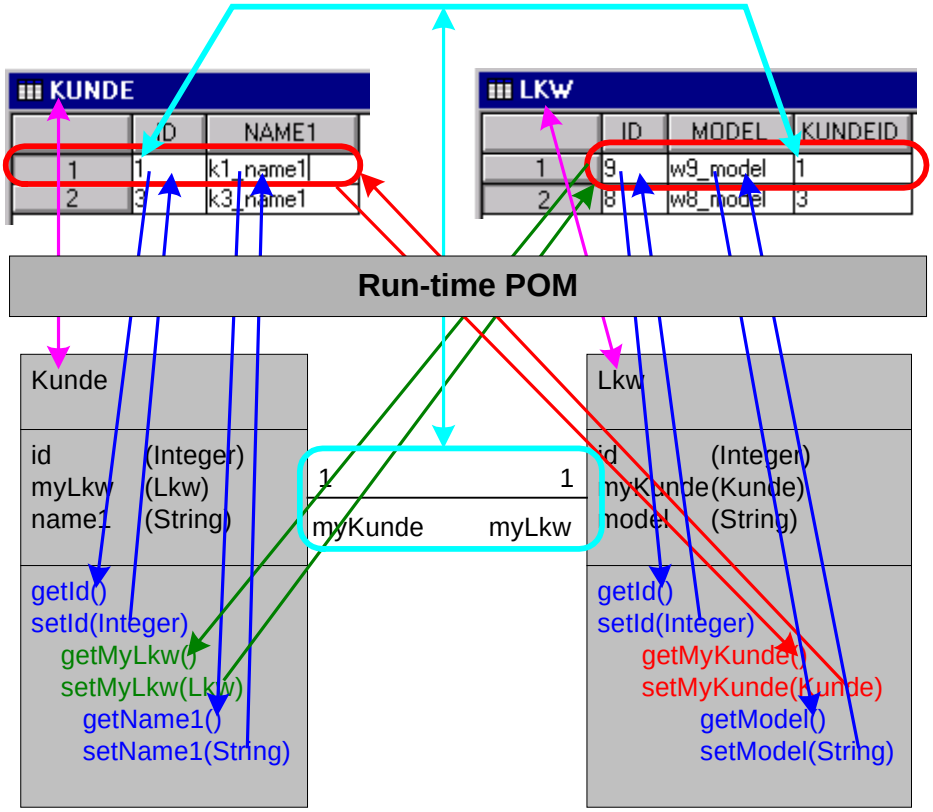


Abbildung 15. RT-POM interface between objects and database tables (908)

6.3.4. Views

Note: The diagrams include the controlConnectDescriptors (in blue). This info is required for understanding the interaction diagram following this section.

6.3.4.1. Suchen

When the program starts:

Note: "cCD.c->" = "controlConnectDescriptor.content->"

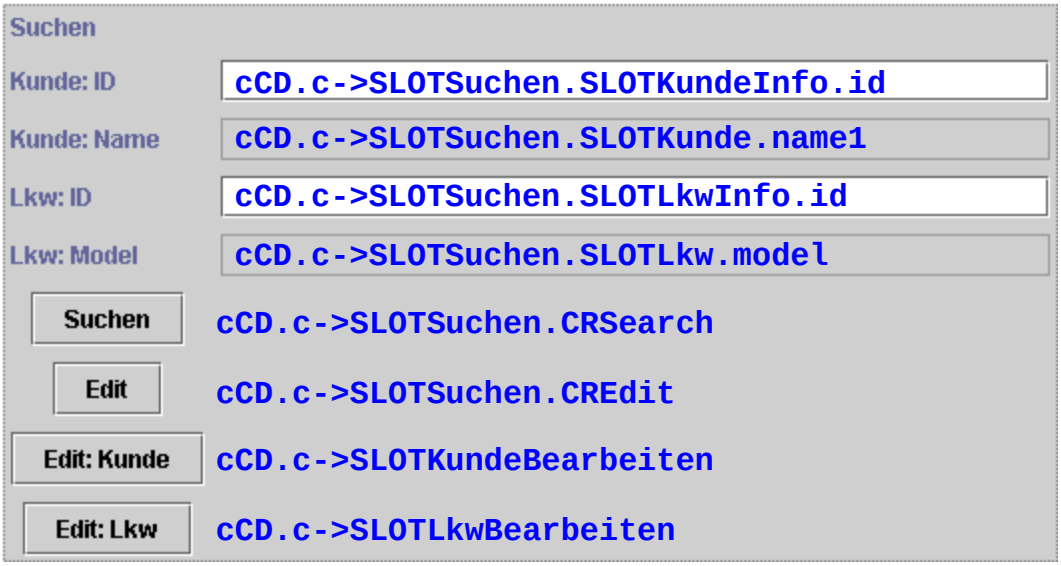


Abbildung 16. Suchen dialog (919)

Note that only the ids can be entered.

6.3.4.2. KundeBearbeiten

The dialog for KundeBearbeiten:

Note: "cCD.c->" = "controlConnectDescriptor.content->"

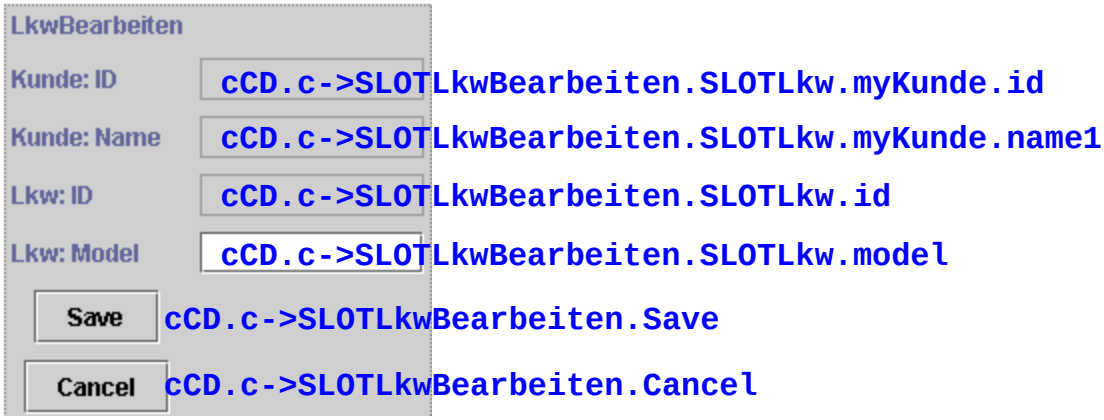
Abbildung 17. KundeBearbeiten dialog (920)

Note that only the name can be changed.

6.3.4.3. LkwBearbeiten

The dialog for LkwBearbeiten.

Note: "cCD.c->" = "controlConnectDescriptor.content->"



Note that only the model can be changed.

Abbildung 18. LkwBearbeiten dialog (921)

6.3.5. MVC Components/Interactions

6.3.5.1. Components

Components in yellow you must create.

ADD SLOTs and POMG etc. to diagram.

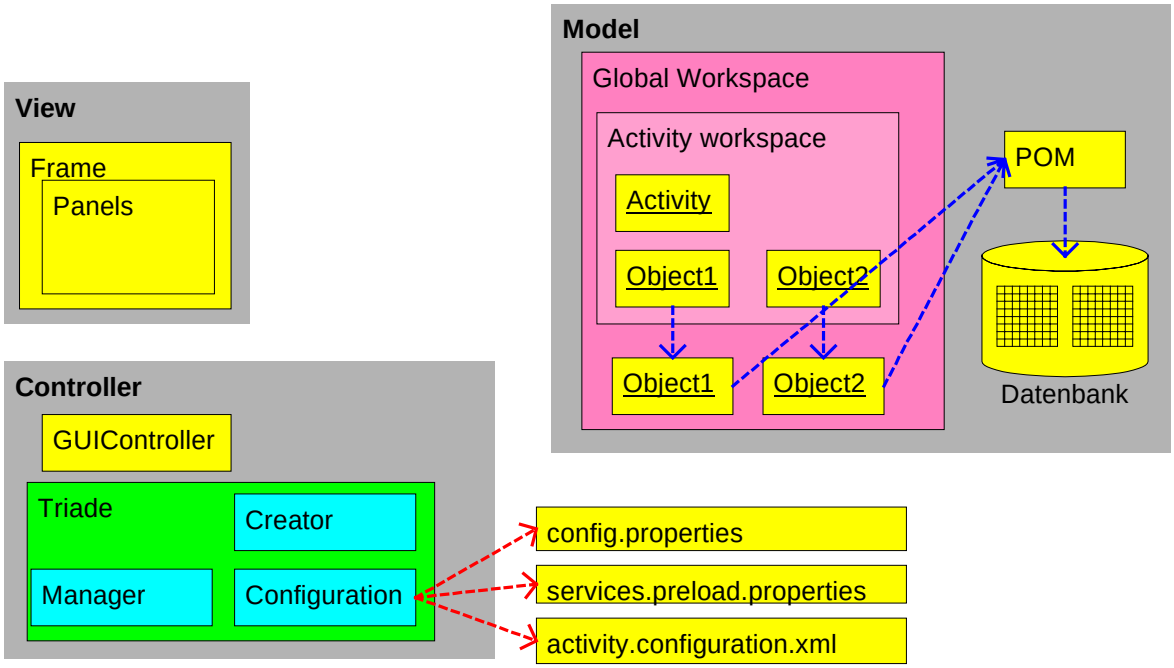


Abbildung 19. MVC components (909)

6.3.5.2. Interactions: Actor and MVC

Interactions.

Note: See the controlConnectorDescriptions above.

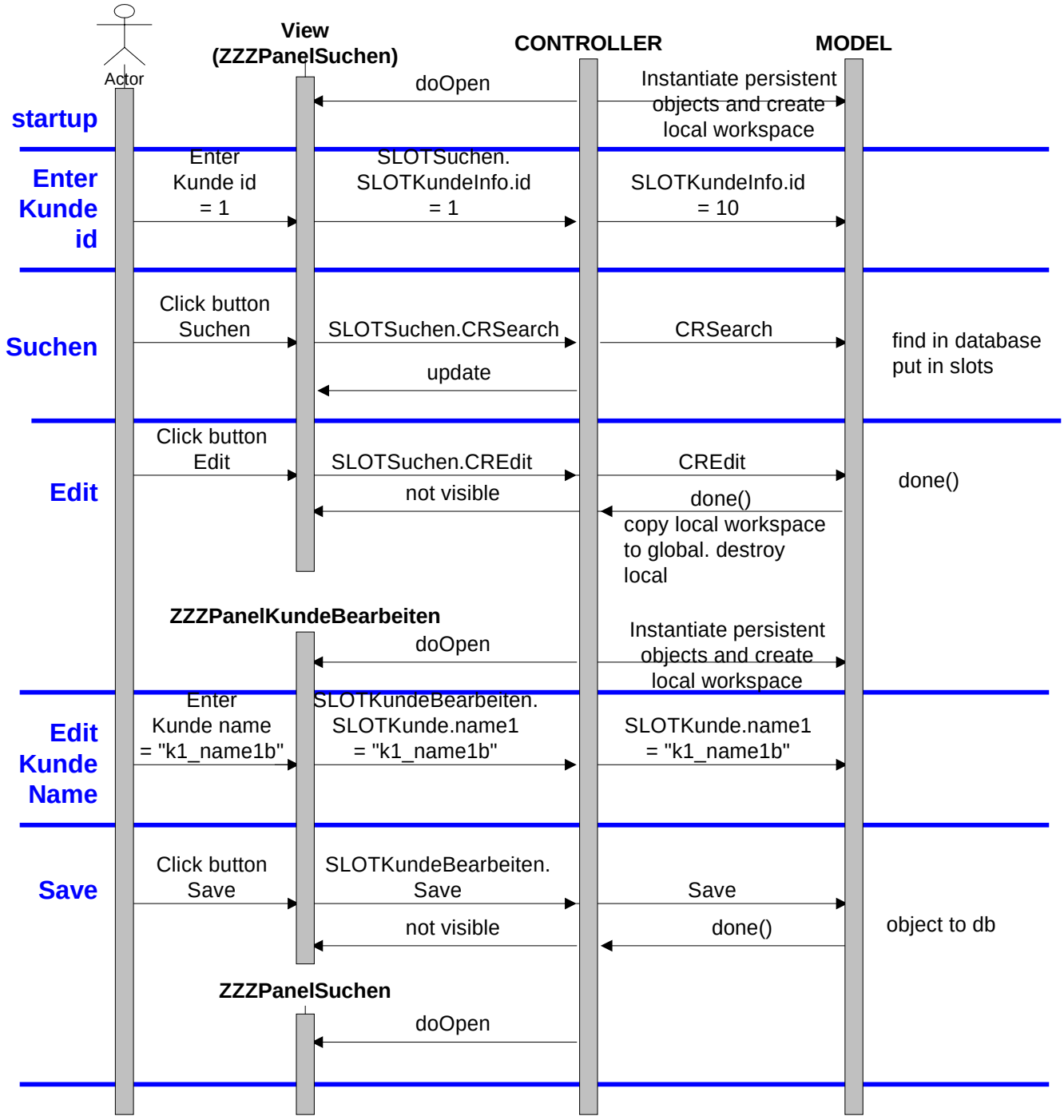


Abbildung 20. MVC actor input interactions (910)

6.4. MVC Implementation

Completed code available in TUTORIAL.DAT project Dekra Tutorial version 2.0.

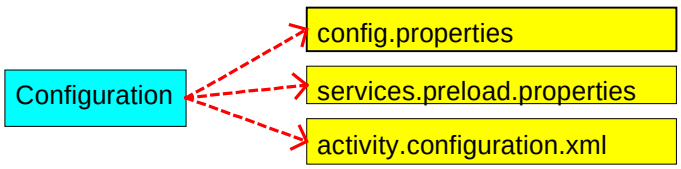
You will now perform all the steps for a "simple" implementation.

6.4.1. Import required files

1. Import from **TUTORIAL.dat**: **Dekra Tutorial** version **base**. This contains required classes for the tutorial.

6.4.2. Configuration files

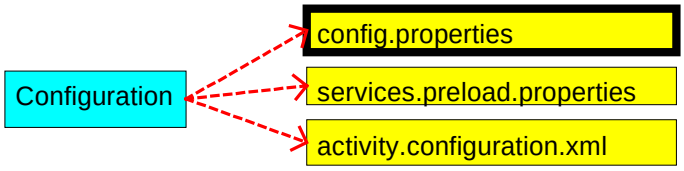
The following files specification the application configuration:



(909b)

6.4.2.1. Main configuration (config.properties)

2. Create config.properties.



(909a)

Verzeichnis/Dateiname

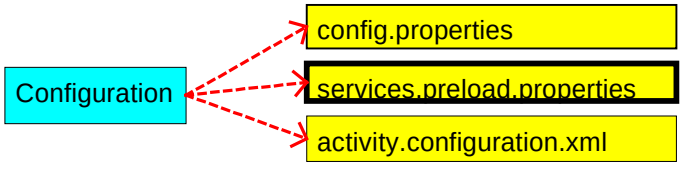
```
[drive letter]:\[VA directory]\ide\project_resources\Dekra Tutorial\
                config.properties
```

Content

```
# lines beginning with "#" are comments
 # config.properties: configuration of the application
# activity.configuration.xml = Activity configuration file
 processName=activity.configuration.xml
# services.preload.properties = service factory
 serviceFactoryFactoryName=services.preload.properties
# de.dekra.tutorial.controller.ZZZGC = GUI controller
 guiControllerName= de.dekra.tutorial.controller.ZZZGC
# de.dekra.tutorial.view.ZZZFrame = frame for the views
 viewName= de.dekra.tutorial.view.ZZZFrame
# searchAgent (used for retrieving objects from the database)
 searchAgent=com.mynd.dse.ps.searchagent.SearchAgentImpl
# pom initializer = de.dekra.tutorial.model.ZZZPOMInitializer
 persistenceManager= de.dekra.tutorial.model.ZZZPOMInitializer
# database settings
 databaseDriver=sun.jdbc.odbc.JdbcOdbcDriver
 databaseSystemName=jdbc:odbc:ZZZDBDSN
 databaseUserID=SYSTEM
 databasePassword=SYSTEM
# eof
```

6.4.2.2. Services configuration (services.preload.properties)

3. Create services.preload.properties.



(909c)

Verzeichnis/Dateiname

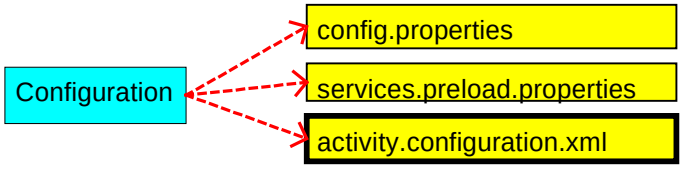
[drive letter]:\[VA directory]\ide\project_resources\Dekra Tutorial\
services.preload.properties

Content

```
# lines beginning with "#" are comments
# format of an entry in services.preload.properties
# tag=Tag_Name : the Service name to be preloaded
# tag.factory=Tag_FactoryClassName :  the ClassName of the Factory
    that is used to create the service
# tag.factory.params=Tag_FactoryParams :  A String that will be given
    to the Factory as additional params
# tag.factory.qos=Tag_QualityOfService :  the qualityOfService under
    which the Service will be registered
# tag.priority=Tag_Priority :  defines the order under which the
    services will be registered. 0=first 99=last
# please keep in mind that several services can be registered by the
    same name and
# only differ in the qos -tag. These entries must be given a different
    tag.
# POM configuration
 POM=persistenceManager
 POM.factory=de.dekra.tutorial.model.PomFactory
 POM.factory.params=***
 POM.priority=1
 POM.qos=
# SearchAgent configuration
 S_AGENT=searchAgent
 S_AGENT.factory=de.dekra.tutorial.model.SearchAgentFactory
 S_AGENT.factory.params=com.mynd.dse.ps.searchagent.SearchResultTypeObject
 S_AGENT.qos=
 S_AGENT.priority=2
# eof
```

6.4.2.3. Activity configuration (activity.configuration.xml)

4. Create activity.configuration.xml.



(909d)

Verzeichnis/Dateiname

[drive letter]:\[VA directory]\ide\project_resources\Dekra
Tutorial**activity.configuration.xml**

Content

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ActivityFlow SYSTEM "activity.configuration.dtd">
<!--  $Header: activity.configuration.xml$ -->
<!--- top level --->
<ActivityFlow>
```

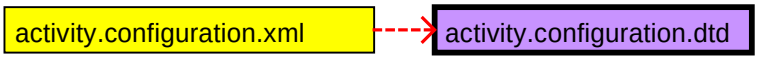
```
<!--- activity Suchen configuration: --->
<!--- 1. slot name "SLOTSuchen" --->
<!--- 2. package.class --->
<!--- 3. view panel ("user interface") --->
<!--- 4. transaction type ("NEW_CONTEXT") --->
 <Activity name="SLOTSuchen" class="de.dekra.tutorial.model.ZZZActivitySuchen"
                userinterface="de.dekra.tutorial.view.ZZZPanelSuchen"
                transacted="NEW_CONTEXT">
<!--- 5. domain objects for the workspace local to activity --->
 <DomainObjects>
<!--- 5.1. slot name ("SLOTKunde") --->
<!--- 5.2.class --->
       <DomainObject name="SLOTKunde" class="de.dekra.tutorial.model.Kunde"/>
       <DomainObject name="SLOTLkw" class="de.dekra.tutorial.model.Lkw"/>
       <DomainObject name="SLOTKundeInfo"
                class="de.dekra.tutorial.model.ZZZKundeInfo"/>
       <DomainObject name="SLOTLkwInfo" class="de.dekra.tutorial.model.ZZZLkwInfo"/>
     </DomainObjects>
   </Activity>
<!--- activity KundeBearbeiten configuration --->
<!--- note persistent="NEW_CONTEXT". required for saving objects to
     database --->
 <Activity name="SLOTKundeBearbeiten"
                class="de.dekra.tutorial.model.ZZZActivityKundeBearbeiten"
                userinterface="de.dekra.tutorial.view.ZZZPanelKundeBearbeiten"
                persistent="NEW_CONTEXT">
     <DomainObjects>
       <DomainObject name="SLOTKunde" class="de.dekra.tutorial.model.Kunde"/>
       <DomainObject name="SLOTLkw" class="de.dekra.tutorial.model.Lkw"/>
 </DomainObjects>
   </Activity>
<!--- activity LkwBearbeiten configuration --->
 <Activity name="SLOTLkwBearbeiten"
                class="de.dekra.tutorial.model.ZZZActivityLkwBearbeiten"
                userinterface="de.dekra.tutorial.view.ZZZPanelLkwBearbeiten"
                persistent="NEW_CONTEXT">
<!--- domain objects for the workspace local to activity --->
     <DomainObjects>
       <DomainObject name="SLOTKunde" class="de.dekra.tutorial.model.Kunde"/>
       <DomainObject name="SLOTLkw" class="de.dekra.tutorial.model.Lkw"/>
 </DomainObjects>
   </Activity>
<!--- root activity specification --->
 <Activity name="root" start="SLOTSuchen" abend="IGNORE">
<!--- domain objects for the global workspace --->
     <DomainObjects>
       <DomainObject name="SLOTKunde" class="de.dekra.tutorial.model.Kunde"/>
       <DomainObject name="SLOTLkw" class="de.dekra.tutorial.model.Lkw"/>
 </DomainObjects>
<!--- transitions specification --->
     <Transitions>
       <Transition>
<!--- allowed transitions after ZZZActivitySuchen --->
         <Start activity="SLOTSuchen"/>
         <End activity="SLOTKundeBearbeiten"/>
         <End activity="SLOTLkwBearbeiten"/>
         <End/>
       </Transition>
       <Transition>
```

```
<!--- allowed transitions after ZZZActivityKundeBearbeiten --->
        <Start activity="SLOTKundeBearbeiten"/>
        <End activity="SLOTSuchen"/>
        <End/>
      </Transition>
      <Transition>
<!--- allowed transitions after ZZZActivityLkwBearbeiten --->
        <Start activity="SLOTLkwBearbeiten"/>
        <End activity="SLOTSuchen"/>
        <End/>
      </Transition>
    </Transitions>
  </Activity>
</ActivityFlow>
<!--- eof --->
```

6.4.2.4. XML file DTD

5. Create activity.configuration.dtd.

The Data Type Description file describes the allowed valid contents of the XML file.



(909e)

Verzeichnis/Dateiname

DTD file:

```
[drive letter]:\[VA directory]\ide\project_resources\Dekra
               Tutorial\activity.configuration.dtd
```

Contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--  $Header: activity.configuration.dtd$ -->
<!ELEMENT ActivityFlow (Userinterface*,Activity*)>
<!ELEMENT Activity ((DomainObjects?,Transitions?)|(Transitions?,DomainObjects?))>
<!ATTLIST Activity name CDATA #REQUIRED
  class CDATA #IMPLIED
  abend  ( ABEND | IGNORE ) #IMPLIED
  start CDATA #IMPLIED
  userinterface CDATA #IMPLIED
<!--- required for persistence support --->
  persistent  ( FALSE | NEW_CONTEXT | PARENT_CONTEXT ) #IMPLIED
  transacted  ( FALSE | NEW_CONTEXT | PARENT_CONTEXT ) #IMPLIED
  exclusive ( FALSE | TRUE | ACTIVITY | KEY ) #IMPLIED>
<!ELEMENT Userinterface EMPTY>
<!ATTLIST Userinterface name CDATA #REQUIRED
  class CDATA #REQUIRED>
<!ELEMENT Transitions (Transition*)>
<!ELEMENT DomainObjects (DomainObject*)>
<!ELEMENT Transition (Start,((End+,Abend*)|(Abend*|End+)))>
<!ELEMENT Start EMPTY>
<!ATTLIST Start activity CDATA #REQUIRED>
<!ELEMENT End EMPTY>
<!ATTLIST End activity CDATA #IMPLIED>
<!ELEMENT Abend EMPTY>
<!ATTLIST Abend activity CDATA #IMPLIED>
<!ELEMENT DomainObject EMPTY>
<!ATTLIST DomainObject name CDATA #REQUIRED
  class CDATA #REQUIRED
  keep CDATA #IMPLIED>
```

<!--- eof --->

6.4.3. Model: Persistent Objects

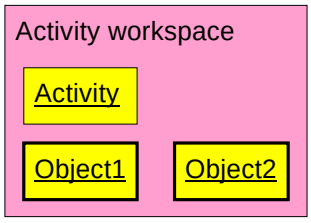
The persistent objects reside in a database and are manipulated by the activities.

6.4.3.1. Overview

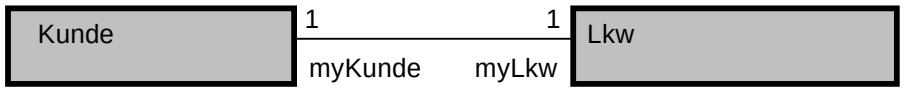
The process of creating persistent objects is complex. However, the complexity results from the extensiveness of the functionality and the various options that the persistence tools offer. What is at first conceived as complexity is later appreciated as far-reaching functionality.

ADD DIAGRAM OF OVERALL PROCESS

6.4.3.2. Create Object Model



(909f)



(906a)

Open Tool Center / OMB

6. From VAJ menu: Select: Workspace / Tools / Mynd environment / Open ToolCenter. The ModelIntegrator appears:

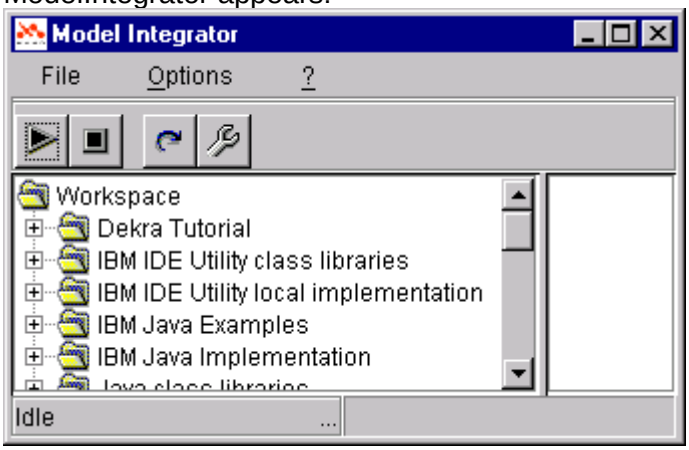
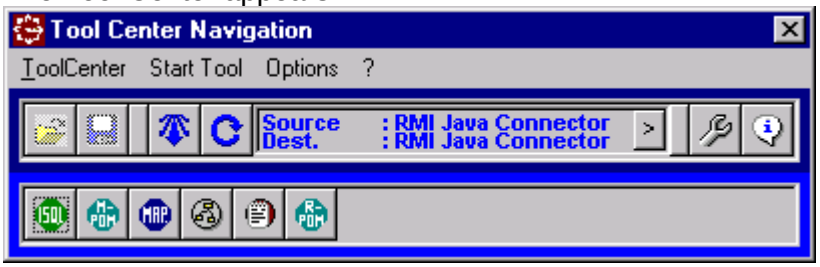


Abbildung 21. Model Integrator dialog (117)

7. Click on the Start icon (arrow).

Note: It may take several minutes for the Tool Center to start.
Note: If an error message appears: Click OK.
The Tool Center appears:



The Object Model Browser (OMB) appears:

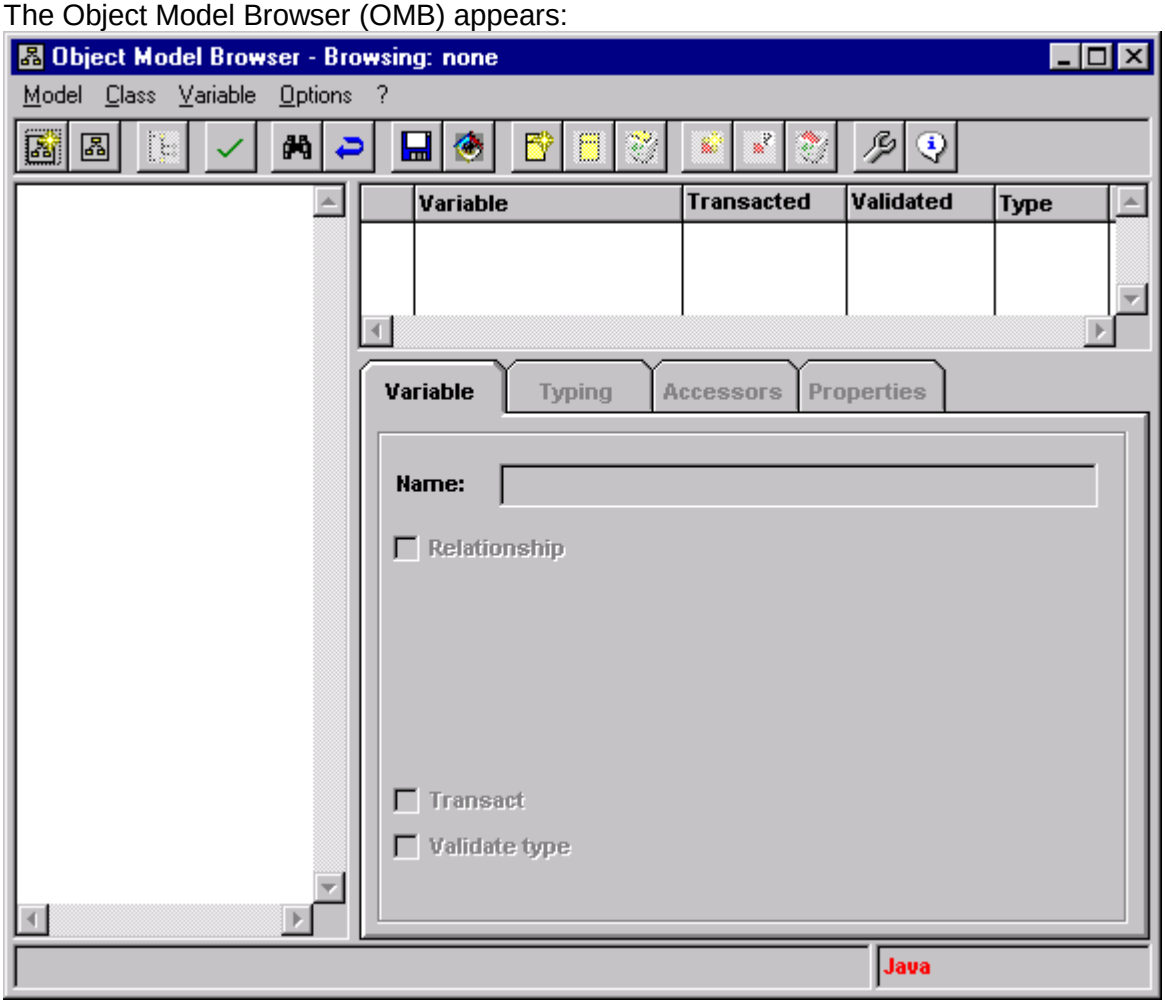
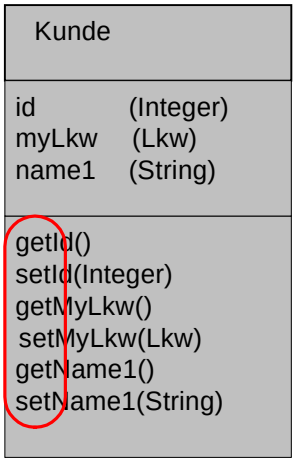


Abbildung 22. ToolCenter and OMB dialogs

Set OMB options



(906b)

8. Select: Options / Preferences. The Options dialog appears.
9. In tab Accessor prefixes: Set Basic read = "basicGet".
10. Set Read = "get".
11. Set Basic write = "basicSet".
12. Set Read = "set".

13. Check checkbox Set all to default.

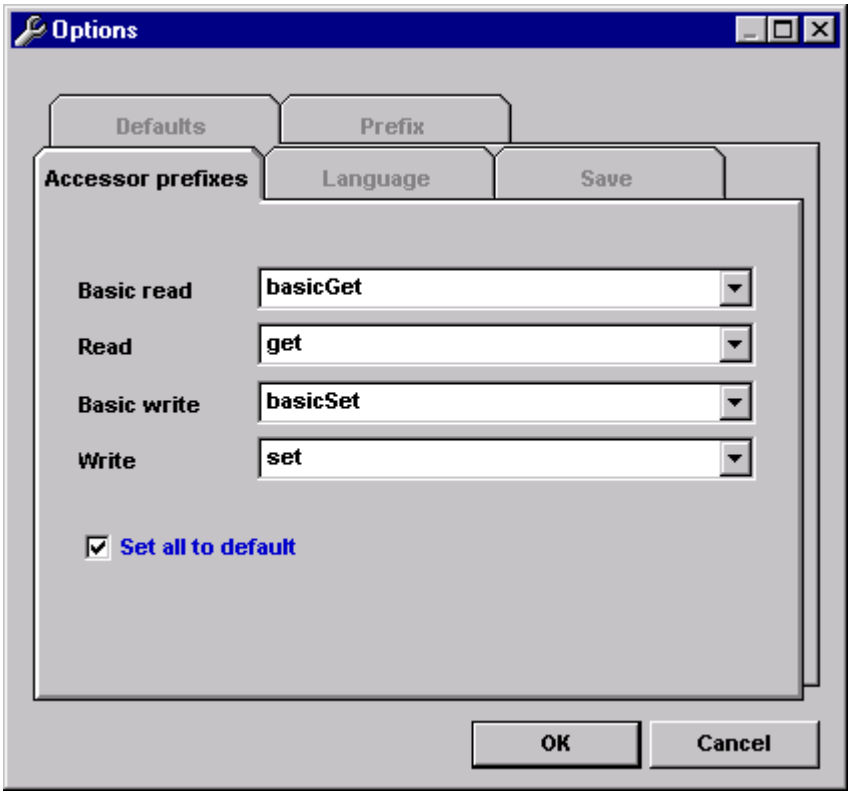


Abbildung 23. OMB options / Accessor prefixes (118)

14. In tab Defaults: For Default packageName: Enter: de.dekra.tutorial.model.
15. Select Use default for new class.
16. Check checkbox Set all to default.

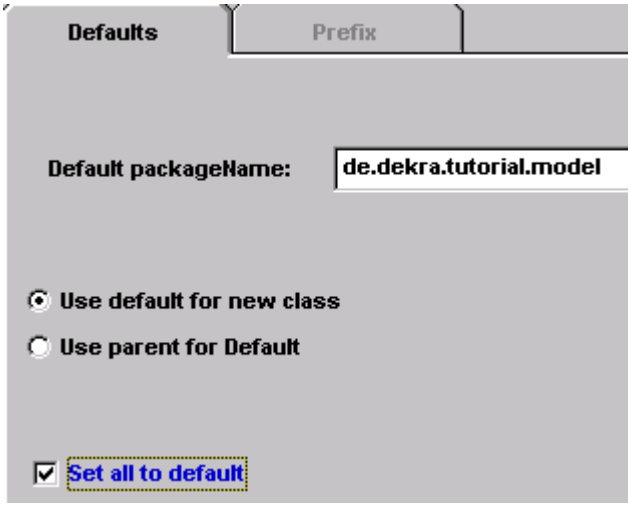


Abbildung 24. OMB default package name (119)

17. Click OK.

Create class Kunde

Kunde (906c)

18. From the OMB main menu: Select: Class / New Class.
The Class specification window appears:

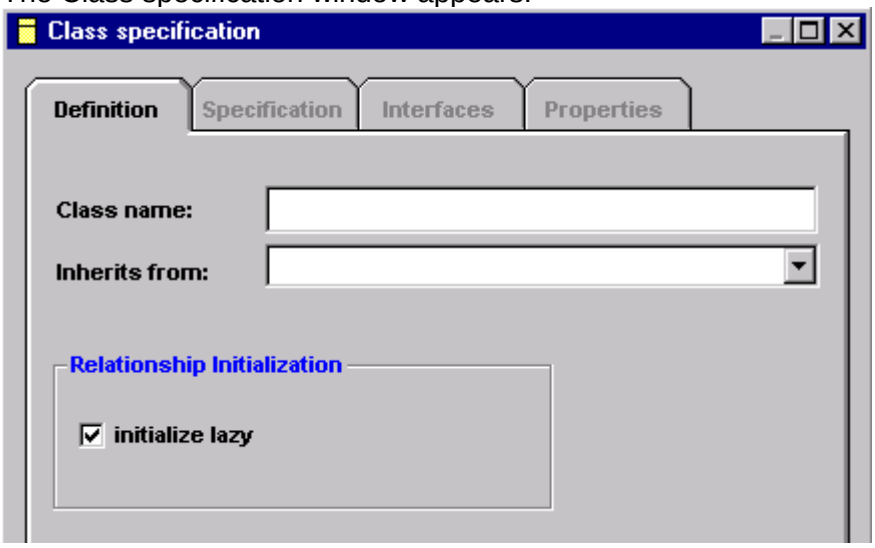


Abbildung 25. Class specification window

19. In tab Definition: For Class name: Enter: Kunde.
20. For Inherits from: Select: PersistentObject.

Abbildung 26. Specification of class Kunde

21. In tab Specification: For Package name: Note that the package default is de.dekra.tutorial.model.

Abbildung 27. Package specification (120)

22. Click OK. Note the Kunde object in OMB:

Abbildung 28. Kunde class in OMB

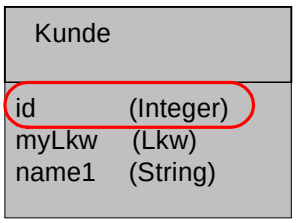
Create class Lkw

(906d)

23. Create class Lkw. **Inherits from Persistent Object.** Package de.dekra.tutoria.model.

Abbildung 29. Lkw class in OMB

Create key variable "id" for Kunde



(906e)

24. Click (select) Kunde.
25. Select: Variable / Add variable. A dialog appears.
26. For the variable name: Enter id.

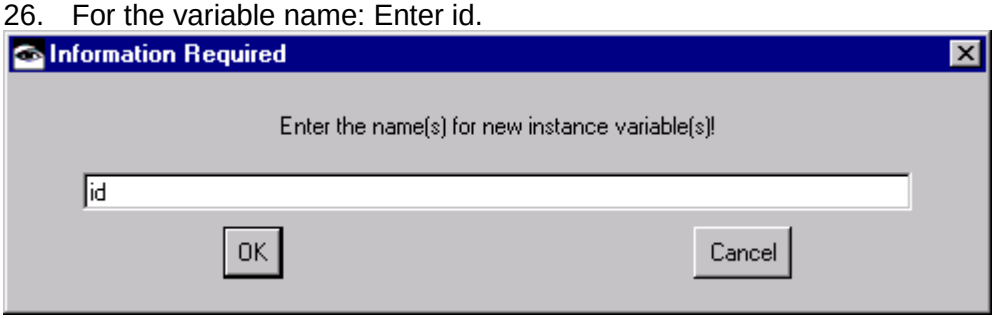


Abbildung 30. Enter name of new instance variables (121)

27. Click OK. Note the variable in OMB:

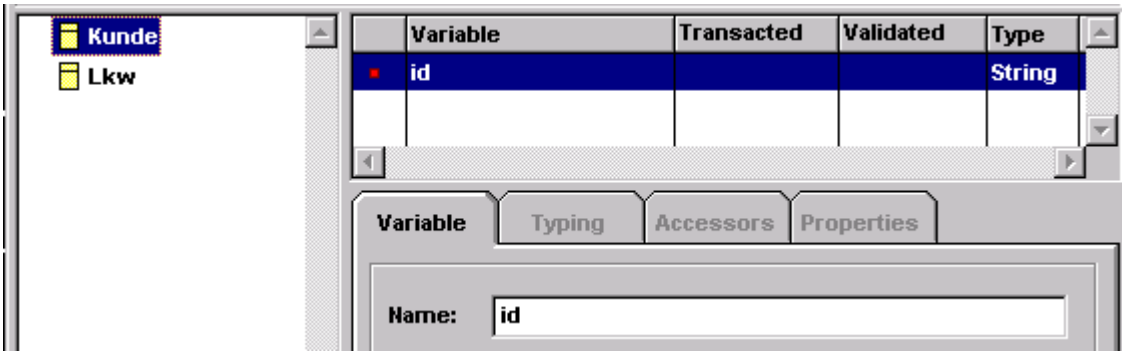


Abbildung 31. Variable id for Kunde in OMB

28. Check checkbox: Transact.

Abbildung 32. Transact checkbox

29. In tab Typing: For Type: Select: **STRING**.

30. Size 20.

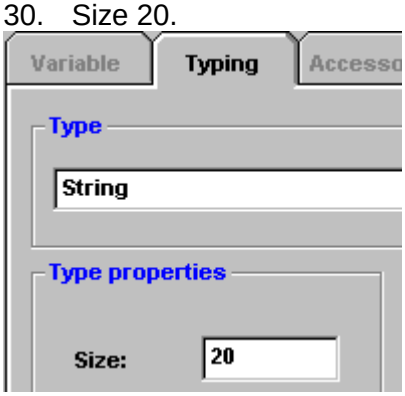


Abbildung 33. Typing for variable id

31. Select tab Accessors. Note the default accessors:

Abbildung 34. Accessors for variable id (124)

32. In tab Properties: For key: Select: true.
33. For persistent: Select: true.

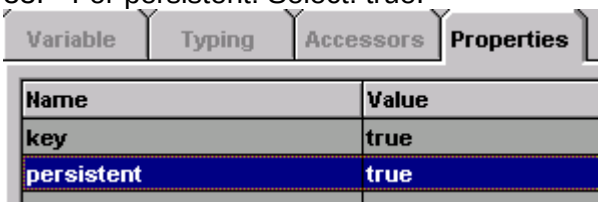
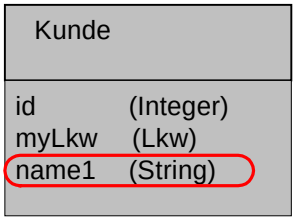


Abbildung 35. Specify id as key and persistent (122)

Create variable name1 for Kunde



(906f)

34. For Kunde: Add variable name1.
35. Transacted.
36. Type: String.
37. Size: 20.
38. Properties: Key: **false**.

39. Properties: Persistent: true.

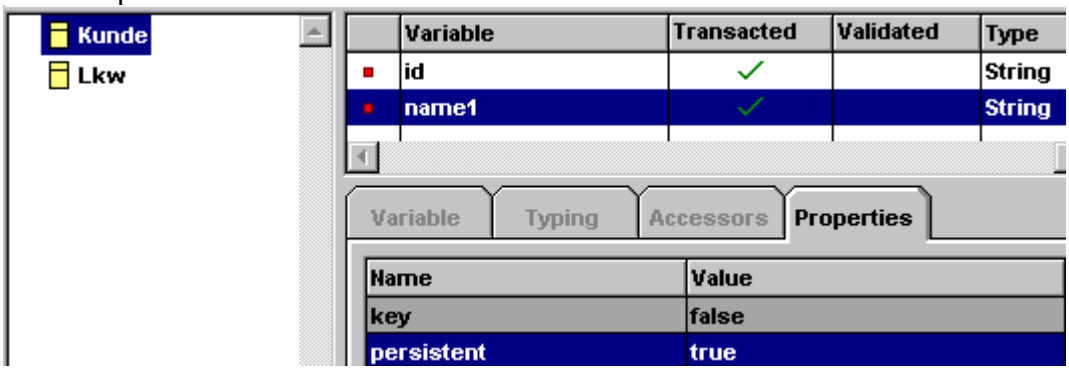
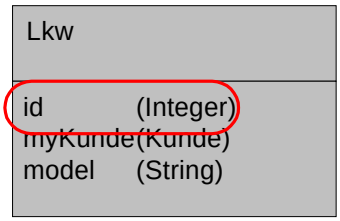


Abbildung 36. Complete Kunde variables (123)

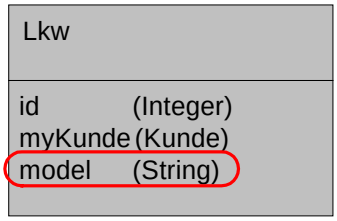
Create key variable id for Lkw



(906g)

40. Click (select) Lkw.
41. Add variable id. Transacted. Type Int. **Key = true**. Persistent = true.

Create variable model for Lkw



(906h)

42. For Lkw: Add variable model. Transacted. Type: String. Size: 20. Key: **false**. Persistent: true.

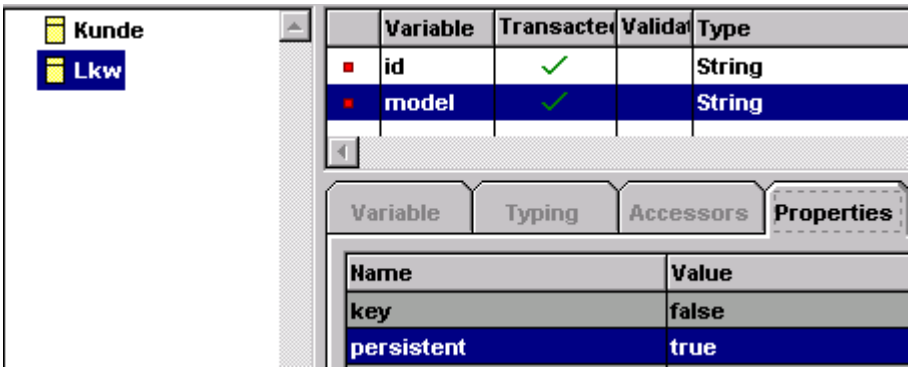
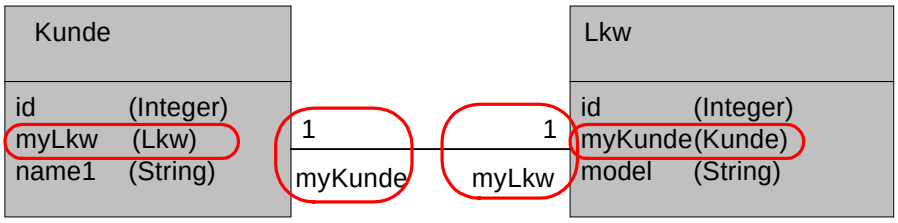


Abbildung 37. Complete Lkw variables (125)

Create 1<->1 relationship between Kunde and Lkw



(906i)

43. For Kunde: Create variable myLkw. Transacted. Persistent = true.
44. For Lkw: Create variable myKunde. Transacted. Persistent = true.
45. For Kunde variable myLkw: In tab Variable: Check checkbox Relationship.

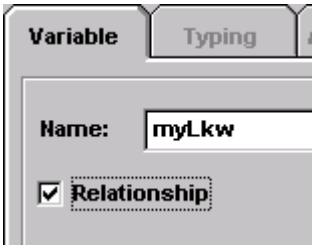
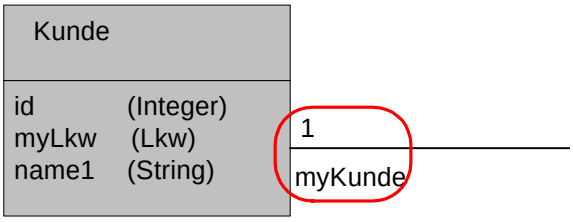
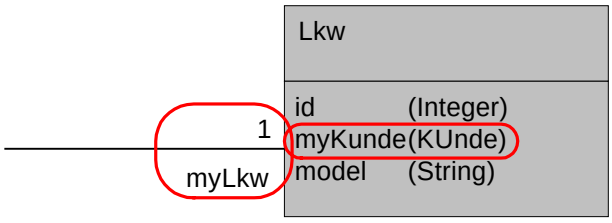


Abbildung 38. Specification of relationship type for variable



(906j)

46. For Source class: For Minimum: Enter 1.
47. For Source class: For Maximum: Enter 1.
48. For Target class: For class: Select Lkw.
49. In tab Typing: Uncheck checkbox Primitive.
50. For Target class: For Variable: Select myKunde.



(906k)

51. For Target class: For Minimum: Select 1. Note: If this button is disabled: Click to change the focus.

52. For Target class: For Maximum: Select 1.

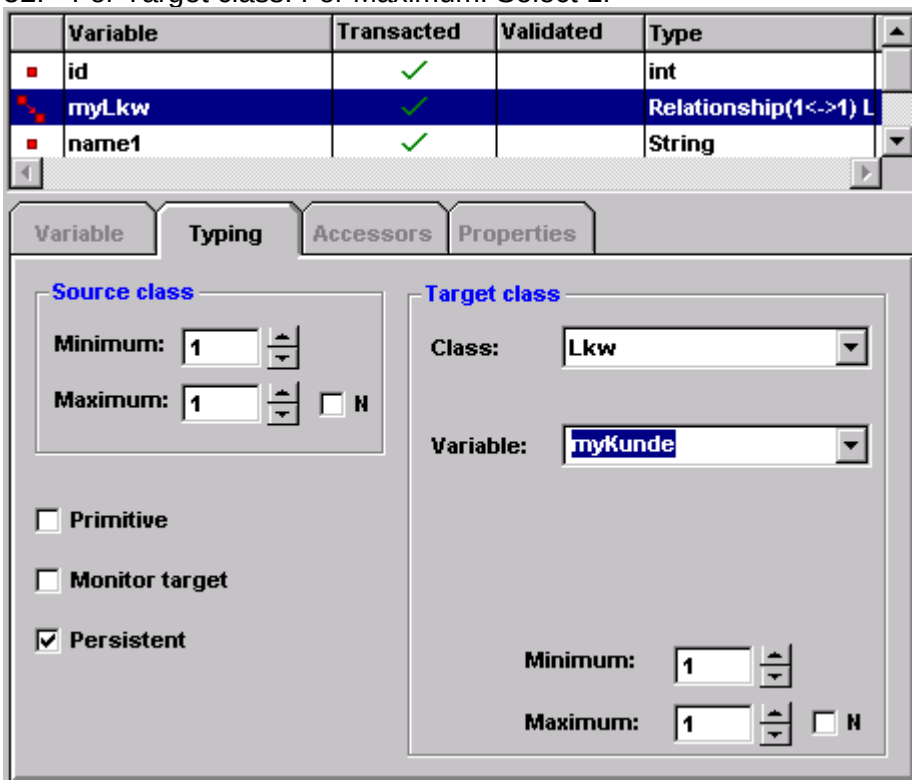


Abbildung 39. Source class maximum/minium settings

53. Select Class Lkw Variable myKunde. Note the settings:

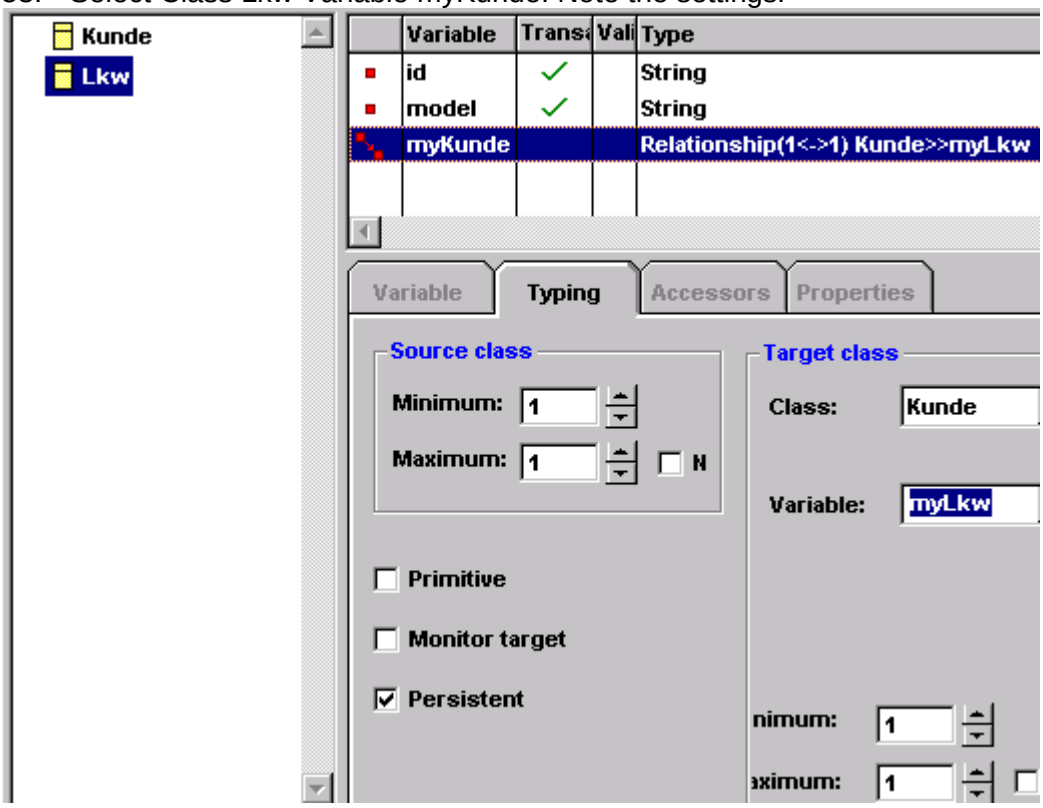


Abbildung 40. Settings for relationship from myKunde (126)

Save the model (.ome file)

54. Select Model / Save model. The Save as dialog appears.
55. [VA drive : directory] \ide\project_resources\Dekra Tutorial**ZZZObjectModel1.ome**.

6.4.3.3. Create VA object classes

56. Select Model / Save to VA. Note: This may reqire several minutes.

57. In VA: Note the created classes in Project Dekra Tutorial / package de.dekra.tutorial.model:

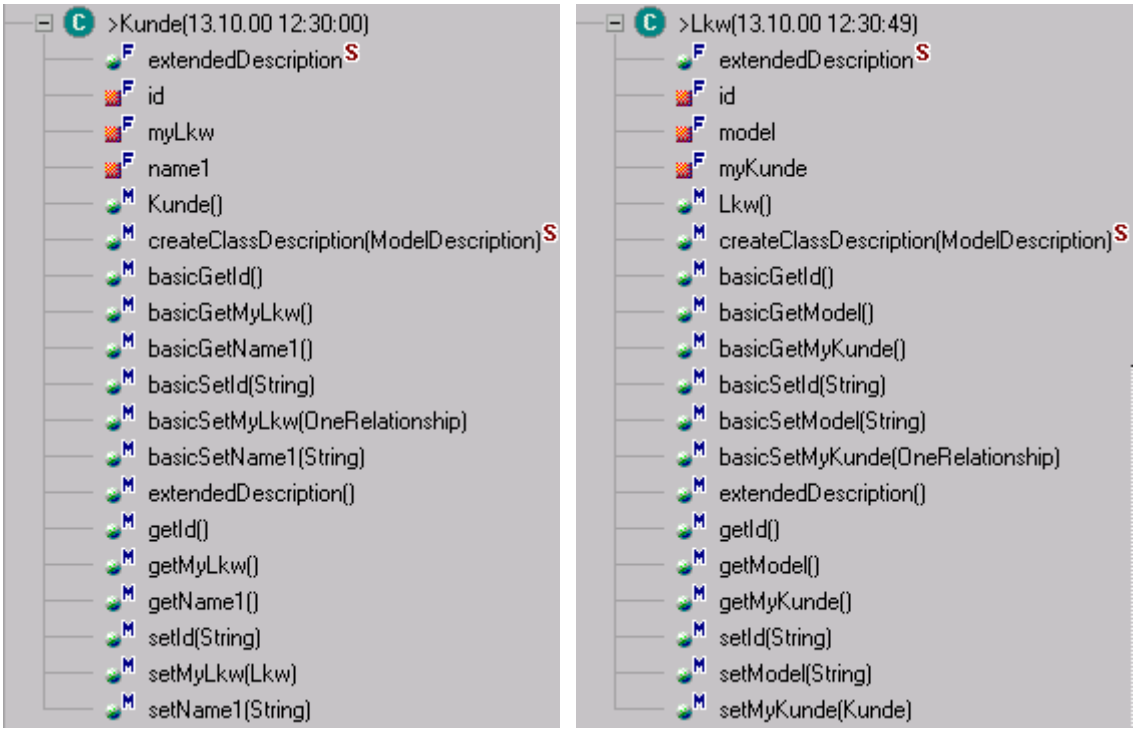
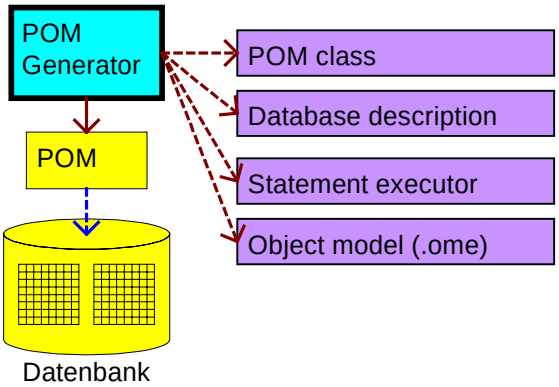


Abbildung 41. Imported model classes Kunde, Lkw in VA (127, 128)

Analysis: Access to Lkw object from Kunde object via OBF accessors

Describe in detail the accessor structure.

6.4.3.4. Create POM Generator



(909q)

Open the POM Generator

The POM generator can be started in 2 ways:

- Direct from the OMB. Model / Tools / Model->POM Generator. In this case the .ome file must not be specified (the object model must not be loaded).
- From the Tool Center: Start Tool / Model->POM Generator. You will be prompted to specify the .ome file.

58. In the OMB. Model / Tools / Model->POM Generator. The POM Generator appears:

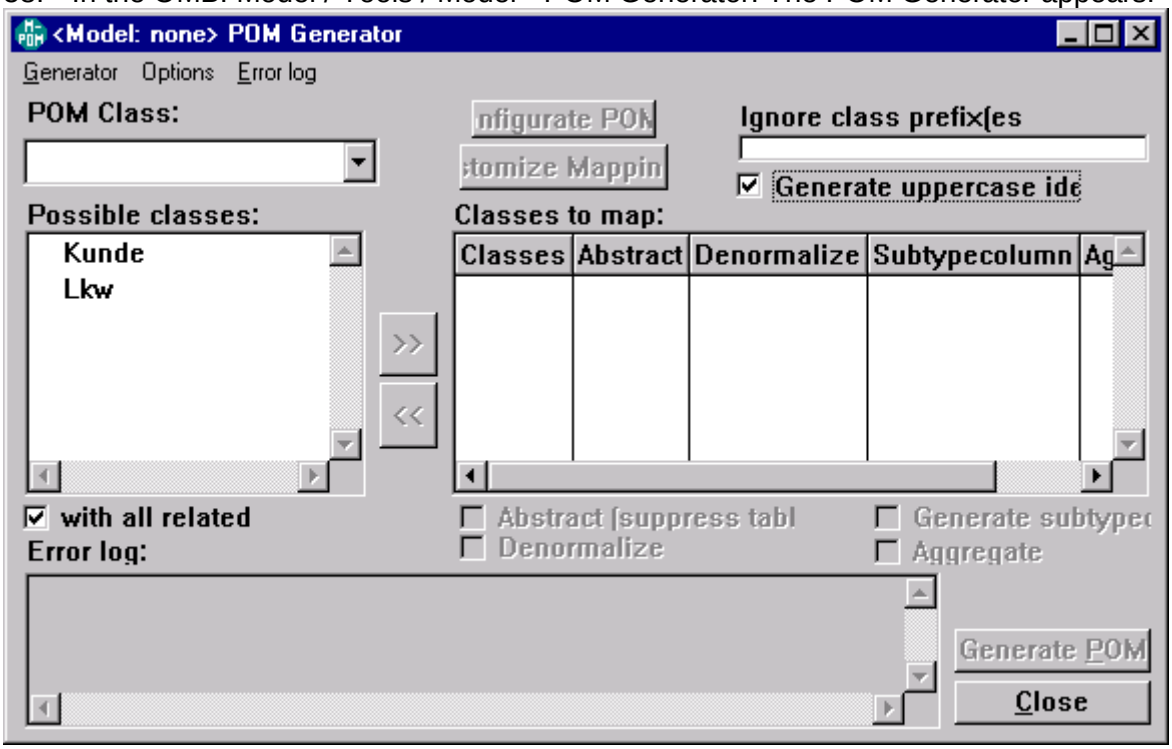
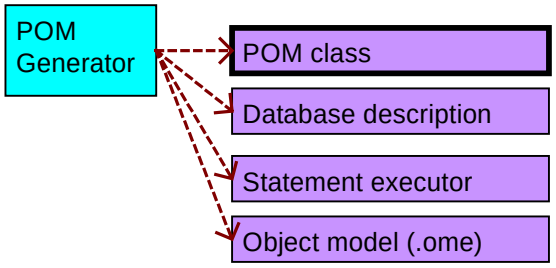


Abbildung 42. POM Generator (129)

Select POM Generator class



(909r)

59. From the drop-down list POM Class: Select: MicFwPassiveToolPomRdb:

Abbildung 43. Select POM class

Configure POM

60. Click the button: Configurate POM.... The configuration dialog appears:

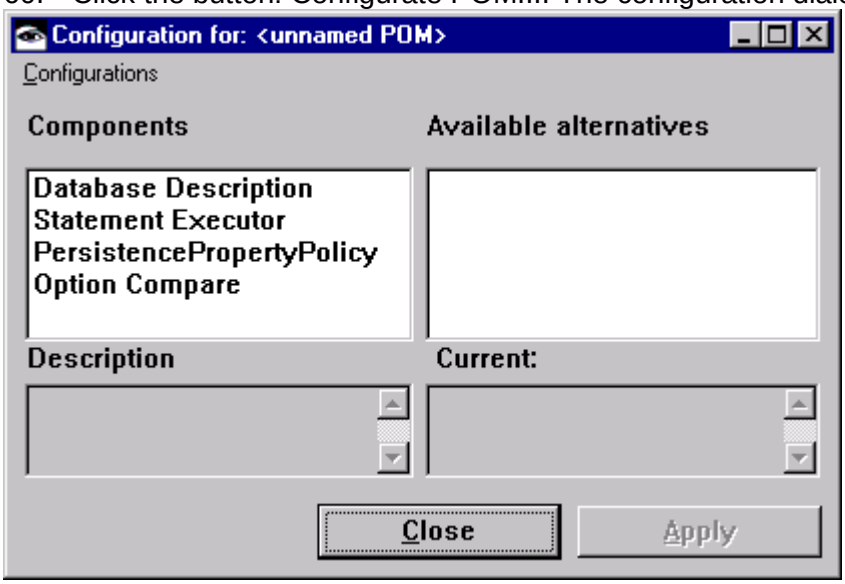
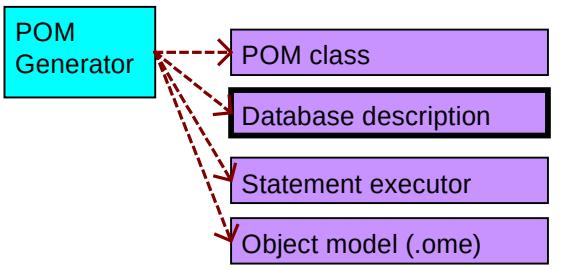


Abbildung 44. Configuration POM dialog

Configure POM: Database description



(909s)

61. For Database Description: Select com.mynd.dse.ps.rdb.jdbc.DatabaseDescriptionJdbcOdbc:

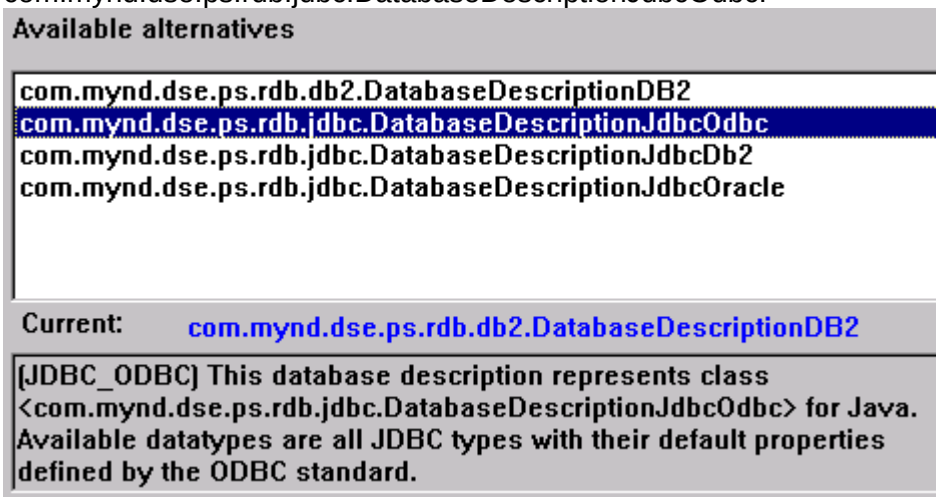
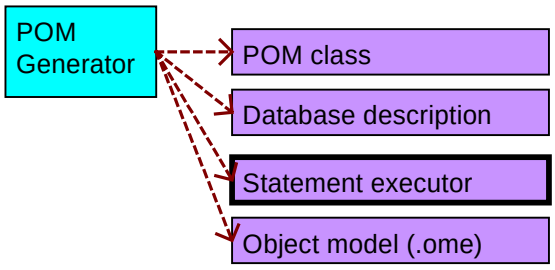


Abbildung 45. Database description (130)

62. Click Apply.

Configure POM: Statement executor



(909t)

63. For Statement Executor: Select com.mynd.dse.ps.rdb.jdbc.StatementExecutorJdbc:

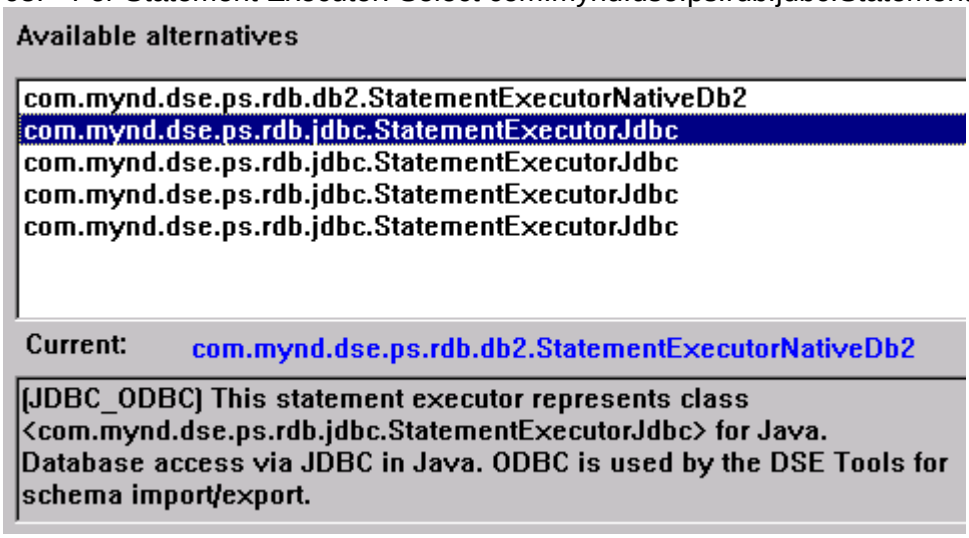


Abbildung 46. Statement executor (131)

64. Click Apply.

Configure POM: Persistent Property Policy / Option compare

65. For Persistent Property Policy: The default is OK.

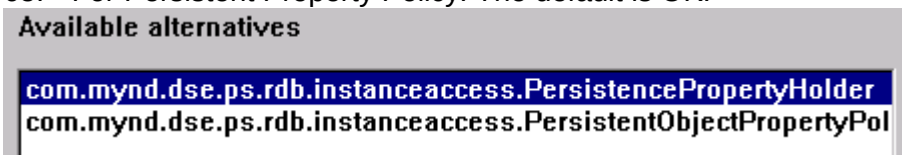
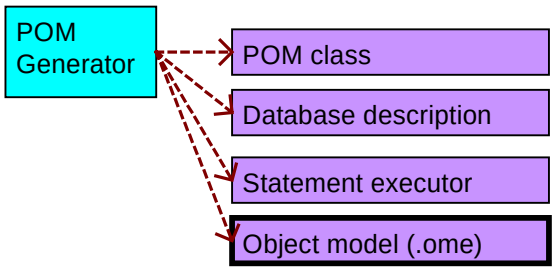


Abbildung 47. Persistent Property Policy

66. For Option Compare: Select OFF (default).
67. Click Close.

Select classes



(909u)

68. In Possible Classes: Select Kunde.

69. Click >>. Note that the Lkw class is also copied, since it has a relationship to the Kunde class and checkbox "with all related" is checked.

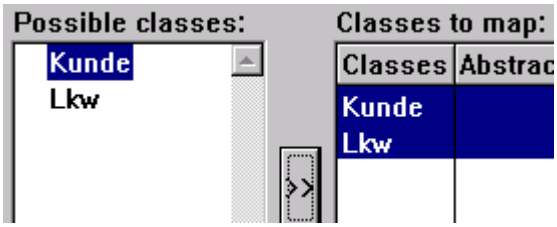
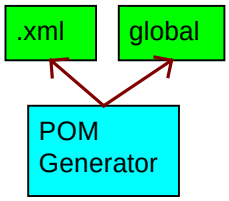


Abbildung 48. Select classes in POM Generator (132)

Save the POM Generator (as file and as globally)

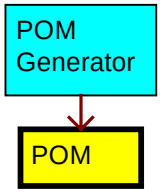


(909v)

70. Select: Generator / Export to file: Save the pom generator as **ZZZPOMG.xml**.

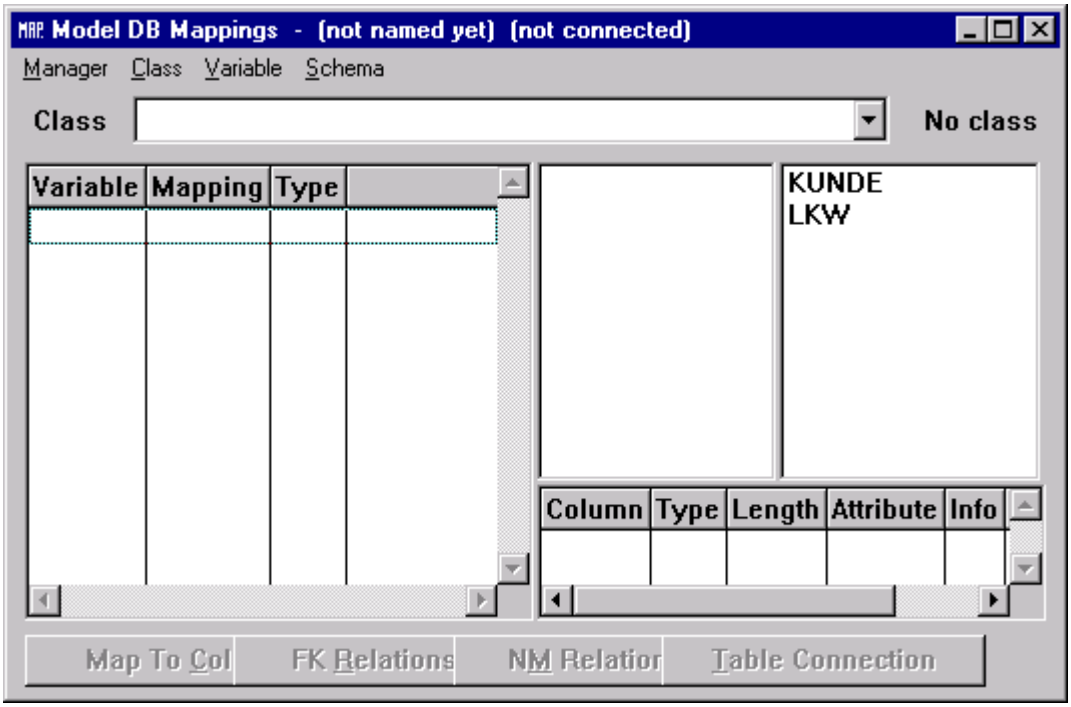
71. Select: Generator / Store globally as.... A dialog appears. Save the pom generator globally as **ZZZPOMG**.

6.4.3.5. Create POM



(909w)

72. Click on the button Generate POM.



Note that the POM has not been named and that the POM is not connected to a database.

Abbildung 49. Model DB Mappings

View the POM<->database mappings

73. From the drop-down list Class: Select Kunde. Note the mappings for the variables.

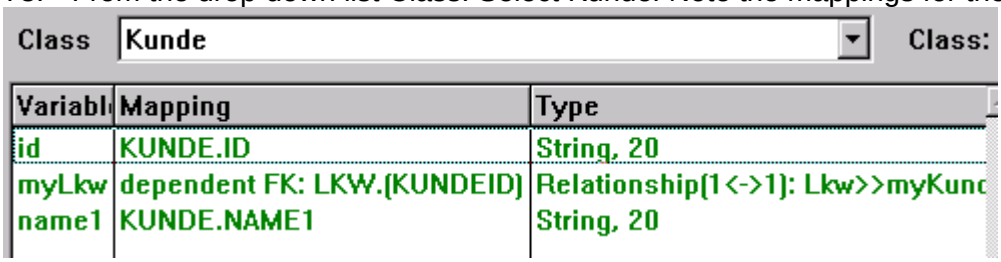


Abbildung 50. Mappings class Kunde (133)

74. In the right window: Click on table KUNDE. Note the mappings for the KUNDE table in the database.

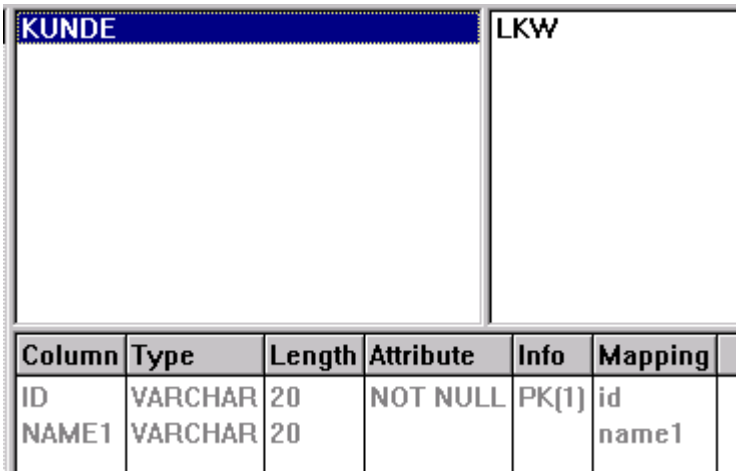


Abbildung 51. Mappings table KUNDE (134)

75. In the most-right window: Click on table LKW. Note the mappings for the LKW table in the database.

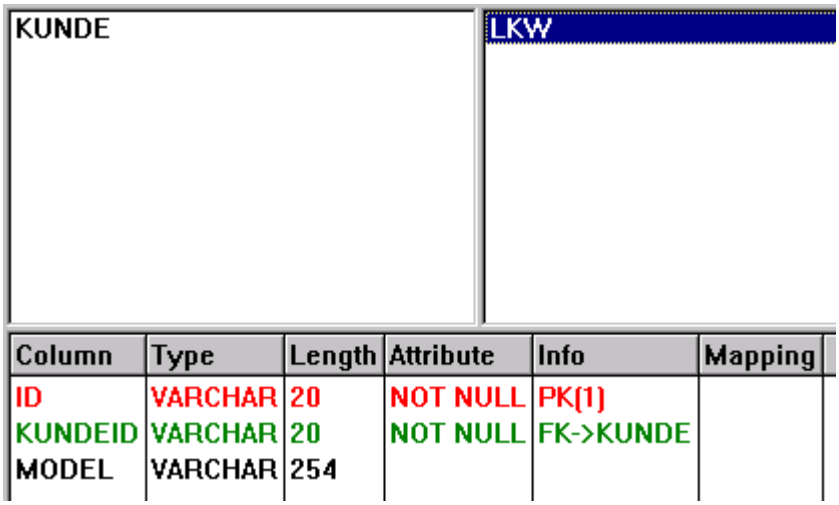


Abbildung 52. Mappings table LKW (135)

76. From the drop-down list Class: Select Lkw. Note the mappings for the variables.

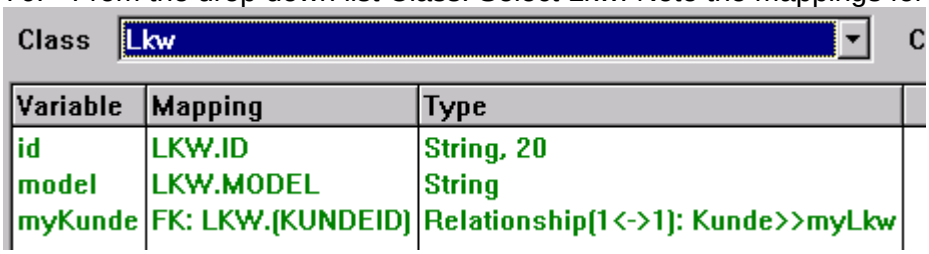


Abbildung 53. Mappings class Lkw (136)

77. In the right window: Click on table LKW. Note the mappings for the LKW table in the database.

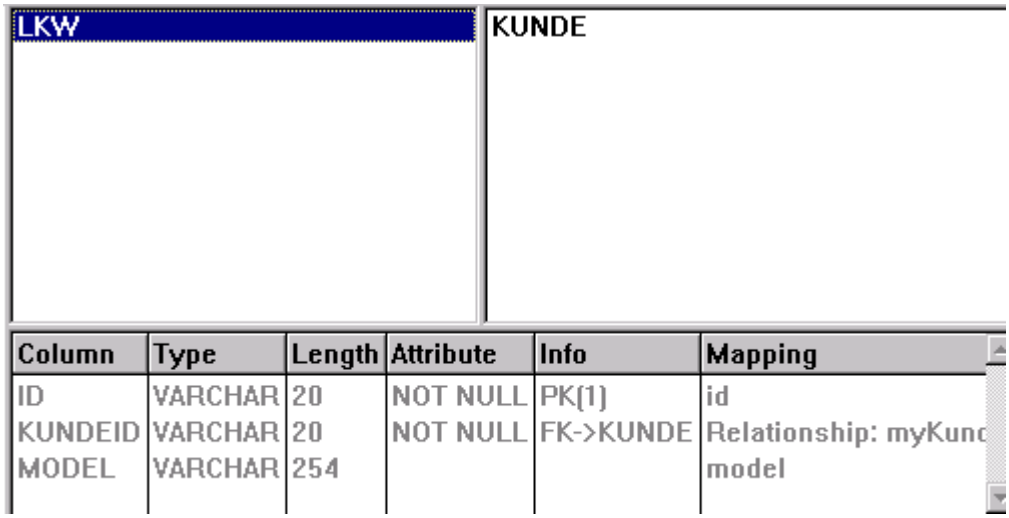


Abbildung 54. Mappings table LKW (137)

78. In the most-right window: Click on table KUNDE. Note the mappings for the KUNDE table in the database.

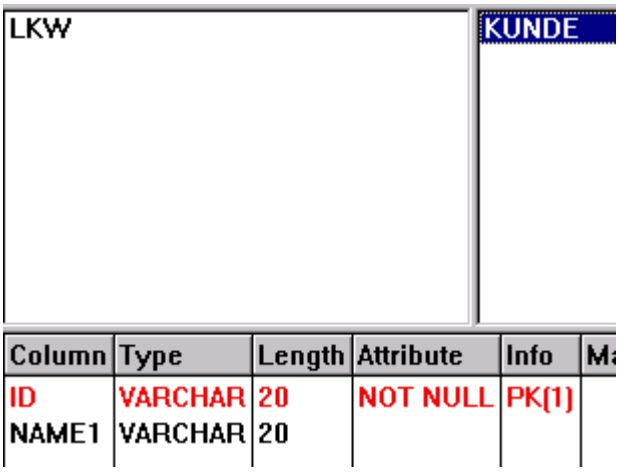
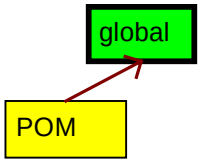


Abbildung 55. Mappings table KUNDE (138)

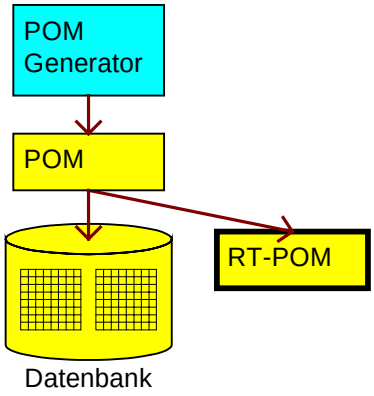
Store POM in memory



(909x)

79. Choose Manager / In memory / Store POM in memory as....
80. Enter **ZZZPOM**.
81. Click OK. Note the name "ZZZPOM" at the top of the dialog.

6.4.3.6. Create Runtime POM



(909y)

The RT-POM (runtime POM) is the runtime interface between the workspace objects and the database.

Create the Java source code

82. In Model DB Mappings: Select: Manager / Open runtime POM Generator.
83. For Package: Enter de.dekra.tutorial.model.
84. For Initializer class: Enter ZZZPOMInitializer.

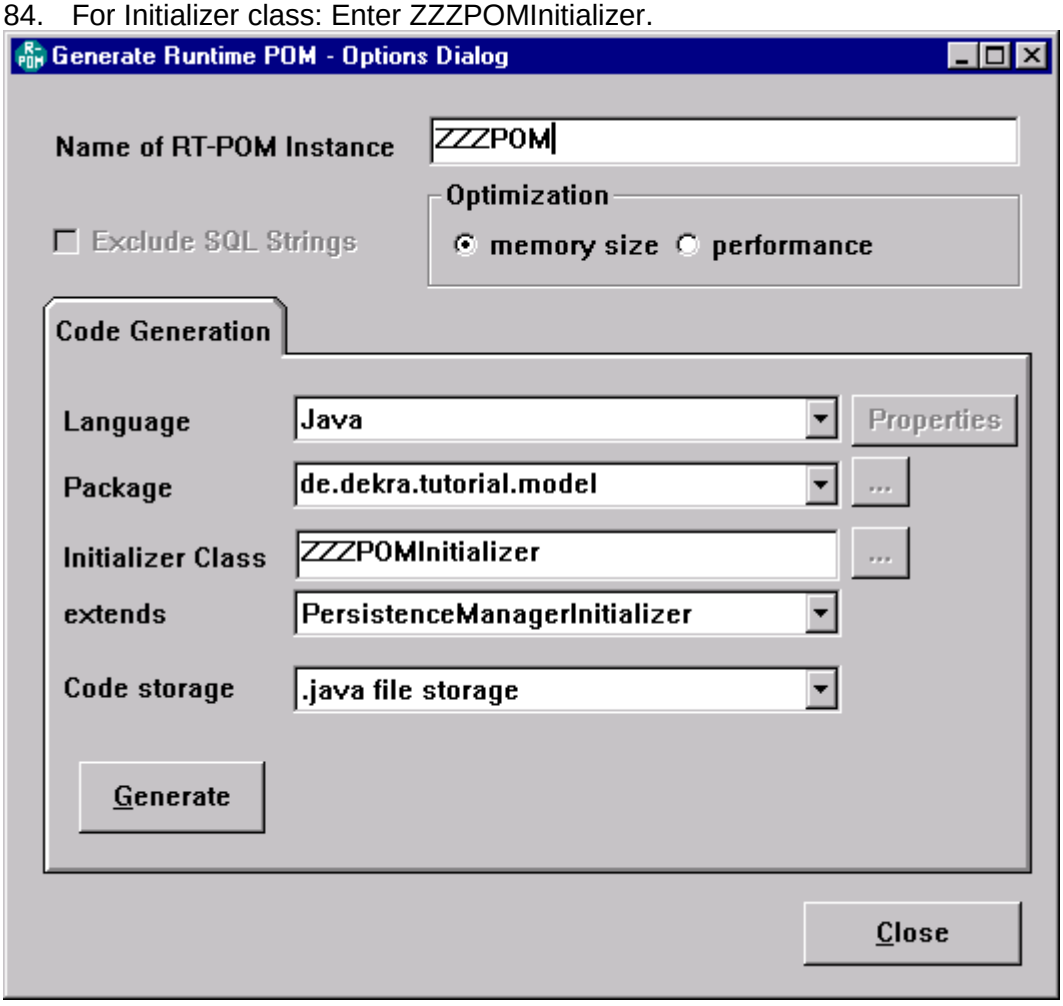


Abbildung 56. Enter RTPOM info (139)

85. Click Generate. The dialog Choose Output file appears.

86. Save the RTPOM as [VA drive : directory] \ide\project_resources\Dekra Tutorial**ZZZPOMInitializer.java**. Note the information dialog that appears:

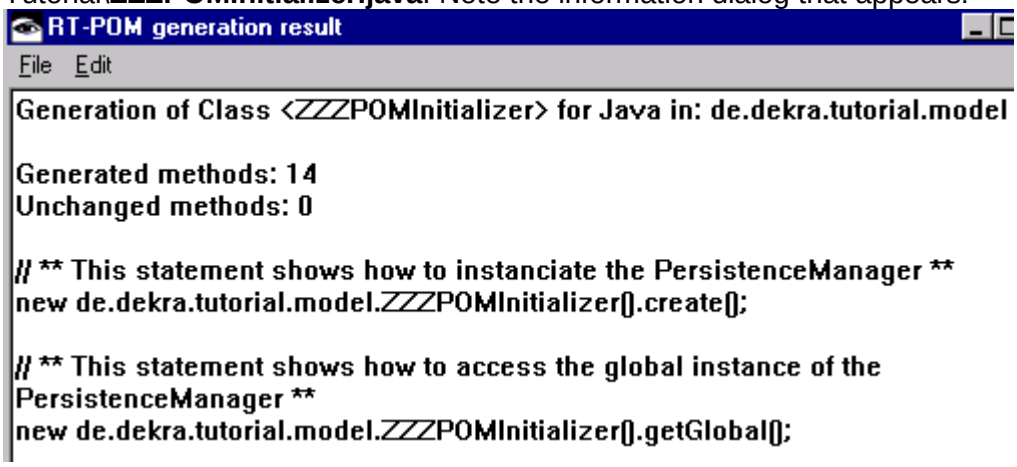


Abbildung 57. RTPOM generation result (140)

Import the source code into VA

87. In VA: Right click on project Dekra Tutorial package de.dekra.tutorial.model.
88. Select Import. The Import Smart Guide appears.
89. Select import source Directory.
90. Click Next. The Import from a directory dialog appears.
91. For directory: Select [VA drive : directory] ide\project_resources\Dekra Tutorial\.
92. For java files: Select **ZZZPOMInitializer.java**.

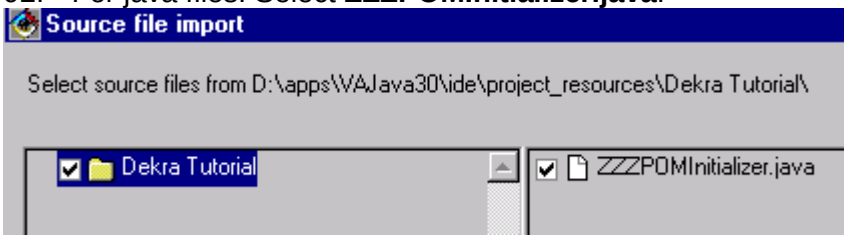


Abbildung 58. Source file import (141)

93. Click OK.

94. Uncheck checkbox resource.

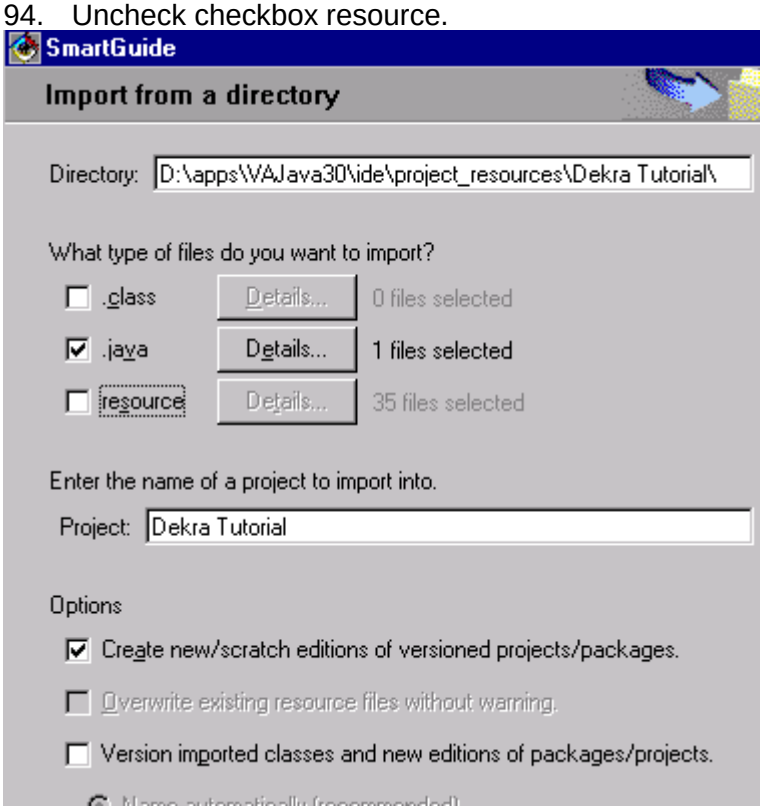


Abbildung 59. SmartGuide for file import (142)

95. Click Finish. The class is imported to project Dekra Tutorial / package de.dekra.tutorial.model.

6.4.3.7. Modify POM-releated classes

Modify de.dekra.tutorial.model.PomConstants

96. Specify the constants required for database access:

- Database URL
- Driver
- Username/password

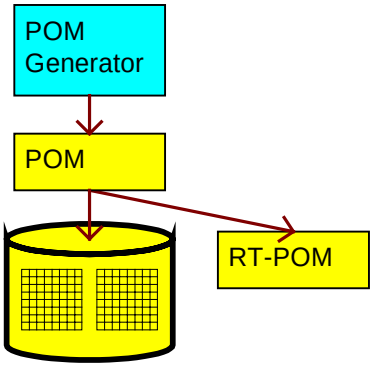
```
public interface PomConstants {
  public static final String DB_URL = "jdbc:odbc:ZZZDB";
  public static final String DB_USER = "SYSTEM";
  public static final String DB_PASSWORD = "SYSTEM";
  public static final String DB_DRIVER = "sun.jdbc.odbc.JdbcOdbcDriver";
}
```

Modify de.dekra.tutorial.model.PomHolder

97. Modify method getPom() for ZZZPOMInitializer().

```
  public static PersistenceManagerRdb getPom() {
    if ( pom == null ) {
      pom = new ZZZPOMInitializer().getGlobal();
    }
    return pom;
  }
```

6.4.3.8. Create database



(909z)

Create database

98. Open Oracle8i lite navigator.

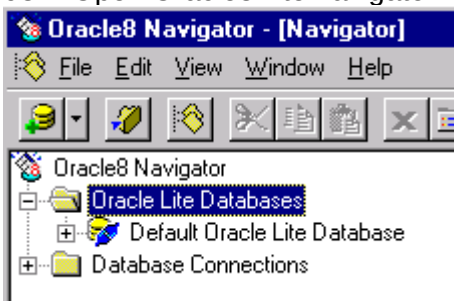


Abbildung 60. Oracle8i lite Navigator main dialog (143)

99. Right-click on "Oracle Lite databases".
100. Select New. The "New Oracle Lite Database Properties" dialog appears.
101. For Name: Enter **ZZZDB**.

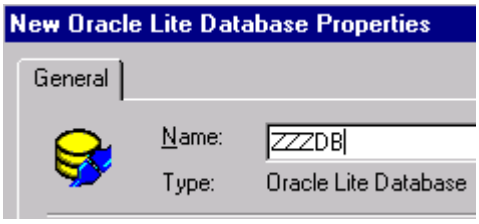


Abbildung 61. Enter database name (144)

102. Click OK.

Abbildung 62. Database in Oracle list (145)

103. Right-click on ZZZDB in the list.
104. Select Properties. Note the database location.

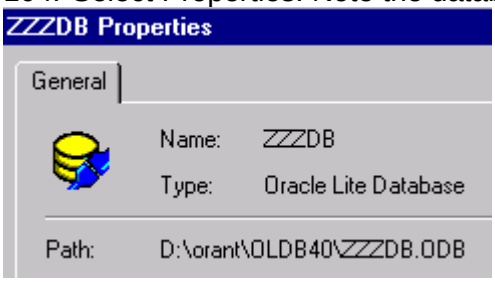


Abbildung 63. Database properties (146)

Add to ODBC list

105. Start / Einstellungen / Systemsteuerung.
106. Double-click on ODBC Datenquellen.

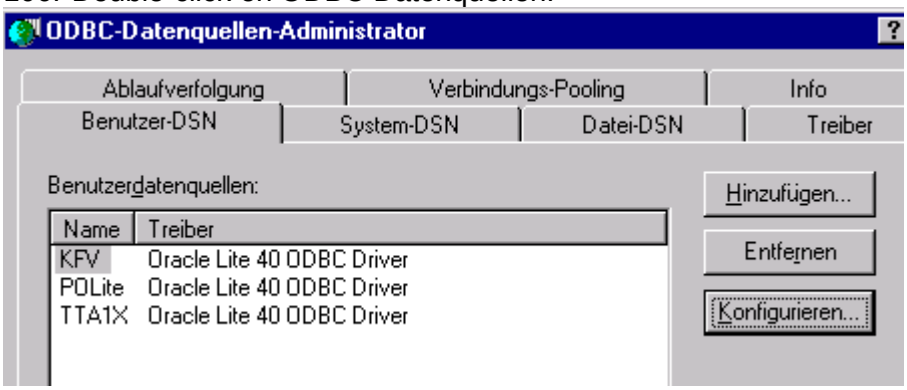


Abbildung 64. ODBC Datenquellen Administrator

107. Click on Hinzufügen.
108. Double-click on Oracle LIte 40 ODBC Driver.
109. Enter the info as shown below.

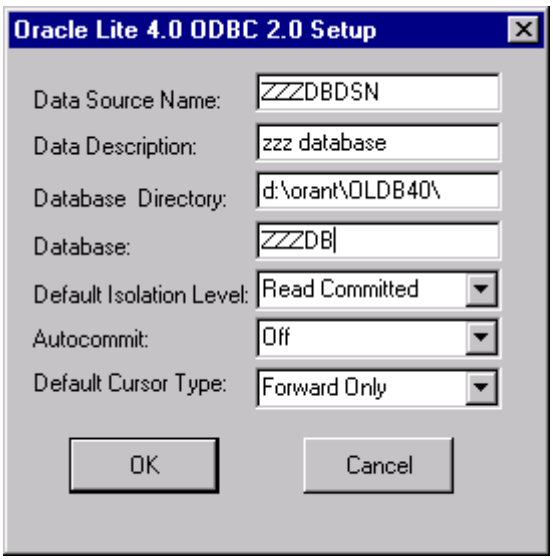
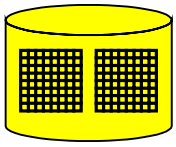


Abbildung 65. Oracle ODBC setup (149)

110. Click OK.
111. Click OK (close ODBC-Datenquellen-Administrator).

6.4.3.9. Create database tables



(909za)

Create DDL

112. In the Model DB Mappings tool: Select Schema / Export. The following dialog appears:

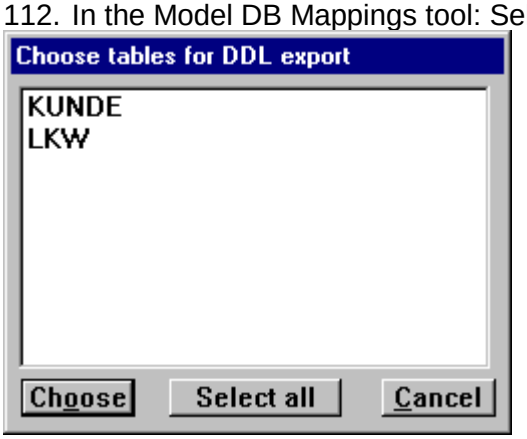


Abbildung 66. Choose tables for DDL export

113. Click Select all.

114. Click Choose. The Schema Export dialog appears: **NOTE: SELECT ODBC JDBC (not Oracle 7.0 native) (this picture is incorrect).**

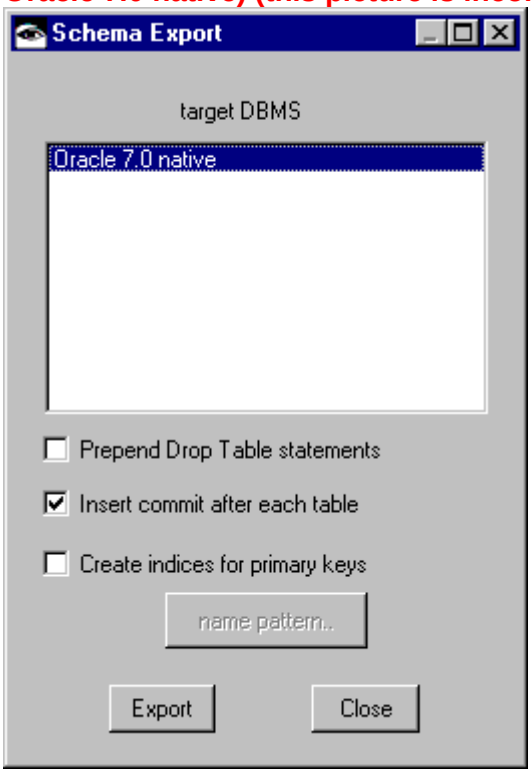


Abbildung 67. Schema export dialog (ERROR)

115. Select ODBC JDBC.

116. Click Export. The following dialog with the DDL text appears. NOTE: this pane was created with Oracle 7.0 Native:

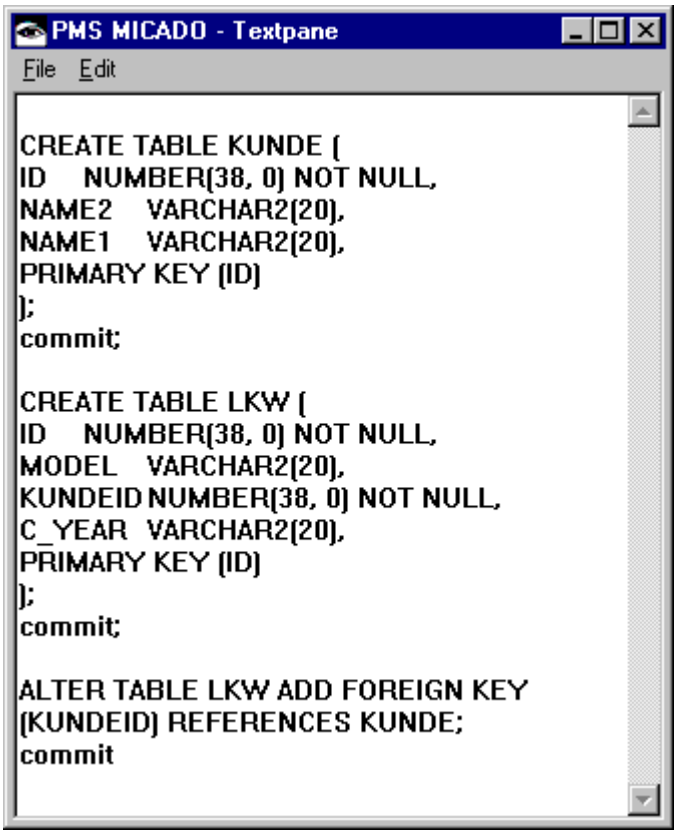


Abbildung 68. Exported DDL text

Create tables (execute DDL against database)

117. From ToolCenter: Select: Start tool / Interactive SQL. The ISQL dialog appears:

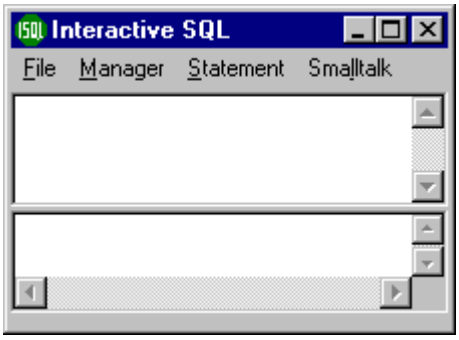


Abbildung 69. ISQL dialog

118. Copy the text (using copy and paste) from the textpane to the top window of ISQL.

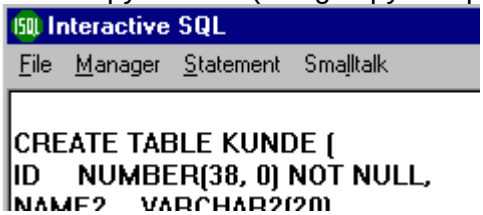


Abbildung 70. ISQL dialog with copied SQL text

119. Select: Manager / Choose. The dialog Choose Persistence Manager appears:

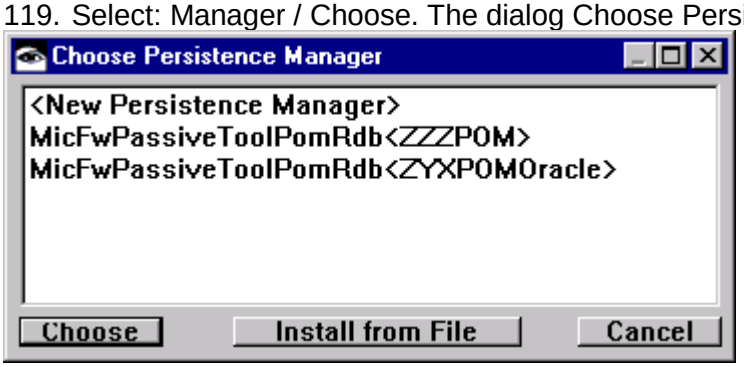


Abbildung 71. Choose Persistence Manager dialog (147)

120. Click on ZZZPOM.

121. Click Choose. Note the top of the ISQL:

Abbildung 72. ISQL dialog header (not connected) (148)

122. Select Manager / Connect. The dialog Datenquelle auswählen appears:

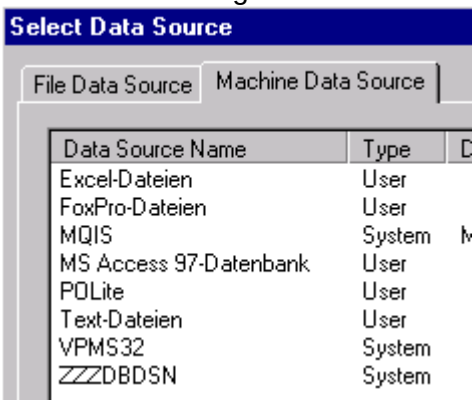


Abbildung 73. Datenquelle auswählen (150)

123. Click on ZZZDBDSN.

124. Click OK. Note the ISQL dialog:

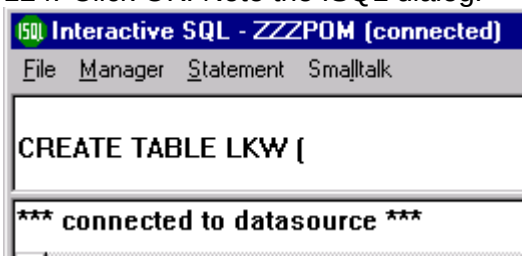


Abbildung 74. SQL in ISQL (151)

125. Select: Statement / Execute once.

Abbildung 75. Ready and commit statements (ISQL)

View tables in Oracle8i navigator

126. In the Navigator: Note the new tables:

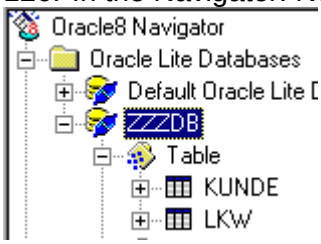


Abbildung 76. New tables in database ZZZDB (152)

127. Right-click on table KUNDE.

128. Select Open. Note the contents of the table:

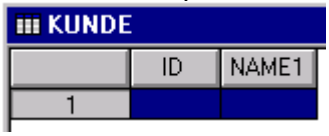


Abbildung 77.Empty table KUNDE (153)

129. Note the contents of LKW:

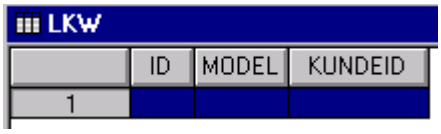


Abbildung 78. Empty table LKW (154)

Add data to tables

130. Add the following data to the tables:

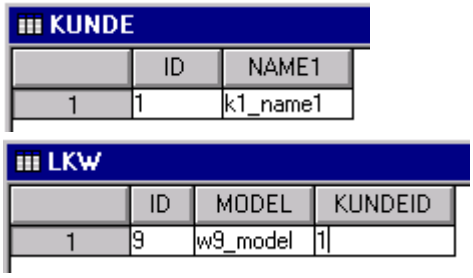
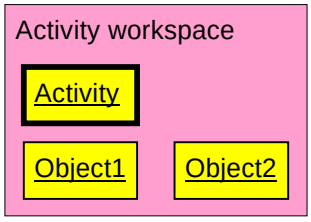


Abbildung 79. Data for tables (155, 156)

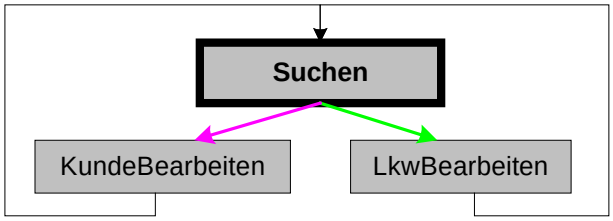
6.4.4. Model: Activities (mit fachlicher Logik)



(909zd)

At this point you will create the activity classes.

6.4.4.1. ZZZActivitySuchen.java



(913a)

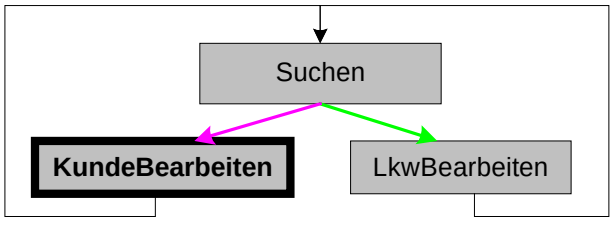
131. Create the following method:

```
package de.dekra.tutorial.model;
import com.mynd.dse.ps.searchagent.*;
import com.mynd.dse.ccb.*;
import com.mynd.dse.base.*;
import java.awt.event.*;
import java.util.*;
// ZZZActivitySuchen
public class ZZZActivitySuchen extends ServiceActivity {
// #3/5 Listeners
// SearchListener for view button Search.
// starts search for Kunde/Lkw in database
   class SearchListener implements ActionListener {
     public void actionPerformed(ActionEvent e) {
       System.out.println(e.toString());
       searchCommand();
     }
   }
// EditListener for view button Edit.
// ends activity
   class EditListener implements ActionListener {
     public void actionPerformed(ActionEvent e) {
       System.out.println(e.toString());
       done();
     }
   }
 public ZZZActivitySuchen() {
   super();
 }
// get object from SLOTKundeInfo (and cast as ZZZKundeInfo
    object)
 protected ZZZKundeInfo getZZZKundeInfo() {
   return (ZZZKundeInfo)get("SLOTKundeInfo");
 }
// get object from SLOTLkwInfo (and cast as ZZZLkwInfo object)
 protected ZZZLkwInfo getZZZLkwInfo() {
   return (ZZZLkwInfo)get("SLOTLkwInfo");
 }
// #1/5 canStart
// always true
 public boolean canStart() {
     return true;
 }
// #2/5 doStart
// addCommandRules, put Info objects in SLOTs
 public void doStart() {
// addCommandRules for View buttons Edit and Search
```

```
    addCommandRule("CREdit", new EditListener());
    addCommandRule("CRSearch", new SearchListener());
// put ZZZKundeInfo object in slot SLOTKundeInfo
    if (get("SLOTKundeInfo") == null) {
      put("SLOTKundeInfo", new ZZZKundeInfo());
    }
// put ZZZLkwInfo object in slot SLOTLkwInfo
    if (get("SLOTLkwInfo") == null) {
      put("SLOTLkwInfo", new ZZZLkwInfo());
    }
  }
// #3/5 Search button clicked: Search for Kunde/Lkw in database
     and put in slots
  protected void searchCommand() {
    Lkw lkw = null;
    Kunde kunde = null;
    ZZZLkwInfo lkwInfo = getZZZLkwInfo();
    ZZZKundeInfo kundeInfo = getZZZKundeInfo();
// if ZZZKundeInfo id set: search for kunde in database
    if (kundeInfo.getId().length() > 0) {
      kunde = searchKunde(kundeInfo.getId());
// if kunde found in database:
//    * put kunde from db in slot
//    * put kundes lkw from db in slot
//    * set the id for ZZZLkwInfo
      if (kunde != null) {
        put("SLOTKunde", kunde);
        put("SLOTLkw", kunde.getMyLkw());
        lkwInfo.setId(kunde.getMyLkw().getId());
      }
    }
// if ZZZLkwInfo id set: search for lkw in database
    if (lkwInfo.getId().length() > 0) {
      lkw = searchLkw(lkwInfo.getId());
// if lkw found in database:
//    * put lkw from db in slot
//    * put lkws kunde from db in slot
//    * set the id for ZZZKundeInfo
      if (lkw != null) {
        put("SLOTLkw", lkw);
        put("SLOTKunde", lkw.getMyKunde());
        kundeInfo.setId(lkw.getMyKunde().getId());
      }
    }
  }
// #3.1/5 Liefert den Kunden mit id = key
  protected Kunde searchKunde(String key) {
    Kunde kunde = null;
    try {
      SearchAgent agent = getSearchAgent();
      agent.setRootClass(Kunde.class);
      agent.addAttribute("id", OperatorType.EQUALS, key);
      List result = agent.get();
      if (result.size() > 0)
        kunde = (Kunde) result.get(0);
    } catch (Exception e) {
      e.printStackTrace();
```

```
    }
    return kunde;
  }
  // #3.2/5 Liefert das Lkw mit id = key
  protected Lkw searchLkw(String key) {
    Lkw lkw = null;
    try {
      SearchAgent agent = getSearchAgent();
      agent.setRootClass(Lkw.class);
      agent.addAttribute("id", OperatorType.EQUALS, key);
      List result = agent.get();
      if (result.size() > 0)
        lkw = (Lkw) result.get(0);
    } catch (Exception e) {
      e.printStackTrace();
    }
    return lkw;
  }
  // #4/5 canComplete
  // true if either Kunde or Lkw in slots exists in database
  public boolean canComplete() {
    if (searchKunde(getZZZKundeInfo().getId())!=null)
      return true;
    if (searchLkw(getZZZLkwInfo().getId())!=null)
      return true;
    return false;
  }
  // #5/5 doComplete
  // Send Kunde/Lkw info to console
  public void doComplete() {
    if (getZZZKundeInfo().getId() != null)
      System.out.println("Selected kunde: id: "
        + getZZZKundeInfo().getId());
    if (getZZZLkwInfo().getId() != null)
      System.out.println("Selected lkw: id: "
        + getZZZLkwInfo().getId());
  }
  }
```

6.4.4.2. ZZZActivityKundeBearbeiten.java



(913b)

132. Create the following method:

```
package de.dekra.tutorial.model;
import com.mynd.dse.ps.searchagent.*;
import com.mynd.dse.ccb.*;
import com.mynd.dse.base.*;
import java.awt.event.*;
import java.util.*;
//ZZZActivityKundeBearbeiten
public class ZZZActivityKundeBearbeiten extends ServiceActivity {
  protected Kunde currentCustomer;
```

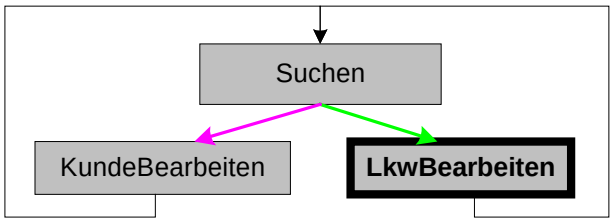
```
// #3/5 Listeners
// SaveListener for view button Save.
// sends message done() to controller
 class SaveListener implements ActionListener {
     public void actionPerformed(ActionEvent e) {
       System.out.println(e.toString());
       done();
     }
   }
// CancelListener for view button Cancel.
// sends message failed() to controller
   class CancelListener implements ActionListener {
     public void actionPerformed(ActionEvent e) {
       System.out.println(e.toString());
       failed();
     }
   }
 public ZZZActivityKundeBearbeiten() {
   super();

   }
// #1/5 canStart
// true if object in SLOTKunde
 public boolean canStart() {
   {
 currentCustomer = (Kunde) get("SLOTKunde");
     return currentCustomer != null;
   }
 }
// #2/5 doStart
// addCommandRules, get Kunde from SLOTKunde
 public void doStart() {
// addCommandRules for View buttons
   addCommandRule("Cancel", new CancelListener());
   addCommandRule("Save", new SaveListener());
// get Kunde from SLOTKunde
   Kunde customer;
   customer = (Kunde) get("SLOTKunde");
// info to console
   System.out.println("ZZZActivityKundeBearbeiten für Kunden: "
     + customer.getId() + " "
     + customer.getMyLkw() + " "
     + customer.getName1());
 }
// #4/5 canComplete
// always true
 public boolean canComplete() {

   return true;
 }
// #5/5 doComplete
// Send Kunde info to console
 public void doComplete() {
   Kunde customer;
   customer = (Kunde) get("SLOTKunde");
   System.out.println("ZZZActivityKundeBearbeiten Ende für Kunden: "
     + customer.getId() + " "
```

```
      + customer.getMyLkw() + " "
      + customer.getName1());
  }
  }
```

6.4.4.3. ZZZActivityLkwBearbeiten.java



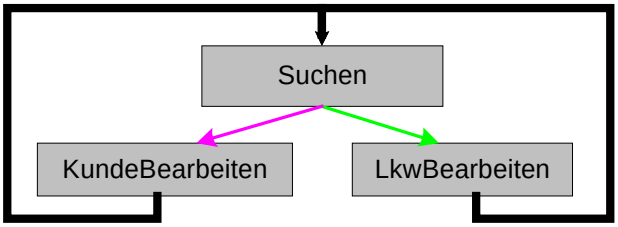
(913c)

133. Create the following method:

```
package de.dekra.tutorial.model;
import com.mynd.dse.ps.searchagent.*;
import com.mynd.dse.ccb.*;
import com.mynd.dse.base.*;
import java.awt.event.*;
import java.util.*;
//ZZZActivityLkwBearbeiten
public class ZZZActivityLkwBearbeiten extends ServiceActivity {
  protected Lkw currentAuto;
// #3/5 Listeners
// SaveListener for view button Save.
// sends message done() to controller
  class SaveListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      System.out.println(e.toString());
      done();
    }
  }
// CancelListener for view button Cancel.
// sends message failed() to controller
  class CancelListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      System.out.println(e.toString());
      failed();
    }
  }
public ZZZActivityLkwBearbeiten() {
  super();
}
// #1/5 canStart
// true if object in SLOTKunde
public boolean canStart() {
  {
    currentAuto = (Lkw) get("SLOTLkw");
    return currentAuto != null;
  }
}
// #2/5 doStart
// addCommandRules, get Lkw from SLOTLkw
public void doStart() {
// addCommandRules for View buttons
  addCommandRule("Cancel", new CancelListener());
```

```
   addCommandRule("Save", new SaveListener());
// get Lkw from SLOTLkw
   Lkw auto;
   auto = (Lkw) get("SLOTLkw");
// info to console
   System.out.println("EditAuto start for auto: "
     + auto.getId() + " "
     + auto.getMyKunde() + " "
     + auto.getModel());
 }
// #4/5 canComplete
// always true
 public boolean canComplete() {
   return true;
 }
// #5/5 doComplete
// Send Lkw info to console
 public void doComplete() {
   Lkw auto;
   auto = (Lkw) get("SLOTLkw");
   System.out.println("EditAuto end for auto: "
     + auto.getId() + " "
     + auto.getMyKunde() + " "
     + auto.getModel());
 }
 }
```

6.4.5. Model: Transitions



(913d)

The transitions have already been specified in the XML datei.

6.4.5.1. Configuration files (XML file)

Start transition

Note that the start transition (the starting activity) is specified by SLOTSuchen (ZZZActivitySuchen).

```
<Activity name="root" start="SLOTSuchen" abend="IGNORE">
```

Transitions from SLOTSuchen (ZZZActivitySuchen)

The transitions from SLOTSuchen (ZZZActivitySuchen) are specified in this code:

```
      <Transition>
<!--allowed transitions after ZZZActivitySuchen -->
        <Start activity="SLOTSuchen"/>
        <End activity="SLOTKundeBearbeiten"/>
        <End activity="SLOTLkwBearbeiten"/>
        <End/>
      </Transition>
```

Final transition

Both SLOTKundeBearbeiten (ZZZActivityKundeBearbeiten) and SLOTLkwBearbeiten (ZZZActivityLkwBearbeiten) specify SLOTSuchen (ZZZActivitySuchen) as the transition.

```
<Transition>
<!--allowed transitions after ZZZActivityKundeBearbeiten -->
        <Start activity="SLOTKundeBearbeiten"/>
        <End activity="SLOTSuchen"/>
        <End/>
      </Transition>
      <Transition>
<!--allowed transitions after ZZZActivityLkwBearbeiten -->
        <Start activity="SLOTLkwBearbeiten"/>
        <End activity="SLOTSuchen"/>
        <End/>
      </Transition>
```

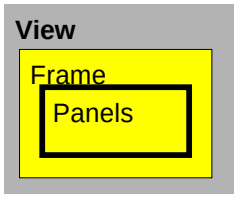
Therefore, the final transition (and end of the application) occurs when the dialog is closed.

6.4.6. Views

You will now create the Frame and Panels for the user interfaces.

6.4.6.1. de.dekra.tutorial.view.ZZZPanelSuchen

Overview



(909zb)

Dialog components:

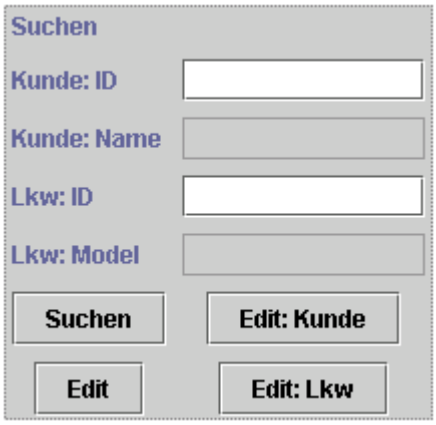


Abbildung 80. Panel de.dekra.tutorial.view.ZZZPanelSuchen (162)

Create class

134. Right-click on de.dekra.tutorial.view.
135. Select Add / Class.... The Create class Smartguide appears.
136. Select Create a new class (already selected).
137. Enter Class name: ZZZPanelSuchen.
138. Enter Superclass: javax.swing.JPanel.
139. Check: Browse class when finished.
140. Check: Compose the class visually.

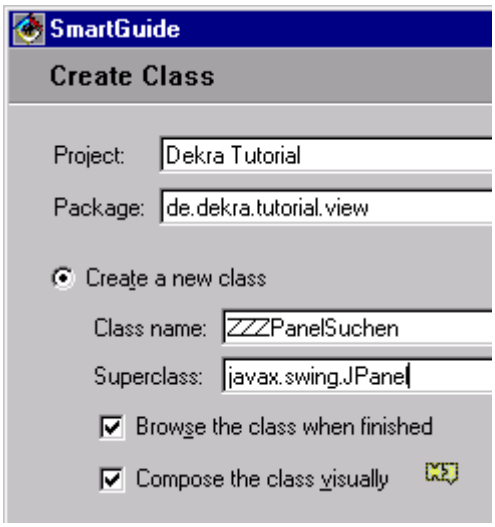


Abbildung 81. SmartGuide for de.dekra.tutorial.view.ZZZPanelSuchen (164)

141. Click Next. The Attributes dialog appears. The default values are ok.

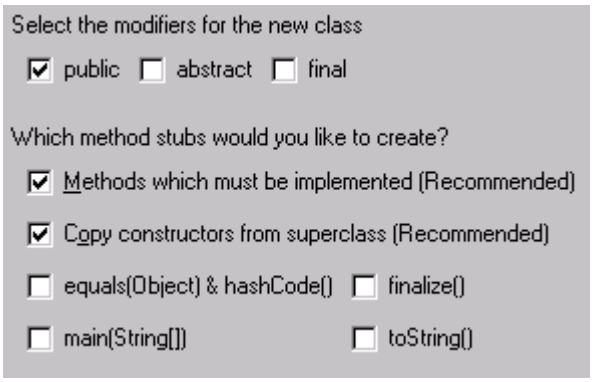


Abbildung 82. Attributes for de.dekra.tutorial.view.ZZZPanelSuchen (165)

142. Select Finish. The class is created.

Add labels

143. Open the class in the Visual Composition editor.
144. Select the panel.
145. Right-click.
146. Select properties. Properties dialog appears.
147. Set layout = GridBagLayout.
148. Add a JLabel with text "Suchen".
149. Add a JLabel below with text "Kunde: ID".
150. Add a JLabel below with text "Kunde: Name".
151. Add a JLabel below with text "Lkw: ID".
152. Add a JLabel below with text "Lkw: Model".

Abbildung 83. JLable's for de.dekra.tutorial.view.ZZZPanelSuchen (157)

Add DseJTextField for Kunde id

153. Add a DseJTextField to the right of label "Kunde: ID".
154. Click in the field to the right of property controlConnectDescriptor. A button appears.
155. Click on the button. The ControlConnectDescriptor dialog appears.
156. Click on "content ->".
157. In "connect to...": Enter SLOTSuchen.SLOTKundeInfo.id.

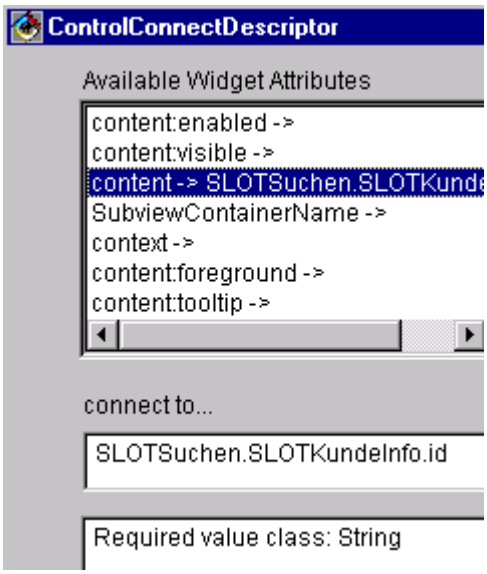


Abbildung 84. ControlConnectDescriptor dialog (158)

Abbildung 85. controlConnectDescriptor setting (159)

158. Set editable = True.

Add DseJTextField for Kunde name

159. Add DseJTextField to the right of Kunde: Name with:

- controlConnectDescriptor: content -> SLOTSuchen.SLOTKunde.name1
- editable = False.

Add DseJTextField for Lkw id

160. Add DseJTextField to the right of Lkw: ID with:

- controlConnectDescriptor: content -> SLOTSuchen.SLOTLkwInfo.id
- editable = True.

Add DseJTextField for Lkw model

161. Add DseJTextField to the right of Lkw: Model with:

- controlConnectDescriptor: content -> SLOTSuchen.SLOTLkw.model
- editable = False.

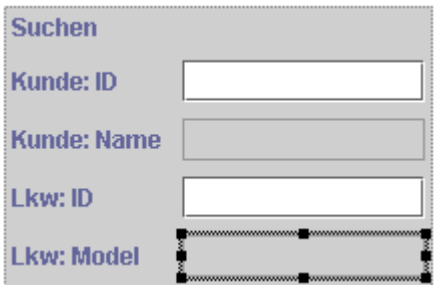


Abbildung 86. ZZZPanelSuchen (160)

Add DseJButton Suchen

162. Add a DseJButton with:

- controlConnectDescriptor: content -> SLOTSuchen.CRSearch
- text Suchen.

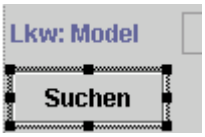


Abbildung 87. controlConnectDescriptor setting (161)

Add DseJButton Edit

163. Add a DseJButton with

- controlConnectDescriptor: content -> SLOTSuchen.CREdit
- text Edit.

Add DseJButton Edit: Kunde

164. Add a DseJButton with

- controlConnectDescriptor: content -> SLOTKundeBearbeiten
- text Edit: Kunde.

Add DseJButton Edit: Lkw

165. Add a DseJButton with

- controlConnectDescriptor: content -> SLOTLkwBearbeiten
- text Edit: Lkw.

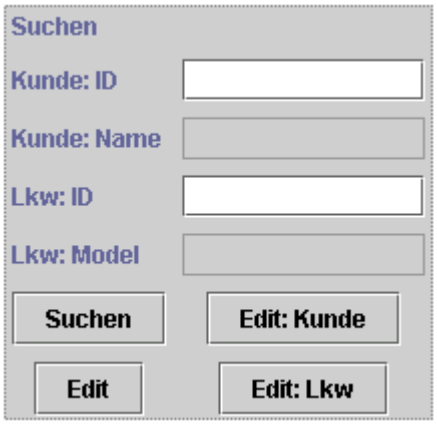


Abbildung 88. Complete de.dekra.tutorial.view.ZZZPanelSuchen (162)

Overview: controlConnectDescriptors.

Note: "cCD.c->" = "controlConnectDescriptor.content->"

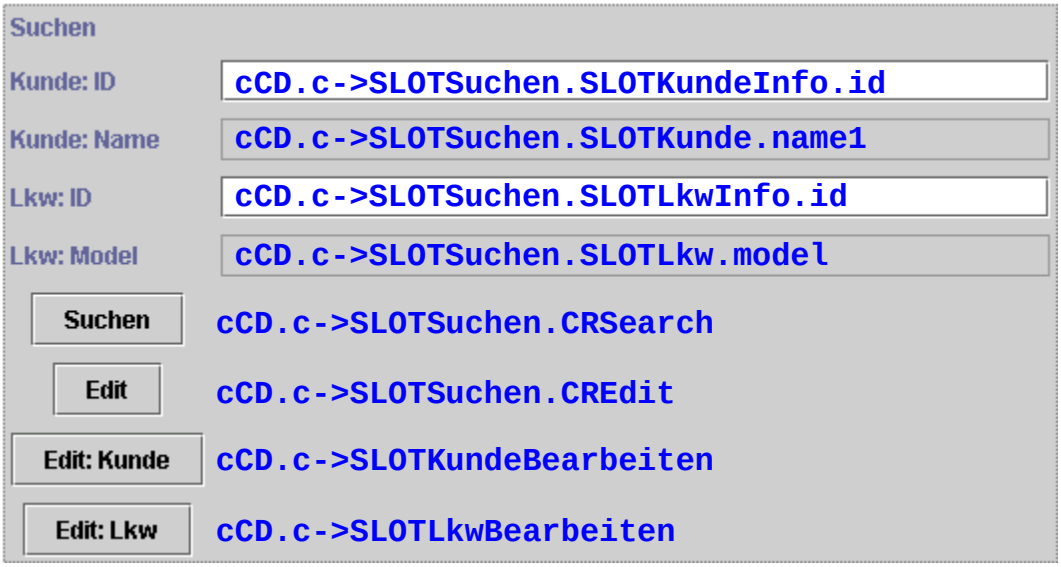
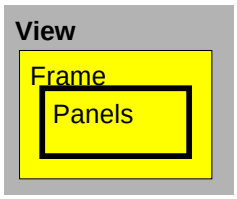


Abbildung 89. controlConnectDescriptors for ZZZPanelSuchen (919)

6.4.6.2. de.dekra.tutorial.view.ZZZPanelKundeBearbeiten



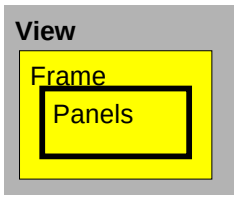
(909zb)

166. Create the following dialog with the components and controlConnectorDescriptors as shown.

Note: "cCD.c->" = "controlConnectDescriptor.content->"

Abbildung 90. Panel de.dekra.tutorial.view.ZZZPanelKundeBearbeiten (920)

6.4.6.3. de.dekra.tutorial.view.ZZZPanelLkwBearbeiten



(909zb)

167. Create the following dialog with the components and controlConnectorDescriptors as shown.

Note: "cCD.c->" = "controlConnectDescriptor.content->"

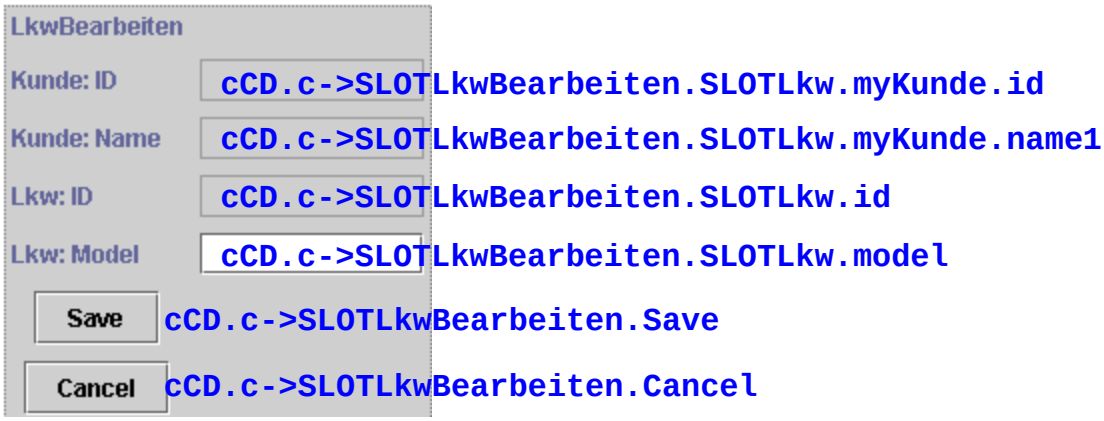
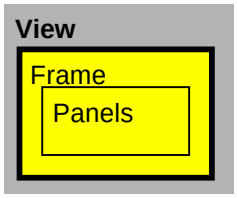


Abbildung 91. Panel de.dekra.tutorial.view.ZZZPanelLkwBearbeiten (921)

6.4.6.4. de.dekra.tutorial.view.ZZZFrame



(909zc)

Create class

168. Create a class for de.dekra.tutorial.view with:

- Class name: ZZZFrame
- Superclass: javax.swing.JFrame

169. For the JFrameContentPane of the Frame: Set layout = BorderLayout.

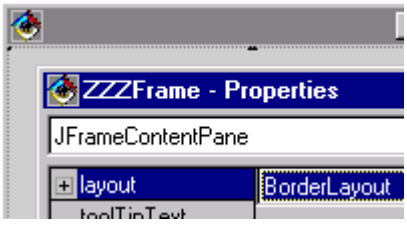


Abbildung 92. ZZZFrame content pane layout setting (168)

Add a DseJPanel to ZZZFrame

170. Click on the Choose bean icon  . The Choose Bean dialog appears.
171. For Class Name: Enter: com.mynd.dse.ac.swingbeans.DseJPanel.

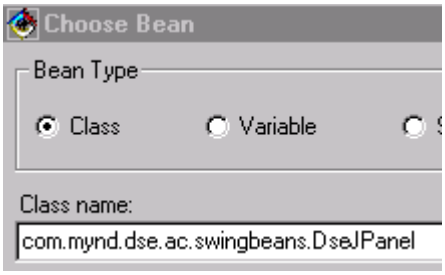


Abbildung 93. Choosing Bean DseJPanel (170)

172. Click OK. the cursor becomes crosshairs.

173. Click in the center of the ZZZFrame panel to add the DseJPanel to the center (constraints = center).

174. Set the layout to card layout.

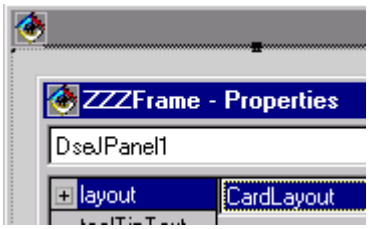
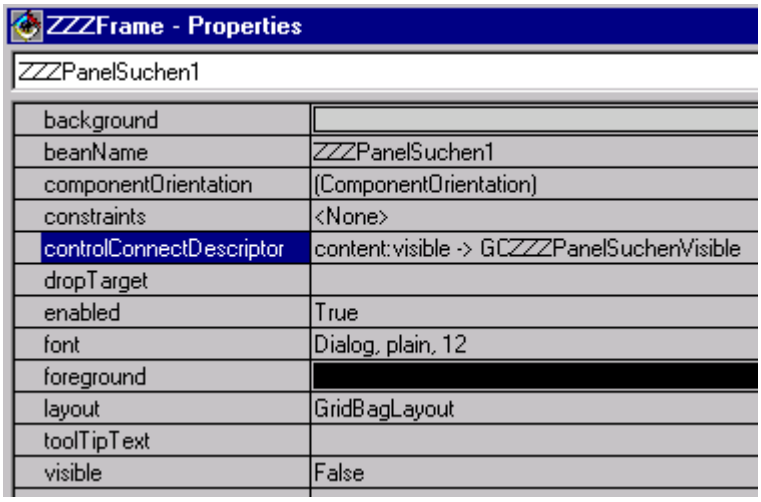


Abbildung 94. DseJPanel layout setting (171)

Add ZZZPanelSuchen

175. Add ZZZPanelSuchen to DseJPanel.

176. Set the controlConnectorDescriptor content:visible-> to GCZZZPanelSuchenVisible.



(172)

Abbildung 95. ZZZPanelSuchen settings in ZZZFrame (172)

Add ZZZPanelKundeBearbeiten

177. Add ZZZPanelKundeBearbeiten to DseJPanel.

178. Set the controlConnectorDescriptor content:visible-> to GCZZZPanelKundeBearbeitenVisible.

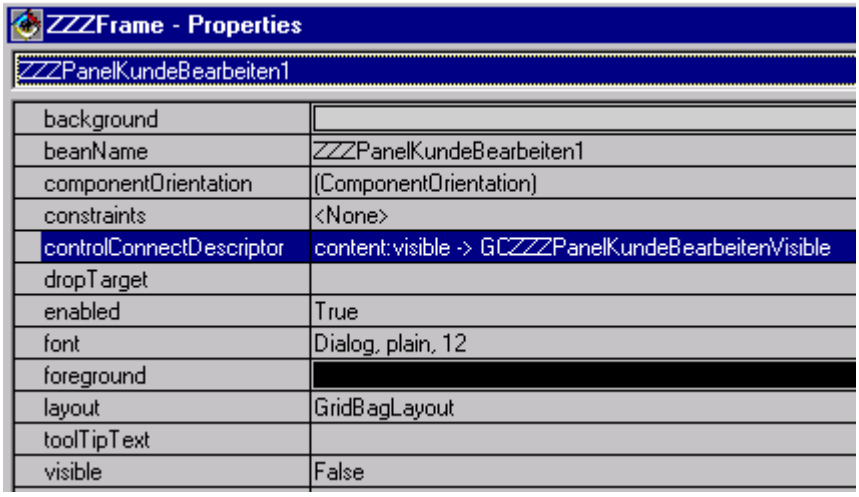


Abbildung 96. ZZZPanelKundeBearbeiten settings in ZZZFrame (173)

Add ZZZPanelLkwBearbeiten

179. Add ZZZPanelLkwBearbeiten to DseJPanel.
180. Set the controlConnectorDescriptor content:visible-> to GCZZZPanelLkwBearbeitenVisible.

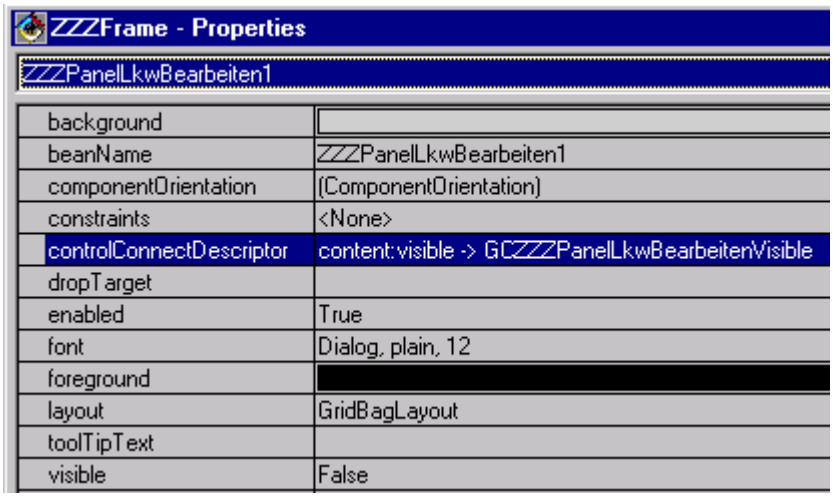
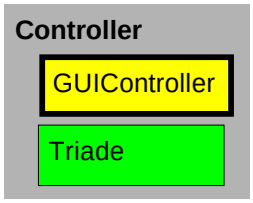


Abbildung 97. ZZZPanelLkwBearbeiten settings in ZZZFrame (174)

181. Save ZZZFrame.

6.4.7. Controller: GUIController class (ZZZGC)



(909ze)

182. Create the following class:

```
package de.dekra.tutorial.controller;
import com.mynd.dse.actfw.*;
import com.mynd.dse.common.*;
import com.mynd.dse.configuration.*;
import com.mynd.dse.ac.*;
import com.mynd.dse.gma.*;
import com.mynd.dse.models.*;
import com.mynd.dse.viewports.*;
```

```
import com.mynd.dse.viewports.desc.*;
import de.dekra.tutorial.model.*;
import de.dekra.tutorial.view.*;
// graphics controller
public class ZZZGC extends GuiControllerBase implements IConfigurationConstants {
public ZZZGC() {
  super(Atom.forString("root"));
}
// open ZZZFrame
public void open()  {
    String procName = ApplicationContext.getProperty(PROCESS_NAME);
    doInit(ZZZFrame.class, procName);
    doOpen();
}
// AttributeDescriptors control which pane is visible in the
    ZZZFrame
public static java.util.List getAttributeDescriptors() {
  java.util.ArrayList aList = new java.util.ArrayList();
  aList.add(new PrimAttributeDescriptor("GCZZZPanelSuchenVisible", boolean.class,
              "getZZZPanelSuchenVisible"));
  aList.add(new PrimAttributeDescriptor("GCZZZPanelKundeBearbeitenVisible",
              boolean.class, "getZZZPanelKundeBearbeitenVisible"));
  aList.add(new PrimAttributeDescriptor("GCZZZPanelLkwBearbeitenVisible",
              boolean.class, "getZZZPanelLkwBearbeitenVisible"));
  return aList;
}
// determine if ZZZPanelSuchen in ZZZFrame should be visible
  public boolean getZZZPanelSuchenVisible()  {
    try {
      UseCaseWrapper ucw = (UseCaseWrapper)model();
      return
              ucw.getCurrentActivity().getActivityHandle().getActivityName().equals
              ("SLOTSuchen");
    } catch(GmaException ge) {
      ge.printStackTrace();
    }
    return false;
  }
// determine if ZZZPanelKundeBearbeiten in ZZZFrame should be visible
  public boolean getZZZPanelKundeBearbeitenVisible()  {
    try {
      UseCaseWrapper ucw = (UseCaseWrapper)model();
      return
              ucw.getCurrentActivity().getActivityHandle().getActivityName().equals
              ("SLOTKundeBearbeiten");
    } catch(GmaException ge) {
      ge.printStackTrace();
    }
    return false;
  }
// determine if ZZZPanelLkwBearbeiten in ZZZFrame should be visible
  public boolean getZZZPanelLkwBearbeitenVisible()  {
    try {
      UseCaseWrapper ucw = (UseCaseWrapper)model();
      return
              ucw.getCurrentActivity().getActivityHandle().getActivityName().equals
              ("SLOTLkwBearbeiten");
    } catch(GmaException ge) {
      ge.printStackTrace();
```

```
    }
    return false;
  }
// Notification of the underlying activity that a transition was
    peformed
public void transitionPerformed(IActivityHandle hActivityFrom, IActivityHandle
              hActivityTo, String presentationHint) {
  super.transitionPerformed(hActivityFrom, hActivityTo, presentationHint);
  thisAttribute().markAllValuesChanged();
}
}
```

6.4.8. Main class (ZZZMain)

183. Create main (runnable) class ZZZMain.

```
package de.dekra.tutorial.controller;
import com.mynd.dse.ps.*;
import com.mynd.dse.actfw.*;
import com.mynd.dse.ccb.*;
import com.mynd.dse.common.*;
import com.mynd.dse.configuration.*;
import org.xml.sax.*;
import java.io.*;
import javax.xml.parsers.*;
// ModelViewController der Schulung
public class ZZZMain extends GuiApplicationContext implements IConfigurationConstants
              {
public ZZZMain() {
}
public static void main(String args[]) {
  new ZZZMain();
}
// create the configuration from the .xml file
protected com.mynd.dse.actfw.Configuration createConfiguration() {
  try {
    return new ActivityStandardConfiguration(getProcessName());
  } catch (SAXException e1) {
    DBG.ASSERT(false, "nonvalid process name " + getProcessName());
  } catch (ParserConfigurationException e2) {
    DBG.ASSERT(false, "nonvalid process name " + getProcessName());
  } catch (IOException e4) {
    DBG.ASSERT(false, "nonvalid process name " + getProcessName());
  }
  return null;
}
protected Creator createCreator() {
  return new StandardCreator();
}
// initialization sequence:
//   initializeConfiguration();
//   initializeServiceRequester();
//   initializeManager();
//   initializeGuiController();
//   initializeActivityGuiConnections();
//   initializePOM();
//   startManager();
//   startGuiController();
```

```
 protected boolean initialize() {
   super.initialize();
   initializePOM();
   startManager();
   startGuiController();
   return true;
 }
// initialize the POM
 private void initializePOM()
 {
   try
   {
     Object o = ServiceRequester.lookup(POM_NAME,POM_DEFAULT_QUALITY);
     PersistenceManager pom = (PersistenceManager)ServiceRequester.narrow(o);
     Class.forName(ApplicationContext.getProperty(DATABASE_DRIVER));
     pom.connect(ApplicationContext.getProperty(DATABASE_SYSTEM_NAME),
       ApplicationContext.getProperty(DATABASE_USER_ID),
       ApplicationContext.getProperty(DATABASE_PASSWORD));
     getManager().setPersistenceManager(pom);
   }
   catch (Exception e1)
   {
     DBG.ASSERT(false,
         "can't create DBConnection");
   }
 }
 }
```

6.5. Test (manually)

Run / view console output.

Note: If your application doesnt run, then try loading the complete example Tutorial1 into your workspace and then try.

6.5.1. Requirements

This section assumes that you have completed section 6.4 MVC Implementation. If not, then perform the following:

1. Import from **TUTORIAL.dat**: Project **Dekra Tutorial** version **2.0**.
2. Copy the following files to [VA drive : directory] **\ide\project_resources\Dekra Tutorial**.

- config.properties
- services.preload.properties
- activity.configuration.xml
- activity.configuration.dtd

3. Add the database **zzzdb.odb** as ODBC datasource **ZZZDBDSN**.

6.5.2. Compute class path

4. Right-click on ZZZMain. The properties dialog appears.
5. In tab Class Path: For project path: Click Compute Now.
6. Click yes.
7. Click Edit.
8. Click Select All.

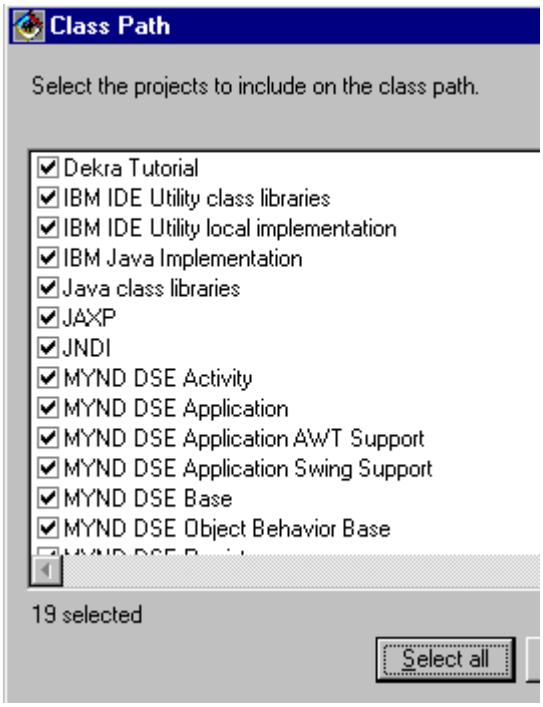


Abbildung 98. Projects for class path

9. Click OK.
10. Click OK.

6.5.3. Start

11. For ZZZMain: Select run / main. The following dialog appears (this may require a minute or 2).

Abbildung 99. ZZZMain dialog (175)

6.5.4. Kunden suchen

12. For Kunde ID: Enter 1.
13. Click Suchen. Note the results:

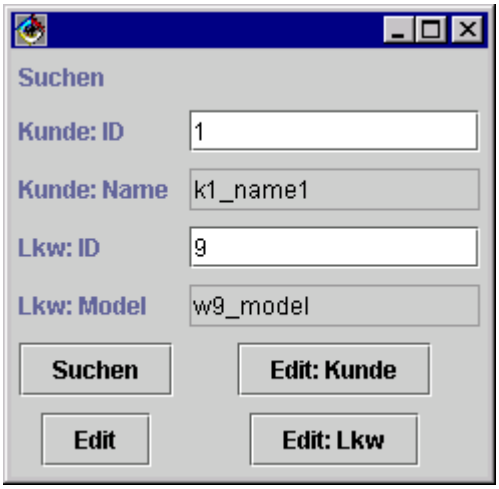


Abbildung 100. Search for Kunde with id=1 results (176)

6.5.5. Go to activity specified by XML file and canStart

Note that both the customer and auto have been found. Therefore, pressing on the Edit button will go the ZZZActivityKundeBearbeiten activity, since it is **specified first in the xml file**.

14. Click Edit. The ZZZPanelKundeBearbeiten view is shown.

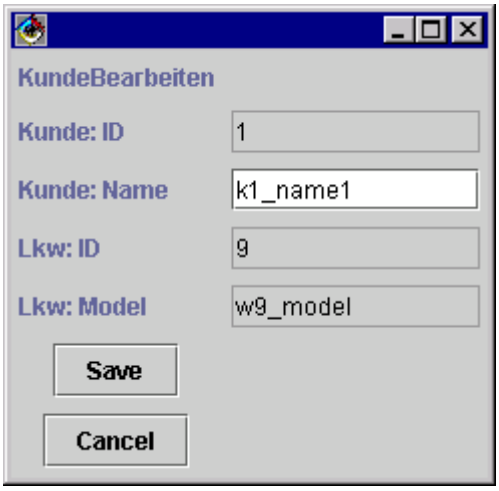


Abbildung 101. ZZZPanelKundeBearbeiten (result of pressing Edit button) (177)

6.5.6. Save changes to the database

15. Change Kunde name to k1_name1b.
16. Click Save.

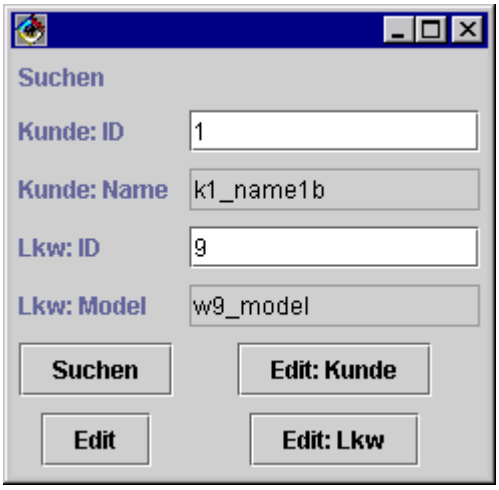


Abbildung 102. Saved changes (178)

17. Open the database KUNDE in Oracle. Note the change:

	ID	NAME1
1	1	k1_name1b

Abbildung 103. Saved changes to Kunde in the database (179)

6.5.7. Go direct to an activity

18. Click the Edit: Lkw button. Note that although a Kunde is available, the ZZZPanelLkwBearbeiten view is opened.

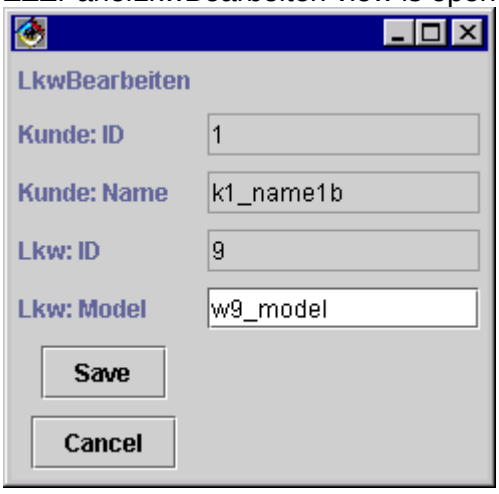


Abbildung 104. Direct to ZZZPanelLkwBearbeiten (180)

6.6. Test (with TestSuite)

NOTE: this section is not finished (001016)

The TestSuite can be used to automate testing.

6.6.1. Requirements

This section assumes that you have completed section 6.4 MVC Implementation. If not, then perform the following:

1. Import from **TUTORIAL.dat**: Project **Dekra Tutorial** version **2.0**.
2. Copy the following files to [VA drive : directory] **\ide\project_resources\Dekra Tutorial**.

- config.properties
- services.preload.properties
- activity.configuration.xml
- activity.configuration.dtd

3. Add the database **zzzdb.odb** as ODBC datasource **ZZZDBDSN**.

6.6.2. Create ActivityManagerTestSuite

6.6.2.1. Create class

4. Create the following class.

```
package de.dekra.tutorial.test;

import com.mynd.dse.testsuite.*;
import javax.swing.*;
import com.mynd.dse.actfw.*;

public class ActivityManagerTestSuite extends TestSuite {
  private static Manager manager;
  private static String aktuelleActivitaet ="";
public static ActivityGuiEventSupport getManager() {
  return (ActivityGuiEventSupport) manager;
}
/**
 * Starts the application.
 * @param args an array of command-line arguments
 */
public static void main(java.lang.String[] args) {
  // Insert code to start the application here.
  TestSuiteBrowser.openOn(new ActivityManagerTestSuite());
}
public static void setAktuelleActivitaet(String ta) {
  aktuelleActivitaet = ta;
}
/*
 * Testen der Funktionalitaet der Activity Login
 *
 **/
public static TestSuiteResult testCaseActivityIDEnterOK() {

  TestSuiteResult suiteRes = new TestSuiteResult(true);

  try {
      //starten des ActivityFrameworks
      testCaseStartActivityManager();
```

```
      aktuelleActivitaet = "";
      IActivityHandle aHandle = getManager().getActivityHandle("root", "Edit");
      IObjectHandle oHandle = (IObjectHandle)getManager().getValue(aHandle,
                "CustomerInfo");

      //Das Attribut "userID" im Slot "CustomerInfo" setzen

      getManager().setValue(oHandle, "id", "10");

      //Das Command "Search" in der Aktivitaet "Edit" ausfuehren
      getManager().command(aHandle, "Search");

      //Das Command "Edit" in der Aktivitaet "Edit" ausfuehren
      getManager().command(aHandle, "Edit");

      //Ueberpruefen des erwarteten Ergebnisses
      if (aktuelleActivitaet.equals(Configuration._$END$)){
        return suiteRes.passed("Kunde mit id=10 in datenbank gefunden!");
        }
      return suiteRes.failed();
    }
    catch (Throwable e) {
    e.printStackTrace(System.out);
    return suiteRes.failed(e.toString());
  }
}
public static TestSuiteResult testCaseStartActivityManager() {
  TestSuiteResult suiteRes = new TestSuiteResult(true);
  Object result;
  String configFilePath = null;
  try {
    JFileChooser fileDialog = new JFileChooser(".");
    fileDialog.showOpenDialog(null);
    configFilePath = (String) fileDialog.getSelectedFile().getAbsolutePath();
    ActivityStandardConfiguration config = new
                ActivityStandardConfiguration(configFilePath);
    Creator creator = new StandardCreator();
    manager = new Manager();
    manager.setConfiguration(config);
    manager.setCreator(creator);
    manager.startProcess("root", "");
    manager.setGuiActivityNotificationListener(new GuiActivityNotificationListener()
                {
      public void transitionPerformed(IActivityHandle from, IActivityHandle to,
                String blubber) {
        setAktuelleActivitaet(to.getActivityName());
      }
    });
    return suiteRes;
  } catch (Throwable e) {
    e.printStackTrace(System.out);
    return suiteRes.failed(e.toString());
  }
}
}
```

6.6.2.2. Recompute class path

5. Compute the class path for the above class. Include the project Dekra Tutorial.

6.6.3. Run single test

6.6.3.1. Start class

6. Run the class. The following dialog appears:

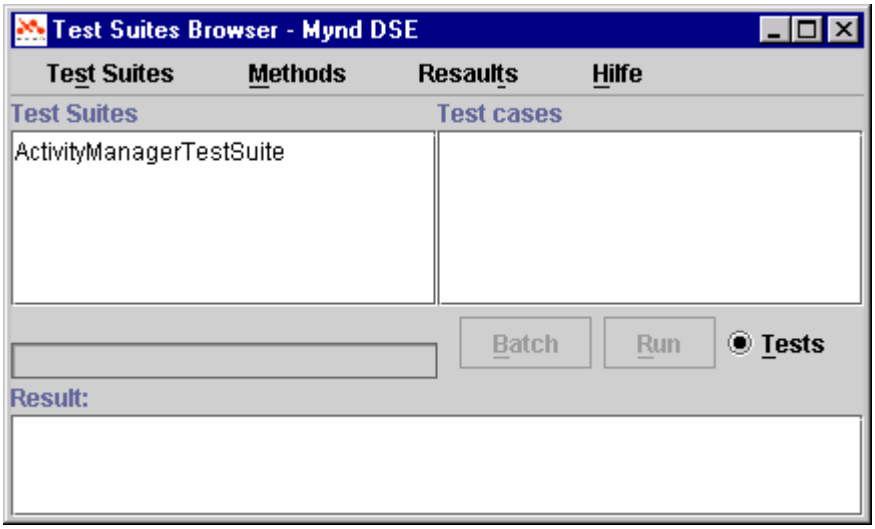


Abbildung 105. TestSuiteBrowser main dialog

6.6.3.2. Display available test suites

7. Click on TestSuite ActivityManagerTestSuite. Note the available test suites.

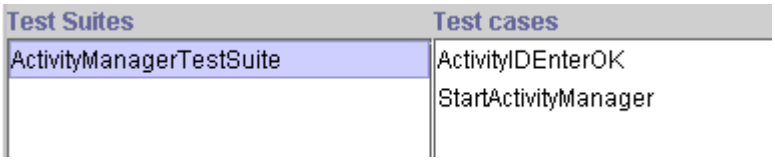


Abbildung 106. Available test suites

6.6.3.3. Run ActivityIDEnterOK

8. Click on ActivityIDEnterOK.
9. Click Run. The Open dialog appears.
10. Select activity.configuration.xml.

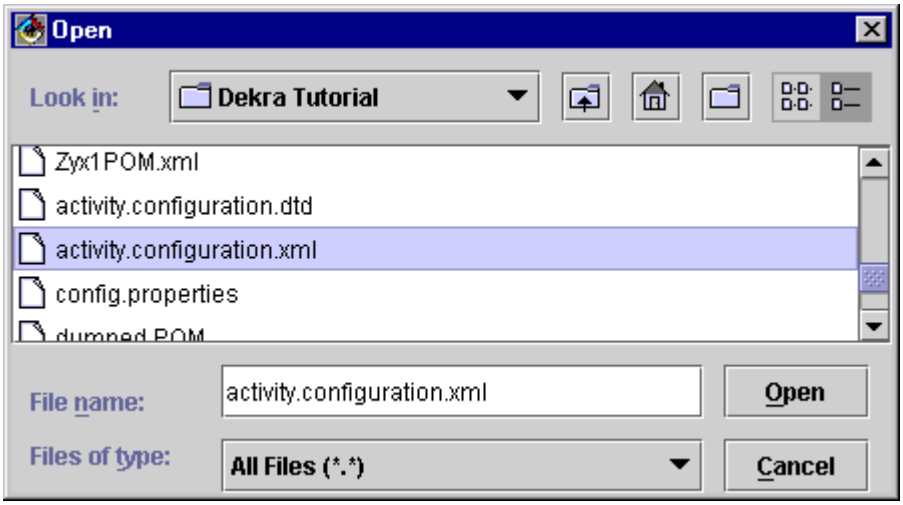


Abbildung 107. Selecting activity.configuration.xml file for test

11. Click Open. The test case is executed. Note the message.

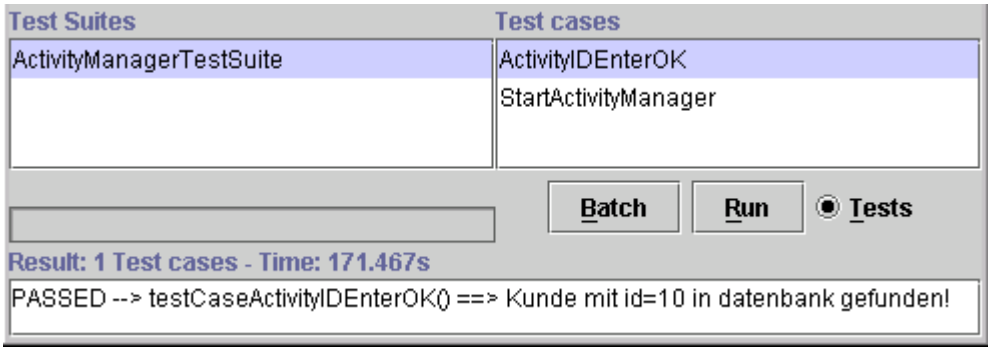


Abbildung 108.Test passed

6.6.4. Batch test (run multiple tests)

12. Hold down the Shift key.
13. Click on StartActivityManager. Both test cases should be selected (blue).

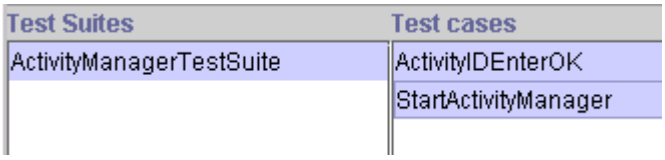


Abbildung 109. Selecting multiple test cases

14. Click Run. Note that both tests are carried out (you will be prompted 2 times to specify the xml file).

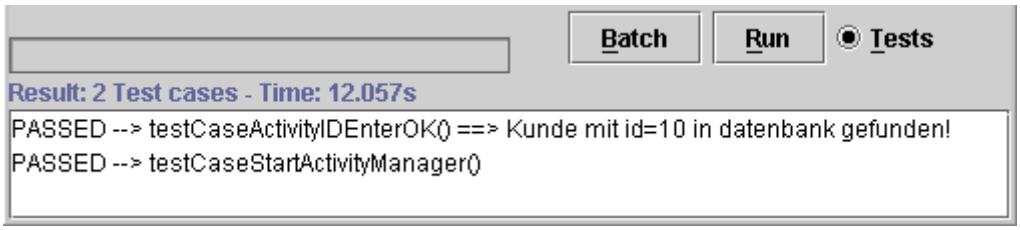


Abbildung 110. Results of multiple tests

6.6.5. Failing test

6.6.5.1. Create method

15. Create this method.

```
/*
 * Testen der Funktionalitaet der Activity Login
 *
 **/
public static TestSuiteResult testCaseActivityIDEnterFAIL() {

  TestSuiteResult suiteRes = new TestSuiteResult(true);

  try {
      //starten des ActivityFrameworks
      testCaseStartActivityManager();

      aktuelleActivitaet = "";
      IActivityHandle aHandle = getManager().getActivityHandle("root", "Edit");
      IObjectHandle oHandle = (IObjectHandle)getManager().getValue(aHandle,
               "CustomerInfo");

      //Das Attribut "userID" im Slot "CustomerInfo" setzen
```

```
        getManager().setValue(oHandle, "id", "99");

        //Das Command "Search" in der Aktivitaet "Edit" ausfuehren
        getManager().command(aHandle, "Search");

        //Das Command "Edit" in der Aktivitaet "Edit" ausfuehren
        getManager().command(aHandle, "Edit");

        //Ueberpruefen des erwarteten Ergebnisses
        if (aktuelleActivitaet.equals(Configuration._$END$)){
          return suiteRes.passed("Kunde mit id=99 in datenbank gefunden!");
          }
        return suiteRes.failed();
      }
      catch (Throwable e) {
      e.printStackTrace(System.out);
      return suiteRes.failed(e.toString());
    }
  }
```

6.6.5.2. Test

16. Refresh the TestSuiteBrowser dialog (by clicking Tools and Tests radio buttons). The new test appears:

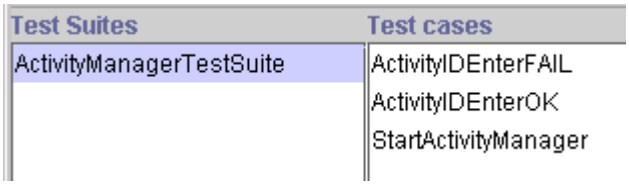


Abbildung 111. New test in TestSuite

Run test ActivityIDEnterFAIL.

6.6.6. Failing test case

6.6.6.1. Add method Start class

17. Run the class. The following dialog appears:

6.7. Modify the object net

Completed code available in TUTORIAL.DAT project Dekra Tutorial version 2.1.

In this section you will make a very simple modification to the object net:

- Add variable name2 to Kunde.

This will require the following changes to:

- Object net: Add variable name2 to Kunde
- Run-time POM (specify in config.properties).
- Database. Add table column to KUNDE.
- Views.

Then you will test.

6.7.1. Add variable name2 to Kunde

You will now create a new object model with the added variable for class Kunde.

6.7.1.1. Modify object net

1. Open the OMB.
2. Select Model / Open model. The Open file... dialog appears.
3. Double-click on ZZZObjectModel1.ome. The object net appears in OMB:

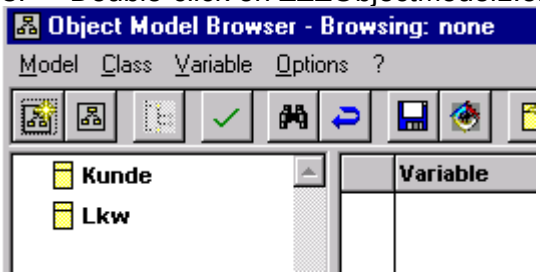


Abbildung 112. ZZZObjectModel1.ome in OMB(187)

4. Add Kunde variable name2 with the same settings as for name1:

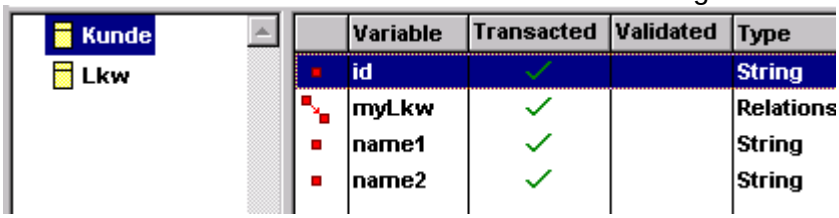


Abbildung 113. Kunde with new variable name2 (188)

5. Save the model (Model / Save...) as ZZZObjectModel2.ome.

6.7.1.2. Create VA classes

6. Save the model to VA (Model / Save to VA). Note the new classes in VA (a scratch edition is automatically created).

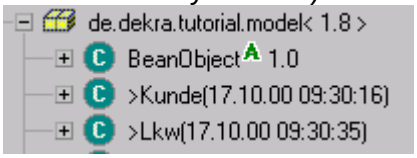


Abbildung 114. New Kunde and Lkw versions in VA (189)

6.7.2. Create new Runtime-POM

6.7.2.1. Create POMG

7. Model / Tools / Model->POM Generator. The POM Generator appears.
8. Choose the POM class.
9. Configurate POM.
10. Choose the possible classes.

Store globally as ZZZPOM2.

6.7.2.2. Generate POM

11. Generate POM

6.7.2.3. Generate RT-POM

12. In Model DB Mappings tool: Select Manager / Generate RTPOM.
13. Specify as shown below:

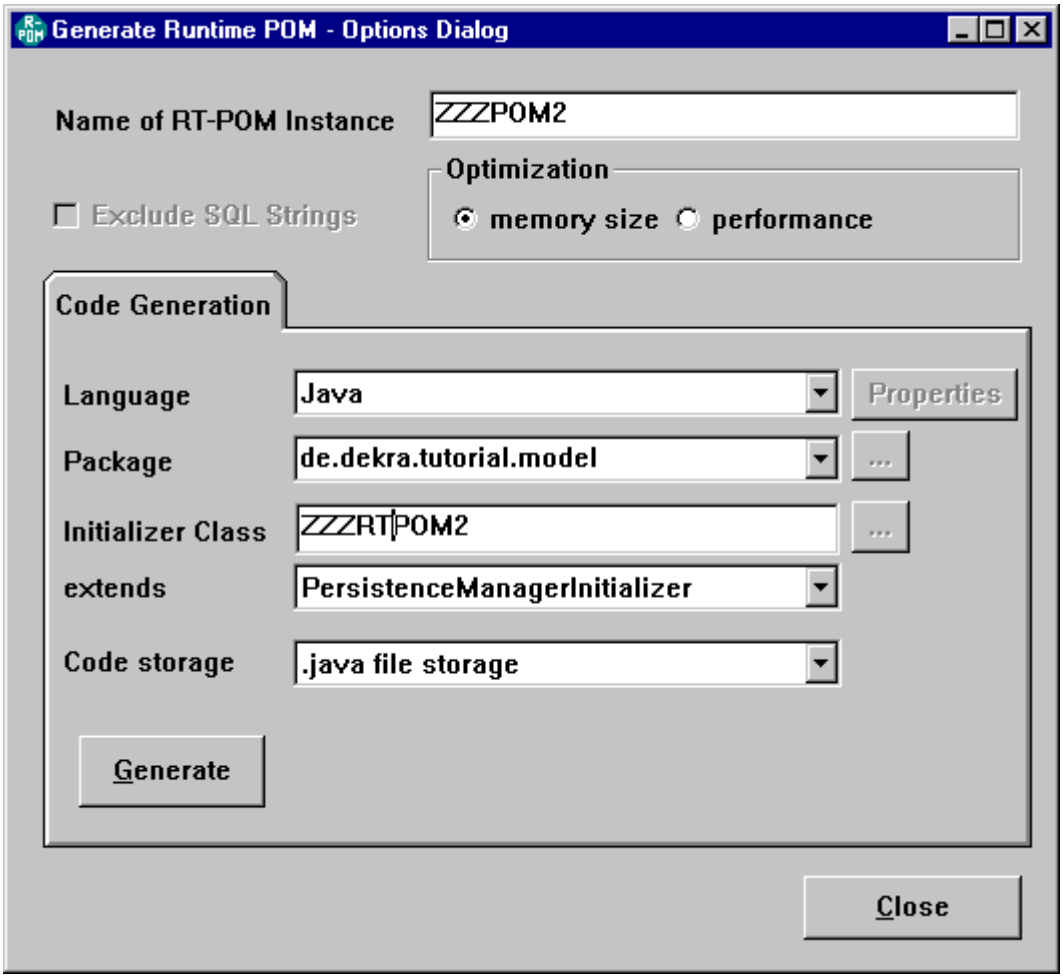


Abbildung 115. RTPOM Generator settings for ZZZRTPOM2 (190)

14. Click Generate.
15. Specify output file **ZZZRTPOM2.java**.

6.7.2.4. Import into VA RT-POM

16. Import **ZZZRTPOM2.java** into VA.

6.7.2.5. Modify RTPOM spec

17. Modify the following in config.properties.

```
# pom initializer
persistenceManager= de.dekra.tutorial.model.ZZZRTPOM2
```

6.7.3. Add table column NAME2 to KUNDE

6.7.3.1. Generate SQL

18. In Model DB Mappings: Select Schema / Export. Following text is generated.

```
CREATE TABLE KUNDE (
ID  VARCHAR2(20) NOT NULL,
NAME2           VARCHAR2(20),
NAME1           VARCHAR2(20),
PRIMARY KEY (ID)
);
commit;

CREATE TABLE LKW (
ID  VARCHAR2(20) NOT NULL,
MODEL           VARCHAR2(254),
KUNDEID         VARCHAR2(20) NOT NULL,
PRIMARY KEY (ID)
);
commit;

ALTER TABLE LKW ADD FOREIGN KEY (KUNDEID) REFERENCES KUNDE;
commit
```

19. Modify the text to the following

```
ALTER TABLE KUNDE ADD COLUMN NAME2 VARCHAR2(20);
commit
```

6.7.3.2. Execute SQL against DB

20. Open ISQL.
21. Select Manager / Choose.
22. Select ZZZPOM2.
23. Connect to ZZZDBDSN.
24. Execute (Statement / Execute once) the following

```
ALTER TABLE KUNDE ADD COLUMN NAME2 VARCHAR2(20);
commit
```

In Oracle 8i: Note the database change:

KUNDE

	ID	NAME1	NAME2
1	1	k1_name1b	

Abbildung 116. KUNDE database with new table NAME2 (191)

6.7.4. Modify views

Add name2 text field for Suchen and KundeBearbeiten.

6.7.4.1. Modify ZZZPanelSuchen

25. Add a label and DseJTextField for name2 with controlConnectDescriptor.

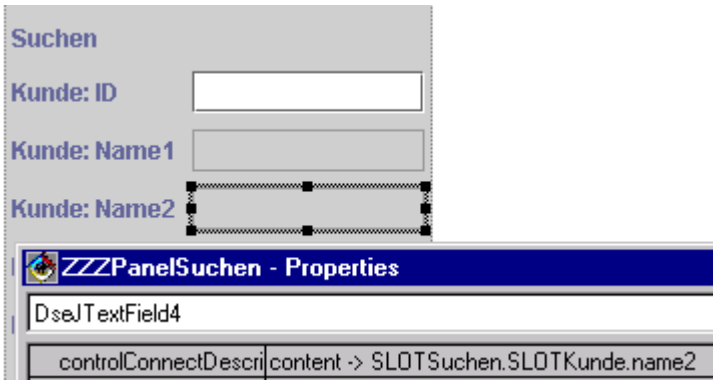


Abbildung 117. ZZZPanelSuchen with label, textfield for name2 (192)

6.7.4.2. Modify ZZZPanelKundeBearbeiten

26. Add a label and DseJTextField for name2 with controlConnectDescriptor.

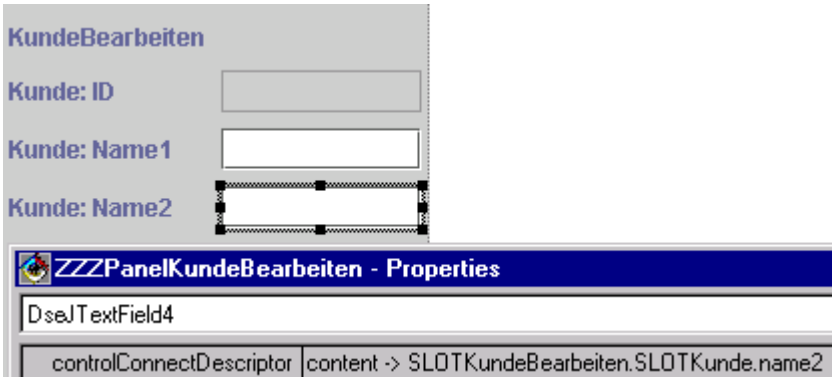


Abbildung 118. ZZZPanelKundeBearbeiten with label, textfield for name2 (193)

6.7.5. Test

27. Run ZZZMain. Note the new dialog:

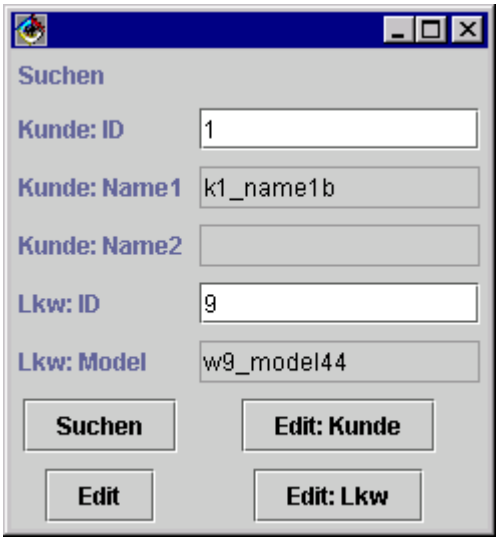


Abbildung 119. ZZZMain main dialog with new textfield (194)

28. Click Edit: Kunde.
29. Enter k1_name2 for name2.
30. Click Save.

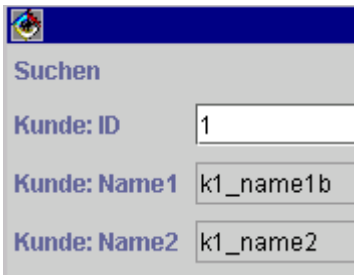


Abbildung 120. Text for name2 succesfully entered (195)

6.8. ViewPorts

Completed code available in TUTORIAL.DAT project Dekra Tutorial version 2.2.

In this section you will make the changes required for displaying the Kunde name in a single text field using a view port.

This will require the following changes :

- Modifiy View.
- Modify GUIController.
- Create Activity Viewport.
- Create Object Viewport.
- Test.

6.8.1. Modify ZZZPanelKundeBearbeiten

1. Add a DseJTextField to ZZZPanelKundeBearbeiten as shown:

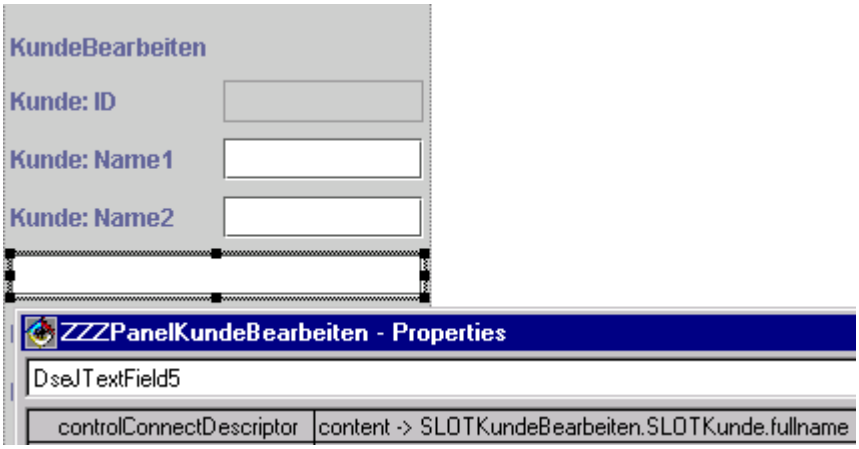


Abbildung 121. ControlConnectDescriptor for fullname (196)

2. Save the class.

6.8.2. Modify GUIController

3. Add the line in bold below to **de.dekra.tutorial.controller.ZZZGC.getAttributeDescriptors**:

```
// AttributeDescriptors control which pane is visible in the ZZZFrame
public static java.util.List getAttributeDescriptors() {
  java.util.ArrayList aList = new java.util.ArrayList();
  aList.add(new PrimAttributeDescriptor("GCZZZPanelSuchenVisible", boolean.class,
                "getZZZPanelSuchenVisible"));
  aList.add(new PrimAttributeDescriptor("GCZZZPanelKundeBearbeitenVisible",
                boolean.class, "getZZZPanelKundeBearbeitenVisible"));
  aList.add(new PrimAttributeDescriptor("GCZZZPanelLkwBearbeitenVisible",
                boolean.class, "getZZZPanelLkwBearbeitenVisible"));
// activity viewport declaration
  aList.add(new ActivityAttributeDescriptor("SLOTKundeBearbeiten",
                ZZZViewportActivityKundeBearbeiten.class));
  return aList;
}
```

4. Save the class.

6.8.3. Create activity viewport

5. Create the following class:

```
package de.dekra.tutorial.controller;
```

```
import com.mynd.dse.actfw.*;
import com.mynd.dse.viewports.*;
import com.mynd.dse.viewports.desc.*;
import com.mynd.dse.models.*;
import java.util.*;
public class ZZZViewportActivityKundeBearbeiten extends ActivityViewPortBase {
  public ZZZViewportActivityKundeBearbeiten(IAttribute thisAttribute)
  {
    super(thisAttribute);
  }
public static List getAttributeDescriptors() {
  List aList = new ArrayList();
  aList.add(new ActivityAttributeDescriptor("SLOTKunde", ZZZViewportKunde.class));
  return aList;
}
}
```

6. Save the class.

6.8.4. Create object viewport

7. Create the following class:

```
package de.dekra.tutorial.controller;

import com.mynd.dse.gma.*;
import com.mynd.dse.models.*;
import com.mynd.dse.viewports.*;
import com.mynd.dse.viewports.desc.*;
import java.util.*;
public class ZZZViewportKunde extends EntityViewPortBase {
public ZZZViewportKunde(IAttribute thisAttribute)
{
  super(thisAttribute);
}
public static List getAttributeDescriptors() {
  List aList = new ArrayList();
  aList.add(new PrimAttributeDescriptor("fullname", String.class, "getFullname"));
  return aList;
}
public String getFullname() {
  String name1 = (String) getValue("name1");
  String name2 = (String) getValue("name2");
  String fullname = "";
  fullname = (name1 == null) ? "" : name1 + " ";
  if (name2 != null) {
    fullname += name2;
  }
  return fullname;
}
}
```

8. Save the class.

6.8.5. Test

9. Run ZZZMain. The Suchen dialog appears.
10. For Kunde: ID: Enter: 1.
11. Click Search.

12. Click Edit: Kunde. Note the fullname:

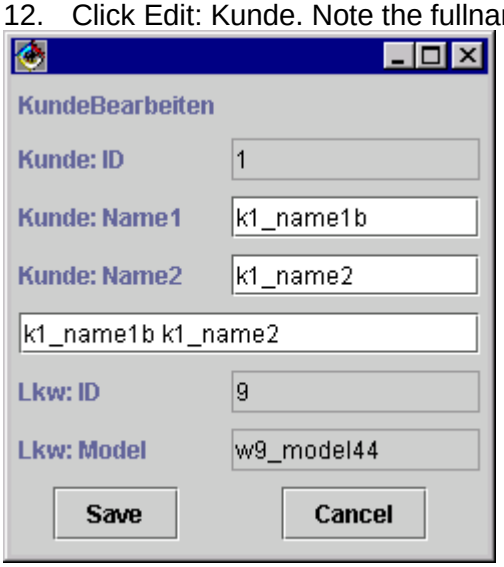


Abbildung 122. fullname in KundeBearbeiten view (197)

6.9. Transactions

This section describes the transaction mechanisms supported by Frameworks.

6.9.1. Why transactions

The following is a simple demonstration of what can happen when transactions are not used. Open multiple windows and demonstrate the problems.

6.9.2. Transactions: Non-isolated

Changes to .xml file.

6.9.3. Transactions: Isolated

6.10. Adding/deleting objects (from database)

6.11. Enable/disable view components

6.12. Validating input

Demonstrate how invalid input can be rejected based on:

- Type
- Range

6.13. Implementing tooltips

7. Anwendungsszenarien (Komplette Beispiele)

7.1. Einbindung in die DSE Gesamtarchitektur

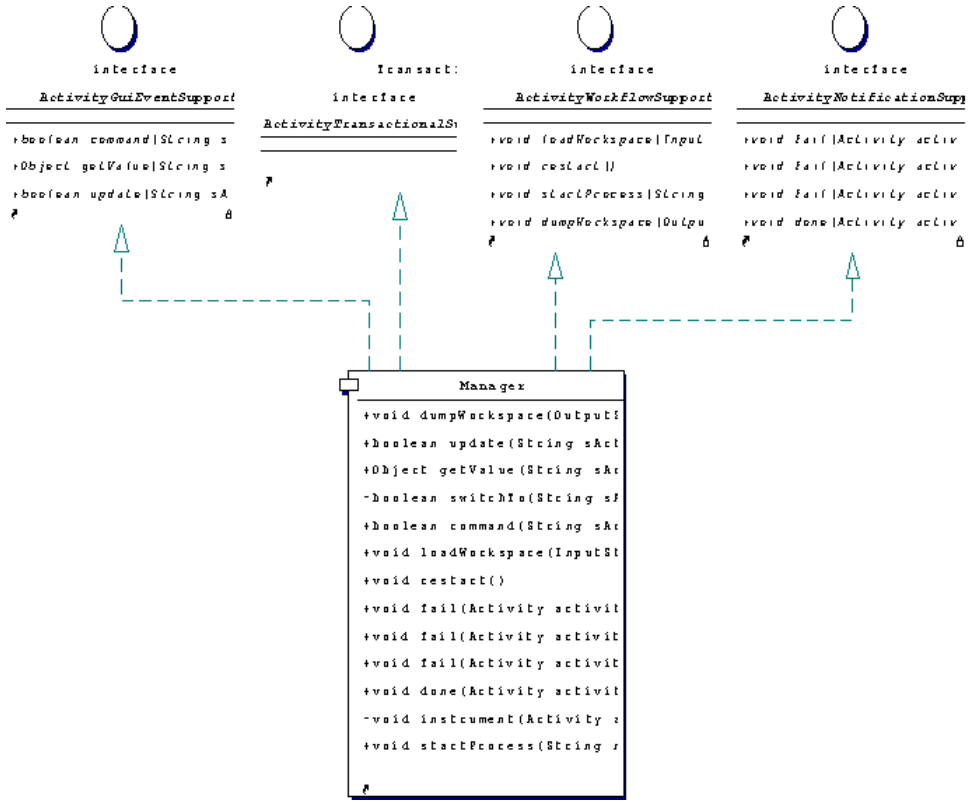


Abbildung 123: Manager Subsystem (managersubsystem.gif)

7.2. Einbindung in den Kommunikationsfluss einer Gesamtanwendung

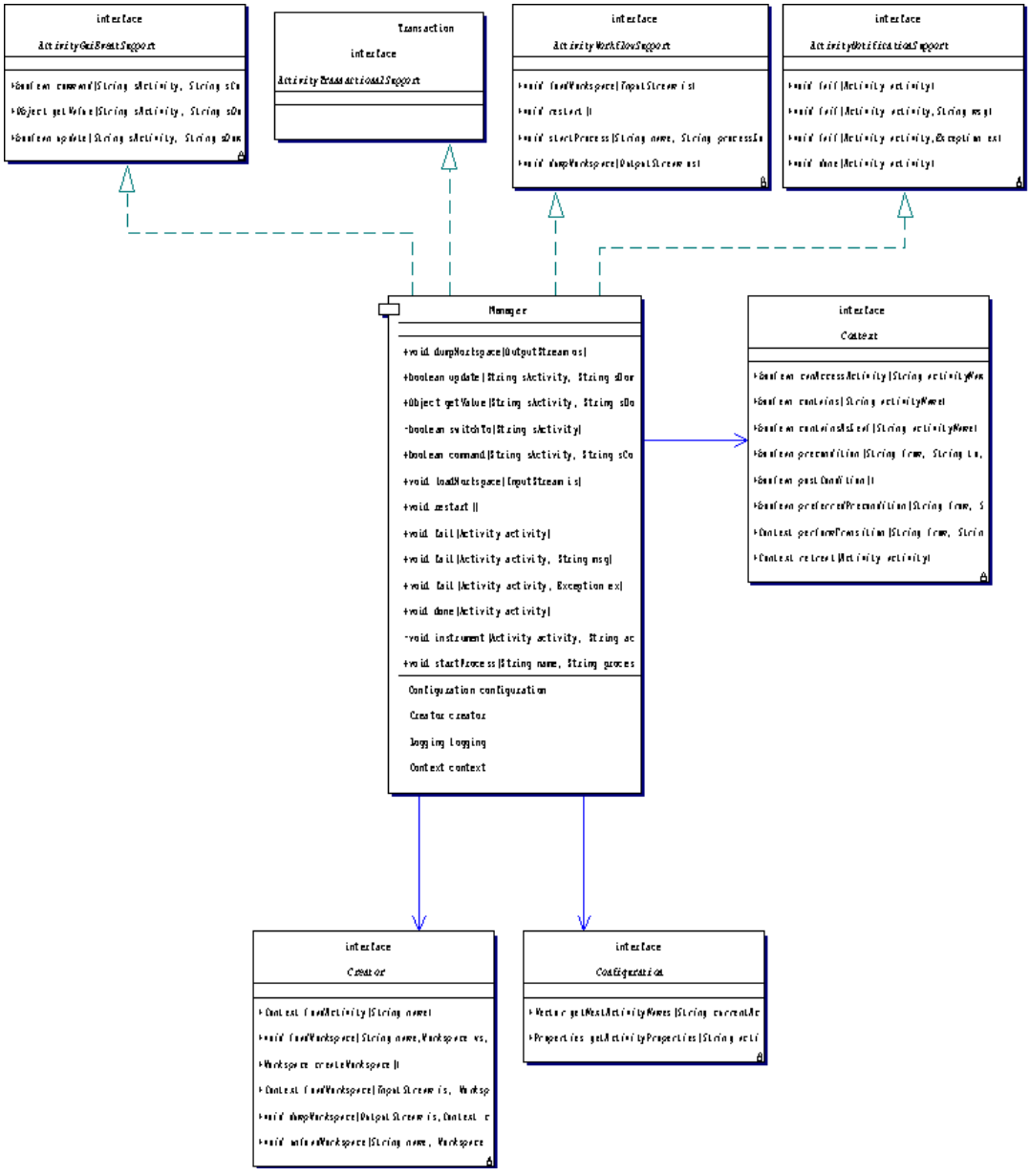


Abbildung 124: Manager All (managerall.gif)

7.3. Fat-client (mvc)

7.4. EJB-Container

7.5. Web-Server

7.6. Workflow-System (DSE-OFC)

8. Concepts

8.1. Triade (Manager, Konfiguration, Creator)

8.1.1. Überblick

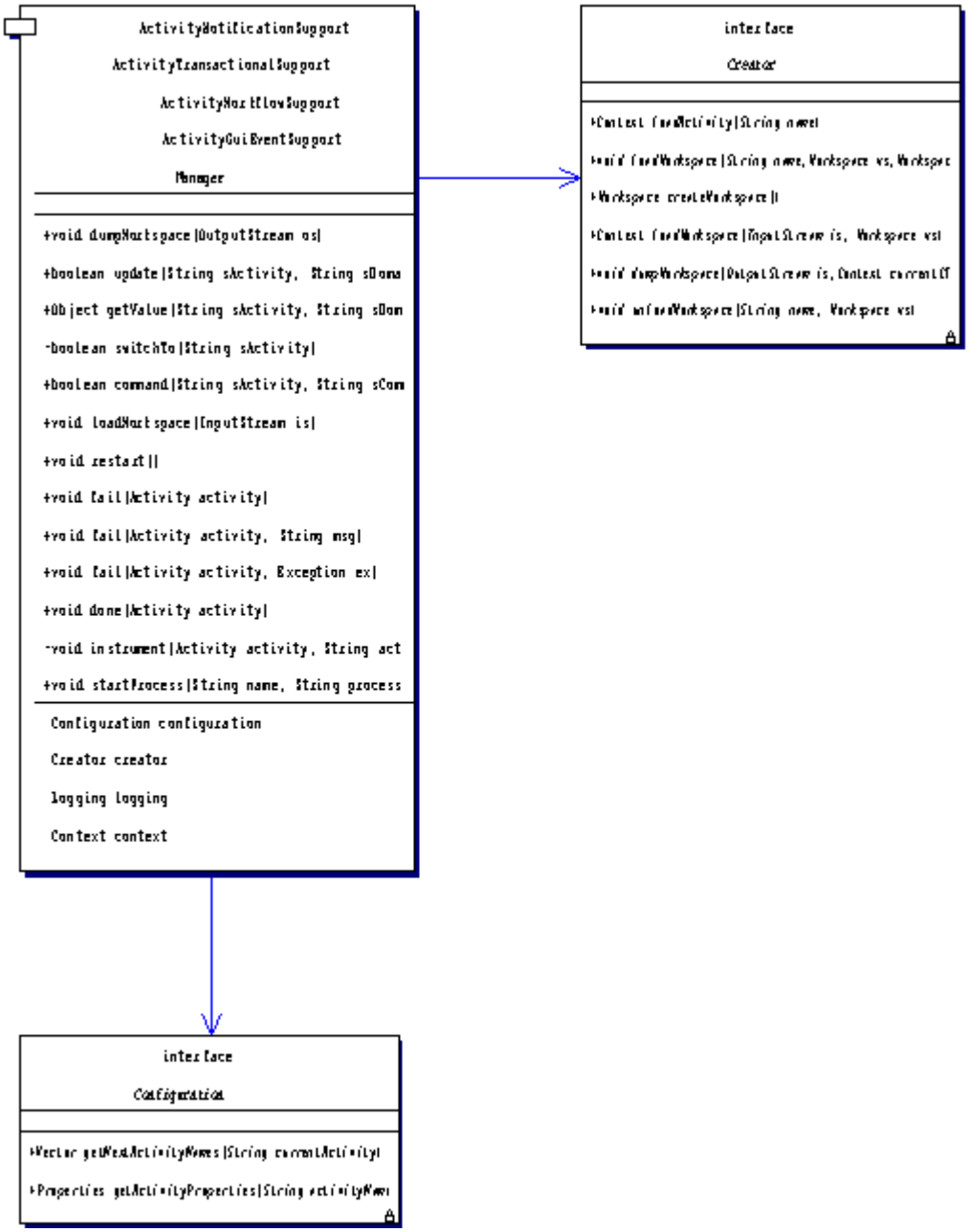


Abbildung 125: Triade (triade.gif)

8.1.2. Begriffsdefinition

Das Activity-Framework stellt eine Ablaufumgebung für in Java-Klassen implementierte Geschäftsregeln dar. Als Ablaufumgebung stellt es Container-Funktionalitäten ähnlich einem EJB-Container einem externen System zur Verfügung. Für die Implementation eines solchen Containers setzen wir ein bestimmtes Design-Pattern ein, das intern als Triade bezeichnet wird.

Eine Triade beschreibt die Kernkomponenten eines Subsystems als eine Gruppe von 3 Klassen :

- einer Engine bzw einem Manager,
- einer Konfiguration

- einer Factory.

Dabei sind die Verantwortlichkeiten dieser 3 Klassen regelmässig genau definiert :

Der Manager erhält von aussen ein auszuwertendes Kommando (Tag bzw ActivityName). Mit diesem Kommando wendet er sich an die Konfiguration, die ihm die Informationen bereitstellt, welche Aktionen er mit diesem Kommando ausführen soll. Die Rückgabe der Konfiguration leitet der Manager an den Creator weiter, der dem Manager die Objektinstanzen (Regeln bzw Activities) zur Verfügung stellt, die der Manager dann zur Ausführung bringt.(*Kommandosequenz*)

Die Konfiguration kennt als einzige die Verbindung zwischen Kommando und VerarbeitungsSequenz, und stellt dem Manager diese Information zur Verfügung. Diese Informationen werden in der Regel extern, z.B. in einer XML-Datei abgelegt.

Der Creator ist der einzige in einem solchen System, der die konkreten Ausprägungen der auszuführenden Klassen kennt.

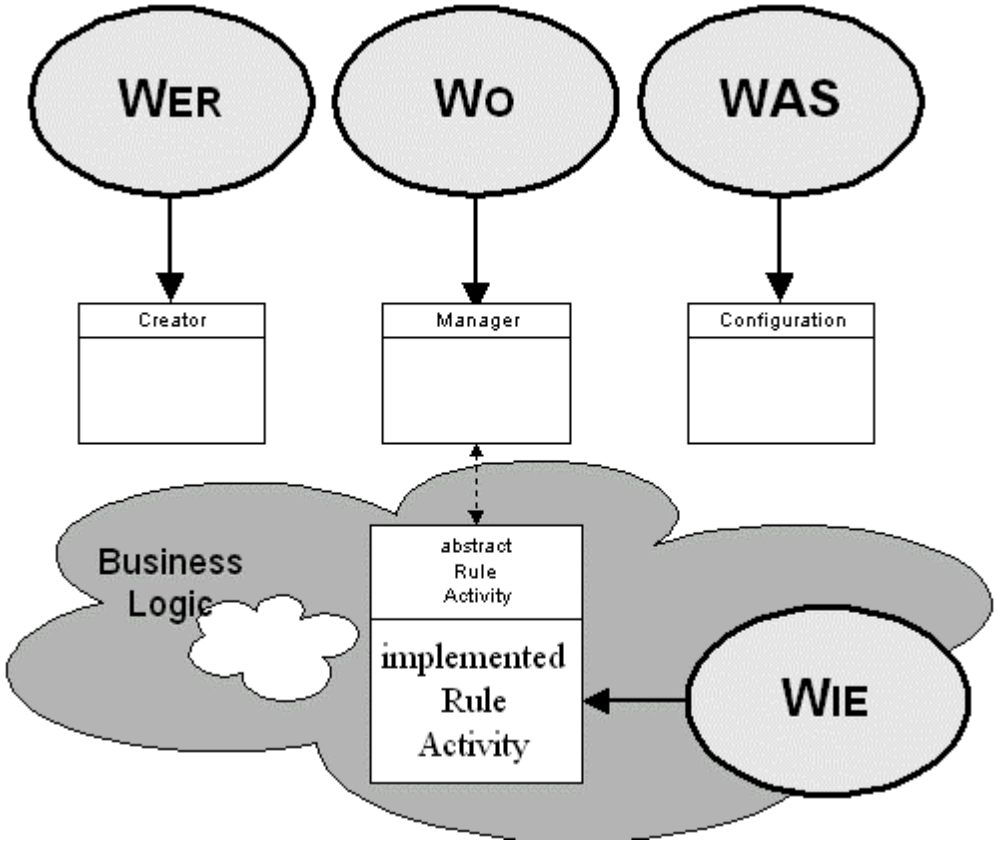


Abbildung 126: Wer Wo Was (werwowas.gif)

Der Manager kennt diese auszuführenden Klassen nur als abstrakte Basisklassen, ist aber in der Lage, über diese Schnittstelle diese so zu konfigurieren (wir verwenden den Begriff „instrumentieren“), dass er diese ausführen kann.

In der Regel ist der Manager eine konkrete Klasse, die vom aussenliegenden Applikationskontext instanziiert wird.

Creator und Configuration sind dem Manager nur als Interfaces bekannt. Sie werden ebenfalls vom aussenliegenden Applikationskontext instanziiert und dem Manager bei der Erzeugung mitgegeben.

Creator und Configuration kennen den Manager nicht.

Die gesamte Fachlichkeit ist in den externen Regeln bzw. Activities abgelegt.

Somit wird eine maximale Trennung von Fachlichkeit, Steuerung und Technik erzielt.

8.1.3. Die Ablaufumgebung

Der Manager des ActFW ist implementiert als ManagerKomponente einer Triade.

Den Auslöser zum Starten einer KommandoSequenz erhält er über 3 Methoden :

```
Manager.update
Manager.command
Manager.startProcess
```

Anhand des übergebenen Parameters kann der Manager die Methoden seiner beigeordneten Configuration aufrufen :

```
Configuration.getNextActivityNames
Configuration.getActivityProperties
```

Mittels der von Configuration zurückgegebenen Daten kann der Manager die Methoden seines beigeordneten Creators aufrufen

```
Creator.createXXX
```

Nach diesen Aufrufen steht dem Manager eine voll konfigurierte und parametrisierte Activity zur Verfügung, die mittels

```
Activity.canStart
Activity.doStart
```

gestartet wird

8.1.4. Die Verwaltung der Nachrichten

Jede Kommunikation mit und innerhalb des ActivityFrameworks findet über Interfaces des Managers statt.

Es gibt konzeptionell keine direkten Durchgriffe von einer Klasse auf eine andere.

Dieses Prinzip mag auf dem ersten Blick ineffektiv und umständlich erscheinen, verhindert aber wirksam überraschende Seiteneffekte.

8.1.5. Das „Innen und Aussen“

Das System ist somit in auf 2 Ebenen geteilt :

- In der Komponenten- bzw Verteilungssicht in die inneren Klassen (Activity, Configuration und Creator) und die Äusseren (die vom Manager exportierten Interfaces)
- In der Implementationssicht in „technische“ bzw. Kernel – Klassen : (Manager, Configuration und Creator) und fachliche (die jeweiligen Activities)

Beide Übergänge (Innen/Aussen) und (Fachlich/Technisch) sind durch definierte Interfaces gekapselt.

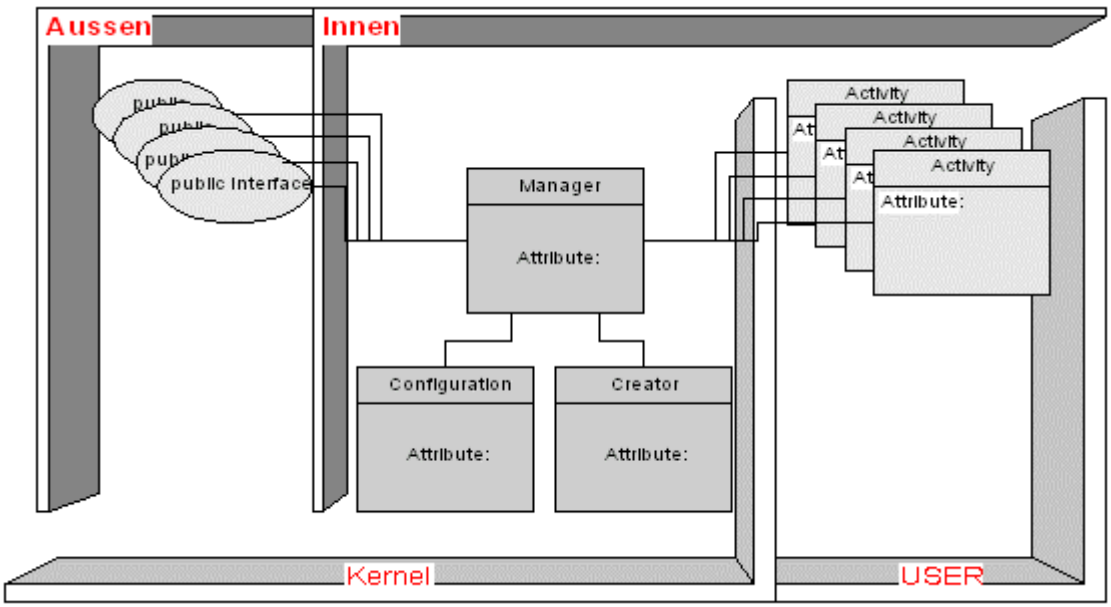


Abbildung 127: Aussen - Innen (ausseninnen.gif)

8.2. Manager

8.2.1. Aufgaben des Managers : Aussensicht

Der Manager erfüllt in der Aussensicht die Aufgabe, sämtliche Schnittstellen nach Aussen zu implementieren. Dadurch wirkt der Manager in einer bestimmten Konfiguration eines Programms nach Aussen wie ein Singleton und wird wie ein solcher angesprochen.

8.2.2. Aufgaben des Managers : Innensicht

Der Manager erfüllt in der Innensicht die Aufgabe, Aktivitäten zu instrumentieren , zu starten und zu beenden. Dabei greift er auf die Informationen und Funktionalitäten vom Configuration und Creator zu.

8.2.3. Aufgaben des Managers : Kernelsicht

Der Manager erfüllt aus Sicht des Kernels die Kommunikation zwischen Aussen und Innen. Von aussen eintreffende Ereignisse werden ausgewertet und Regeln (Activities) in der Innensicht angewendet.

8.2.4. Aufgaben des Managers : Usersicht

Der Manager erfüllt aus Sicht des Users die Kommunikation der fachlichen Regeln mit dem Gesamtsystem. Auch aus Sicht des Usercodes werden Informationen dem System mitgeteilt, auf die das System reagiert. Diese Reaktion erfolgt nach den gleichen Regeln wie das Auswerten von externen Ereignissen.

8.2.5. Die Kontrakte :

Der Manager erfüllt folgende Kontrakte :

- zwischen System und Activity : Reihenfolge der Aufrufe
- zwischen GUI und Activity : Rückgabe des Systemzustandes der Attribute

Diese Kontrakte werden in späteren Kapiteln detailliert erläutert

8.3. Configuration

8.3.1. Aufgaben der Konfiguration : Abstraktion der Hierarchischen Abbildung eines Geschäftsprozesses

Der Manager arbeitet nach einem einfachen Prinzip :

Für ein definierte Ereignis (Kommando) wird eine Reihe von Regeln abgearbeitet.

Die Zuordnung Kommando-> Regel(n) ist dem Manager nicht bekannt, sondern wird in der Configuration vorgenommen. Somit wird die intern als Baum abgebildete Hierarchie der Aktivitäten gegenüber dem Manager nicht sichtbar.

8.3.2. Aufgaben der Konfiguration : Grundlage der Instrumentierung einer Aktivität.

Jede Aktivität besitzt bestimmte Eigenschaften wie z.B. ob sie unterbrechbar und/oder transaktioniert ist, welche Geschäftsobjekte mit dieser Aktivität bearbeitet werden etc... Diese Informationen liefert ebenfalls die Configuration.

8.3.3. Ablage der Informationen

Die Configuration wird dem Manager beim Starten mitgegeben. Somit kann die Configuration ihre Daten grundsätzlich aus jeder beliebigen Form herauslesen. Um ein geregeltes Deployment der Anwendung zu vereinfachen, hat es sich nicht nur im Zusammenhang mit dem ActFW, sondern mit nahezu jedem als Triade implementierten System, als sinnvoll herausgestellt, diese Informationen in XML-Dateien abzulegen.

8.3.4. Konfigurierbarkeit

Durch diese externe Ablage wird das System zu einem zur Laufzeit umkonfigurierbaren Regel-Interpreter, Anwendungen müssen nicht mehr aufwendig neu kompiliert werden, sondern können von aussen, teilweise sogar während das System läuft, um Funktionalitäten ergänzt oder Fehler korrigiert werden.

8.4. Creator

8.4.1. Aufgaben des Creators: Schnittstelle Kernel/User

Der Manager kennt die durch ihn auszuführenden Regeln nur als (weitgehend) abstrakte Basisklasse. Im gesamten System kennt nur der Creator die konkrete Klasse der Geschäftsregel. Diese erzeugt er aufgrund der von Configuration gelieferten Informationen.

8.4.2. Aufgaben des Creators: Workspace-Management

Das der Creator ohnehin in der Lage sein muss, Objektinstanzen abhängig von der Umgebung zu erzeugen, hat er im ActFW zusätzlich die Aufgabe, die Arbeitsdaten der Activities , die Workspaces, zu erzeugen und zu verwalten.

8.5. Activities

8.5.1. Beschreibung der fachlichen Logik

Die Activities beschreiben die fachliche Logik des ActFW. Wie bei der Beschreibung der Triade bereits erläutert, versteht sich das ActFW in Regelinterpreter. Die auszuführenden Regeln sind dem System nicht bekannt, sondern liegen außerhalb des Systems.

Ein Geschäftsprozess wird in diesem System als eine Menge von Regeln verstanden, die der Manager aufgrund definierter Ereignisse abarbeitet.

Diese Regeln werden vom Benutzer des ActFW implemetiert. Als Implementationssprache wurde hier genau wie im gesamten System JAVA gewählt, die erstellten Regel-Klassen (Activities) laufen im Kontext der Anwendung.

Diese Regeln/Activities beinhalten :

- fachlich definierte Vorbedingungen
- fachlich definierte Nachbedingungen
- fachliche Ablauflogik

8.5.2. Fachliche Ablauflogik

Die fachliche Ablauflogik wie z.B. Der Benutzer hat den Button XY gedrückt, deshalb muss ein neuer Auftrag gedruckt werden, oder der Benutzer hat das Geburtsdatum geändert, das Endalter muss neu berechnet werden, sind innerhalb der jeweiligen Activity implementierte Geschäftsregeln.

Jede dieser Geschäftsregeln wird aufgrund eines bestimmten Ereignisses, eines Kommandos oder einer Zustandsänderung eines Attributs ausgelöst.

8.5.3. Datenhaltung

Jeder Activity wird bereits bei der Definition als Usecase eine bestimmte, genau definierte Menge an Geschäftsobjekten zugeordnet. Nur diese Geschäftsobjekte stehen im Zugriff einer Activity. Der Datencontainer oder Namingservice, über den diese Objektinstanzen verwaltet werden, wird lokaler Workspace genannt. Dieser lokale Workspace steht nicht direkt im Zugriff der Activity, sondern wird intern von der Activity verwaltet.

Die Vereinigungsmenge aller dieser lokalen Workspaces bildet den globalen Workspace, der dem gesamten Geschäftsprozeß zugeordnet ist.

8.5.4. Passive Ablaufsteuerung

Die API jeder Activity zur internen Abbildung der Ablaufsteuerung besteht aus 4 zu überschreibenden Methoden :

```
-- canStart
-- doStart
-- canComplete
-- doComplete
```

Hierbei werden die modellierten Preconditions als Java-Code in der Methode `canStart` implementiert. Ein Rückgabewert von TRUE bedeutet, dass die Vorbedingungen erfüllt sind.

Die PostConditions werden in der Methode `canComplete` definiert. Ein Rückgabewert von TRUE bedeutet, dass die Nachbedingungen erfüllt sind, d.H. die Aktivität kann beendet werden.

Die Regeln der Geschäftslogik werden in der Methode `doStart` implementiert,

Das Aufbauen des konsistenten Zustandes beim Beenden, (die Zusicherung) wird in der Methode `doComplete()` implemetiert.

8.5.5. Implementation der Vorbedingungen und Nachbedingungen , Datenzugriffe

Die Zugriffe auf diese Workspaces erfolgen nur innerhalb der jeweiligen Activity ohne Zugriffe auf externe Systeme oder Services. Das Lesen von Objekten aus dem Workspace erfolgt mittels Aufruf von

`Get(„Logischer_Name“)` wobei logischer Name der im Usecase definierte Eintrag im Workspace ist.

Das Einstellen von Daten in den Workspace erfolgt mittels `put(„Logischer_Name“, Object)`wobei logischer Name der im Usecase definierte Eintrag im Workspace ist.

Das Abfragen von Objekten erfolgt mittels `iterator ()`;

Diese Einträge werden „Slot“ genannt.

8.5.6. Implementation der Geschäftsregeln

Geschäftsregeln werden analog dem Java-Beans-Pattern als Property-, Action bzw. VetoableChangeListener implementiert. Diese werden in der Regel als Inner Classes der Activity angelegt. Somit steht innerhalb dieser Listener die gesamte API der Activity zur Verfügung.

Listener hören auf Attributänderungen und Kommandos . Diese Attributänderungen werden beschrieben als „Geschäftsobjekt-“ bzw. „Slotname“ und „Attributname“. Wann immer eine Änderung eines Attributs von aussen erfolgt, bzw das Kommando von aussen getriggert wird, so werden die registrierten Listener aufgerufen.

8.5.7. Design

Diese groben Usecases werden im Verlauf der nächsten Phase, des Designs, weiter in kleinere Usecases zergliedert, (dekomponiert) .

Dieser Vorgang definiert

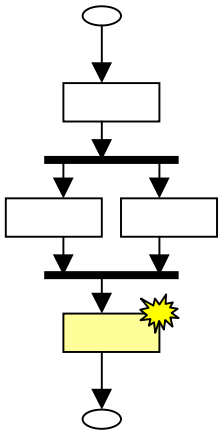
- die Abfolge der dekomponierten Usecases als Elemente eines Activity-Diagramms
- zu jedem dieser Usecases :
  - die fachlich beschriebenen, an genau diesem Vorgang beteiligten Geschäftsobjekte
- die Vorbedingungen, unter denen der Vorgang starten kann
- die Nachbedingungen, unter denen die Verarbeitung beendet werden kann
- den Systemzustand beim Verlassen der Verarbeitung des Usecases
- die genaue Beschreibung der fachlichen Regeln (Geschäftslogik)
- Zu jedem Element des Activity-Diagramms entsteht genau ein Usecase.
- Gleichzeitig wird aus der Vereinigungsmenge der an den Aktivitäten beteiligten Geschäftsobjekte das (fachliche) Objektmodell gebildet.

Sobald einzelne Aktivitäten in dieser Form definiert sind, so können diese implementiert werden.

8.5.8. Implementation

Das Activity-Subsystem von DSE implementiert in seiner API eine weitgehende 1:1 Umsetzung der Ergebnisse des Designprozesses.

Schritt 1.) Das Activity-Diagramm wird in ein XML-Format übersetzt :

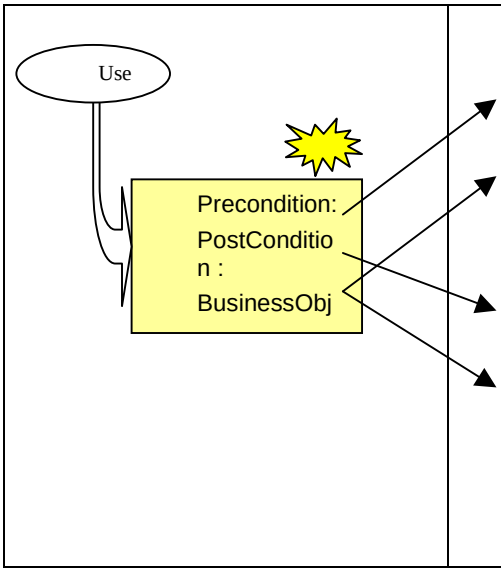


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ActivityFlow SYSTEM "activity.configuration.dtd">
<ActivityFlow>
    <Activity name="root" class="de.dekra.bp.minihu.root"
                start="Activity1" >
                <Transitions>
                        <Transition>
                                <Start activity="Activity1"/>
                                <End activity ="Activity2"/>
                                <End activity ="Activity3"/>
                                <End activity ="Activity4"/>
                                <End activity ="Activity5"/>
                        </Transition>
    <Activity name="Activity2"
                class="de.dekra.bp.minihu.Activity2"
                start="Activity1">
                <DomainObjects>
                        <DomainObject name="Person"
                class="de.dekra.db.minihu.Person"/>
                </DomainObjects>
                <Transitions>
                        <Transition>
                                <Start activity="SubActivity1"/>
                                <End activity ="SubActivity2"/>
                                <End activity ="SubActivity3"/>
                        </Transition>
                </Transitions>
    </Activity>
</ActivityFlow>
```

Abbildung 128. Activity Diagramm als XML

In diesem Format werden die Übergänge der Activities beschrieben : (Transitionen / `Transition`) sowie die an dem Usecase / der Activity beteiligten Domainobjekte (`DomainObject`).

Schritt 2.) Jeder Usecase wird als abgeleitete Instanz der Klasse Activity abgebildet:




```
public Class MyActivity extends Activity
{
   public boolean canStart()
   {
         ...
   }
   public void doStart()
   {
         ...
   }
   public boolean canComplete()
   {
         ...
   }
   public boolean doComplete()
   {
         ...
   }
}
```

Abbildung 129. UseCase als Activity

8.5.9. Implementation einer einzelnen Activity

8.5.9.1. Einordnung einer einzelnen Aktivität in den Gesamtkontext

Activity-Instanzen sind als atomar zu verstehen:

- sie starten mit einem konsistenten Zustand und hinterlassen beim Verlassen einen solchen.
- Activities sind grundsätzlich nicht unterbrechbar.

- Activities kennen nur den ihnen in der Modellierung zugeordneten Datenbereich.
- Alle Plausibilitäten und Geschäftsregeln werden nur in diesem Kontext abgearbeitet.
- Keine Activity greift auf eine andere zu, die Übermittlung der Ergebnisse erfolgt nur über den gemeinsamen Datenbereich.

8.5.9.2. Passive Ablaufsteuerung

Die API jeder Activity zur internen Abbildung der Ablaufsteuerung besteht aus 4 zu überschreibenden Methoden :

```
-- canStart
-- doStart
-- canComplete
-- doComplete
```

Hierbei werden die modellierten Preconditions als Java-Code in der Methode `canStart` implementiert. Ein Rückgabewert von TRUE bedeutet, dass die Vorbedingungen erfüllt sind.

Die PostConditions werden in der Methode `canComplete` definiert. Ein Rückgabewert von TRUE bedeutet, dass die Nachbedingungen erfüllt sind, d.H. die Aktivität kann beendet werden.

Die Regeln der Geschäftslogik werden in der Methode `doStart` implementiert,

Das Aufbauen des konsistenten Zustandes beim Beenden, (die Zusicherung) wird in der Methode `doComplete()` implemetiert.

8.5.9.3. Aktive Eingriffe in die Ablaufsteuerung

Jede Activity kann sich selbständig beenden. Wenn eine Activity bei der Abarbeitung ihrer Geschäftslogik feststellt, daß sie nichts weiter zu tun braucht, so kann sie die Methode `done` aufrufen. Damit wird von aussen die Abschluß-Sequenz von `canClose`() und `doClose()` aufgerufen. Wenn eine Activity bei der Abarbeitung ihrer Geschäftslogik feststellt, dass sie nicht weiter sinnvoll arbeiten kann, z.B. wenn ihre internen Daten völlig zerstört sind, oder ein bestimmter Service nicht zur Verfügung steht, so kann sie die Methode `failed` aufrufen. Damit wird von aussen die Aktivität sofort beendet. Die Abschluß-Sequenz von `canClose`() und `doClose()` wird nicht aufgerufen. Alle lokal in der Aktivität vorbereiteten Änderungen werden verworfen. Als Folgeaktivitäten werden die in der Konfigurationsdatei unter dem Tag abend activity = Activity name abgearbeitet.

8.5.9.4. Datenhaltung

Jeder Activity wird bereits bei der Definition als Usecase eine bestimmte, genau definierte Menge an Geschäftsobjekten zugeordnet. Nur diese Geschäftsobjekte stehen im Zugriff einer Activity. Der Datencontainer oder Namingservice, über den diese Objektinstanzen verwaltet werden, wird lokaler Workspace genannt. Dieser lokale Workspace steht nicht direkt im Zugriff der Activity, sondern wird intern von der Activity verwaltet.

Die Vereinigungsmenge aller dieser lokalen Workspaces bildet den globalen Workspace, der dem gesamten Geschäftsprozeß zugeordnet ist.

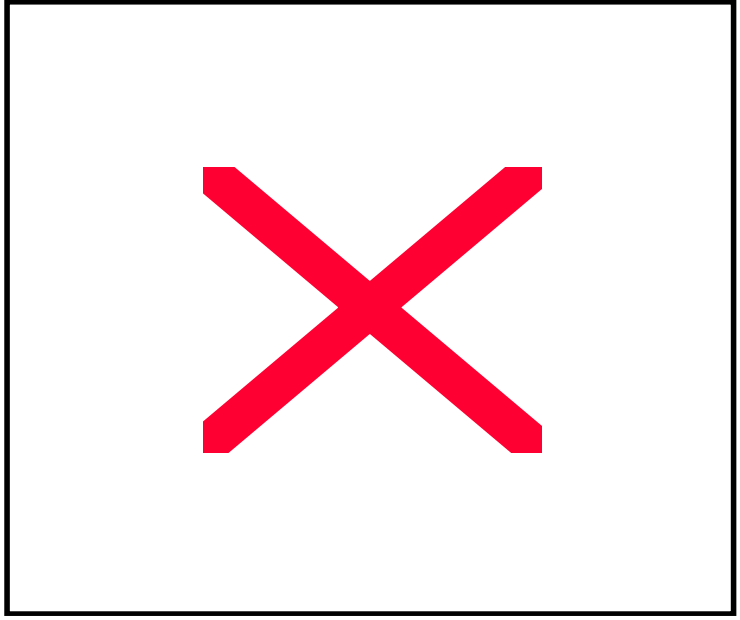


Abbildung 130. Lokale und globale Workspaces

8.5.9.5. Implementation der Vorbedingungen und Nachbedingungen , Datenzugriffe

Die Zugriffe auf diese Workspaces erfolgen nur innerhalb der jeweiligen Activity ohne Zugriffe auf externe Systeme oder Services.

Das Lesen von Objekten aus dem Workspace erfolgt mittels Aufruf von

`Get(„Logischer_Name“)` wobei logischer Name der im Usecase definierte Eintrag im Workspace ist.

Das Einstellen von Daten in den Workspace erfolgt mittels `put(„Logischer_Name“, Object)`wobei logischer Name der im Usecase definierte Eintrag im Workspace ist.

Das Abfragen von Objekten erfolgt mittels `iterator ()`;

Diese Einträge werden „Slot“ genannt.

8.5.9.6. Lebensdauer des Workspaces

Der Workspace wird vor canStart() der Activity zur Verfügung gestellt. Die Änderungen werden nach canCommit in den globalen Workspace eingestellt.

Wird die Activity nicht ordnungsgemäß beendet, so findet das Einstellen in den Workspace nicht statt, die Ergebnisse werden verworfen.

8.5.9.7. Definition der Geschäftsregeln

Geschäftsregeln werden analog dem Java-Beans-Pattern als Property- bzw. VetoableChangeListener implementiert. Da diese in der Regel als Inner Classes der Activity angelegt werden. Somit steht innerhalb dieser Listener die gesamte API der Activity zur Verfügung.

Listener hören auf Attributänderungen. Diese Attributänderungen werden beschrieben als „Geschäftsobjekt-“ bzw. „Spotname“ und „Attributname“. Wann immer eine Änderung eines Attributs von aussen erfolgt, so werden die registrierten Listener aufgerufen.

Ein Beispiel für solche Geschäftsregeln ist :

<< wenn sich das Geburtsdatum ändert, so ist das Endalter neu zu berechnen und zu prüfen, ob ein Führerschein bei dem neuen Endalter möglich ist, ist das Endalter kleiner 18 so ist Führerschein auf „nicht vorhanden“ zu setzen >>.

Im Code registriert man einen PropertyChangeListener auf das Geburtsdatum. In dessen PropertyChange ruft man die Routine berechneEndalter auf. Ist deren Ergebnis < 18 , so wird das Attribut „hasFuehrerschein“ auf „false“ gesetzt.

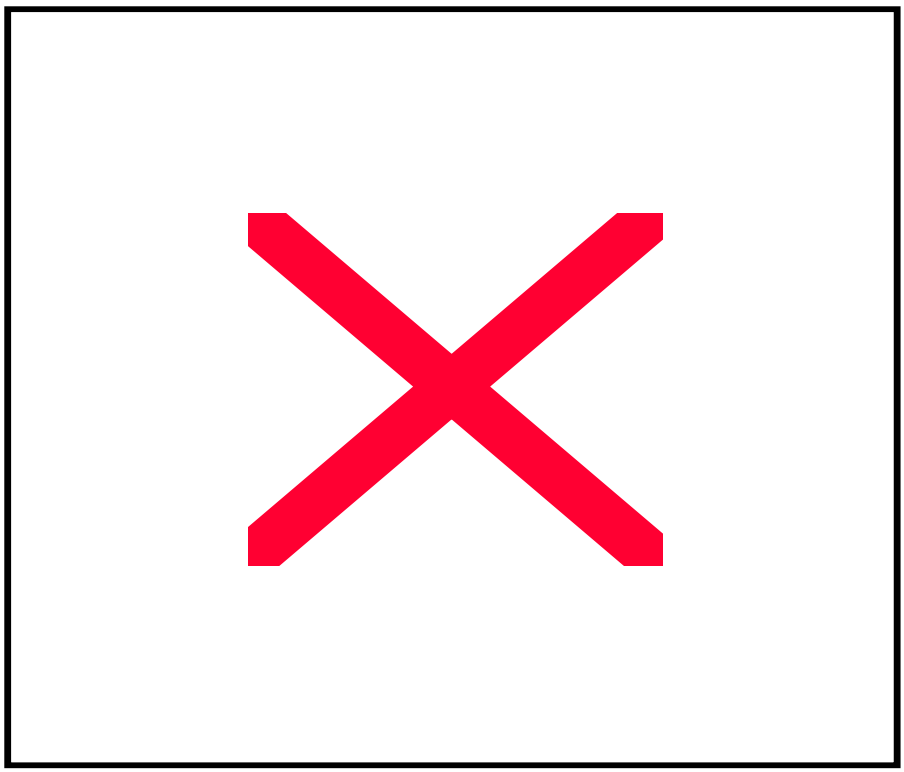


Abbildung 131. Lebensdauer eines lokalen Workspaces

8.6. Kommunikation Manager – Activity

8.6.1. Manager -> Activity

8.6.1.1. Instrumentierung

Bei der Instrumentierung übergibt der Manager der vom Creator instanziierten Activity den Workspace und den ActivityNotificationSupport über ganz normale Setter-Methoden:

```
        Workspace wsA = creator.createWorkspace();
  creator.loadWorkspace(oldActivityName, wsA,ws );
        activity.setName(activityName);
        activity.setWorkspace(wsA);
        activity.setActivityNotificationSupport(this);
```

8.6.1.2. Startsequenz

Nach erfolgreicher Instrumentierung startet der Manager die Startsequenz einer Activity :

```
    if  (activity.canStart())
    {
                activity.doStart();
}
```

8.6.1.3. Endsequenz

Aufgrund bestimmter Ereignisse startet der Manager die Endsequenz einer Activity :

```
    if  (activity.canComplete())
    {
                activity.doComplete();
}
```

8.6.1.4. Übergang zwischen 2 Activities :

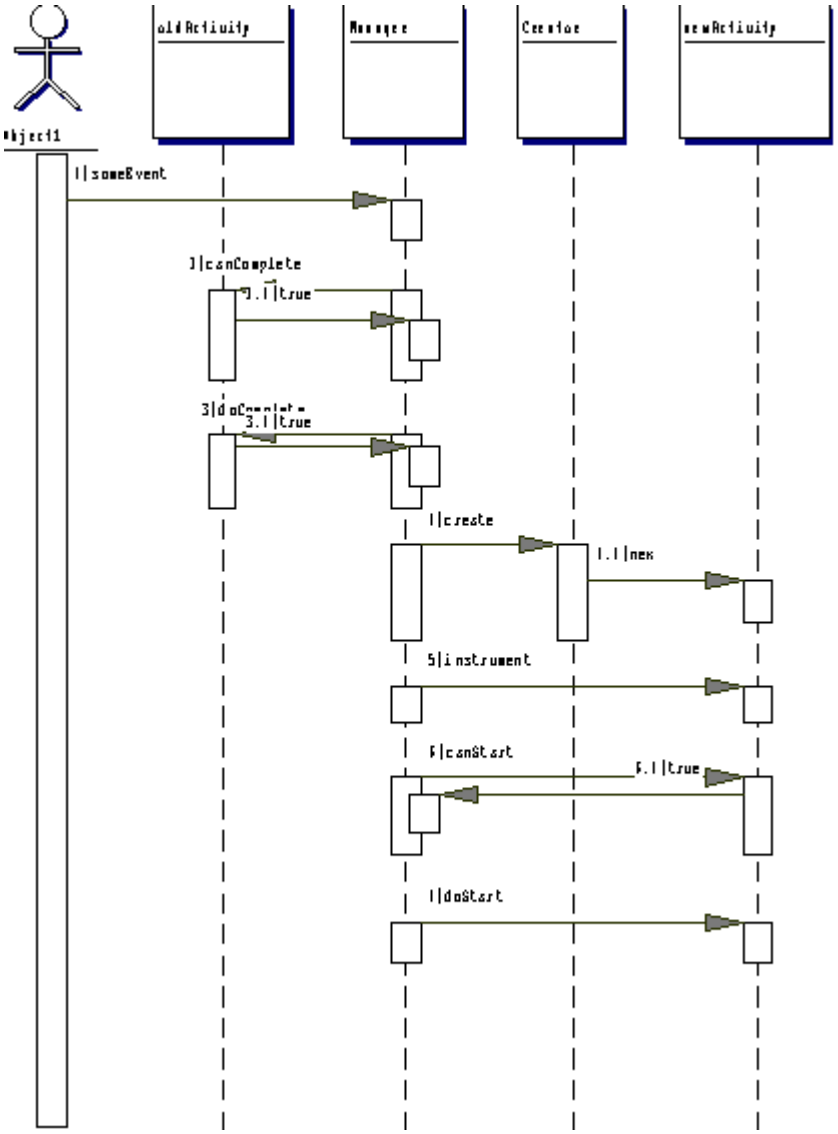


Abbildung 132: Übergang (übergang.gif)

8.6.2. Activity -> Manager

Die jeweilige Activity hat keine Schnittstellen zum Manager, es besteht kein „legaler“ Weg, auf den Manager zuzugreifen.

Die Activity kommuniziert mit dem Manager auf 2 Wegen :

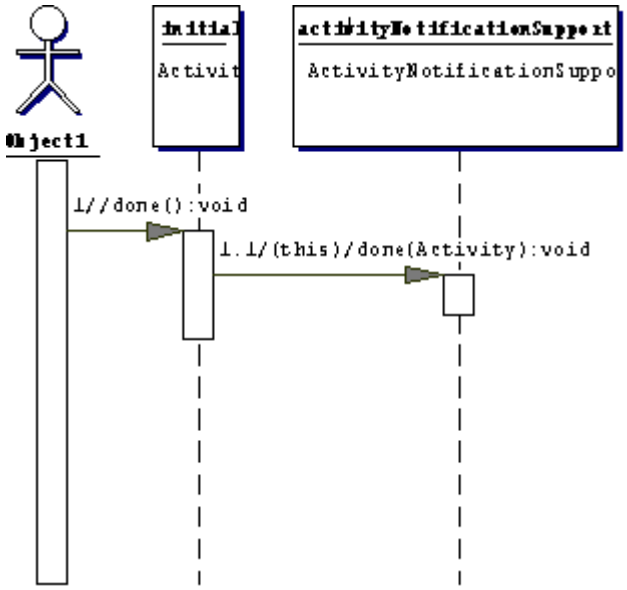


Abbildung 133: Activity done (activity.done.gif)

8.6.2.1. Indirekt

Indirekt über das ActivityNotificationSupport Interface : jede der 4 Methoden (done und 3 * fail) von Activity wird auf den der Activity übergebenen ActivityNotificationsupport umgelenkt.,

Dieser ActivityNotificationsupport **ist** der Manager.

`done()` die Activity wünscht sich zu beenden

`failed(...)` die Activity ist auf einen Fehler gestossen und möchte abgebrochen werden

8.6.2.2. Direkt

Direkt über die Rückgabewerte der Callbacks :

```
canStart()
doStart()
canComplete()
canComplete(...)
doComplete()
```

Die genaue Syntax und Semantik dieser Methoden wird im Kapitel „Technische Beschreibung der Implementation des ActFW“ geliefert.

8.7. Transitions

8.8. Beschreibung der XML-Konfiguration

8.8.1. Die DTD (Document Type Definition)

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT ActivityFlow (Activity* , Userinterface*)>
<!ELEMENT Activity (Transitions, DomainObjects)>
<!ATTLIST Activity name CDATA #REQUIRED>
<!ATTLIST Activity class CDATA #IMPLIED>
<!ATTLIST Activity start CDATA #IMPLIED>
<!ATTLIST Activity userinterface CDATA #IMPLIED>
<!ATTLIST Activity exclusive CDATA #IMPLIED>
<!ATTLIST Activity transacted CDATA #IMPLIED>
<!ELEMENT Userinterface EMPTY>
<!ATTLIST Userinterface name CDATA #REQUIRED>
<!ATTLIST Userinterface class CDATA #REQUIRED>
<!ELEMENT Transitions (Transition*)>
<!ELEMENT DomainObjects (DomainObject*)>
<!ELEMENT Transition (Start, End*)>
<!ELEMENT Start EMPTY>
<!ATTLIST Start activity CDATA #REQUIRED>
<!ATTLIST End activity CDATA #REQUIRED>
<!ELEMENT DomainObject EMPTY>
<!ATTLIST DomainObject name CDATA #REQUIRED>
<!ATTLIST DomainObject class CDATA #REQUIRED>
<!ATTLIST DomainObject keep CDATA #IMPLIED>
```

8.8.2. Die Elemente

Das Root-Objekt einer jeden XML-Konfiguration ist das Element ActivityFlow :

Es teilt die Konfiguration in 2 Hauptbereiche :

Die Beschreibung der Aktivitäten und die Beschreibung der Userinterfaces (aus Sicht der Activity-Frameworks)

8.8.3. Beschreibung einzelner Aktivitäten

Die XML-Konfiguration beschreibt Geschäftsprozesse, d. h. Bäume von Aktivitäten. Dargestellt wird dieses Modell als Activity-Einträge, die weitere Activities enthalten (Geschäftsprozesse) und Activity-Einträge, die keine weiteren Activities enthalten, dies sind die zu implementierenden Activity-Instanzen. Des weiteren können fehlgeschlagene Activities weitere Activities aktivieren, welche auf Fehler reagieren, diese Activities können ebenfalls Activity-Instanzen enthalten.

8.8.3.1. Beispiel :

```
<Activity name="root" class="de.dekra.bp.minihu.root" start="Activity2" >
                <Transitions>
                      <Transition>
                            <Start activity="Activity2"/>
                            <End activity ="Activity4"/>
                            <End activity ="Activity5"/>
                            <abend activity = "Fehler"/>
                      </Transition>
                </Transitions>
                <DomainObjects>
                      <DomainObject name="Person"
class="de.dekra.db.minihu.Person"/>
                      <DomainObject name="Adresse"
class="de.dekra.db.minihu.Adresse"/>
```

```
                </DomainObjects>
    </Activity>
```

8.8.3.2. Name des Elements :

<Activity>

8.8.3.3. Attribute :

Name	Der Name der Aktivität	obligatorisch
Class	Die zu implementierende Java-Klasse	bei Activities, die keine weiteren Activities enthalten obligatorisch, sonst untersagt
Start	Die in ihr enthaltene, als erste zu startende Aktivität, diese muss deklariert sein	bei Activities, die weitere Activities enthalten obligatorisch, sonst untersagt
Abend	Ist eine Aktivität fehlgeschlagen, wird eine Liste abgearbeitet, welche Fehlerausgänge beschreibt. Diese Fehlerausgänge wiederum werden durch eine Aktivität definiert. Diese Fehleraktivitäten werden durch das Abendtag in der XML Datei deklariert. Der Eintrag abend = IGNORE bedeutet, dass die normalen Ausgänge (END = XXX) auch im Fehlerfall benutzt werden.	Optional , gültige Werte: abend = IGNORE, dieser Tag stellt einen Ausnahmefall dar.
Transacted	NEW_CONTEXT : Die Aktivität erhält vor canStart einen eigenen Transaktionskontext. PARENT_CONTEXT : Die Aktivität erhält vor canStart den Transaktionskontext des darüberliegenden Systems FALSE : die Aktivität ist nicht transaktioniert	Optional, default = FALSE
Exclusive	TRUE : die Aktivität muss beendet werden, wenn zu einer beliebigen anderen umgeschaltet wird, sie kann dieses aber verweigern. Es erfolgt kein Aufruf von canComplete(SLEEPONLY) ACTIVITY : die Aktivität muss beendet werden, wenn zu einer anderen desselben Namens	Optional, bei Activities, die weitere Activities enthalten untersagt

	umgeschaltet wird, sie kann dieses aber verweigern. Beim Umschalten erfolgt kein Aufruf von canComplete(SLEEPONLY) FALSE : default Einstellung . Die Activity kann eingefroren werden. KEY : die Aktivität muss beendet werden, wenn zu einer anderen mit den identischen Geschäftsobjekten umgeschaltet wird, sie kann dieses aber verweigern. Beim Umschalten erfolgt kein Aufruf von canComplete(SLEEPONLY)	
Userinterface	Name der Viewport-Klasse	(optional, bei Activities, die weitere Activities enthalten untersagt

8.8.3.4. Enthaltene Elemente :

Transitions	Übergänge zwischen den enthaltenen Aktivitäten	bei Activities, die keine weiteren Activities enthalten untersagt
DomainObjects	Die der Activity zugeordneten DomainObjekte, „Slots“	optional

8.8.4. Beschreibung der Transitionen , Zustandsübergänge

Deklarierte Aktivitäten besitzen optional eine Menge von Transitionen, mittels denen die Übergänge zwischen den Aktivitäten beschrieben werden. Diese sind abgelegt als eine Folge von „Transition“ einträgen.

8.8.4.1. Beispiel :

```
<Activity name="root" class="de.dekra.bp.minihu.root" start="Activity2" >
                <Transitions>
                      <Transition>
                            <Start activity="Activity2"/>
                            <End activity ="Activity4"/>
                            <End activity ="Activity5"/>
                            <abend activity ="Fehler"/>

                      </Transition>
                </Transitions>
</Activity>
```

8.8.4.2. Name des Elements:

<Transition>

8.8.4.3. Attribute :

-.-

8.8.4.4. Enthaltene Elemente :

Start activity	Definition der Aktivität, für deren beenden eine Regel abgelegt wird	Attribut,: Name der Aktivität
Abend activity	Im Fehlerfall sind 1 oder mehrere Einträge von Folgeaktivitäten möglich, die in der Reihenfolge mittels canStart auf potentielle Startfähigkeit abgearbeitet werden . Der Tag Abend activity = IGNORE, stellt einen Ausnahmefall dar.	Attribut; Name der Aktivität
End	Dieser Eintrag beendet den Geschäftsprozess und ist gleichbedeutend mit dem Tag End Activity = „END“.	Attribut
End activity	1 oder mehrere Eintrage von möglichen Folgeaktivitäten, die in der Reihenfolge mittels canStart auf potentielle Startfähigkeit abgearbeitet werden, in der sie in der Konfigurationsdatei abgelegt sind.	Attribut,: Name der Aktivität Der spezielle Name der Aktivität END bedeutet das Ende des Geschäftsprozesses.

Grund für diese Benennung ist, dass die Transitionen als eine Abbildung $a \rightarrow \begin{pmatrix} b \\ c \\ d \end{pmatrix}$ mit a als Start und b,c,d als potenzielle Nachfolger verstanden wird.

8.8.5. Beschreibung der Domainobjekte

Deklarierte Aktivitäten besitzen optional eine Menge von Domainobjekteinträgen, Slots. Diese werden unter DomainObjects als Elemente vom Typ DomainObject abgelegt.

8.8.5.1. Beispiel :

```
<Activity name="root" class="de.dekra.bp.minihu.root" start="Activity2" >
                <Transitions>
                      <Transition>
                            <Start activity="Activity2"/>
                            <End activity ="Activity4"/>
                            <End activity ="Activity5"/>
                            <abend activity = "Fehler"/>
                      </Transition>
                </Transitions>
                <DomainObjects>
                      <DomainObject name="Person"
                class="de.dekra.db.minihu.Person"/>
                      <DomainObject name="Adresse"
                class="de.dekra.db.minihu.Adresse"/>
                </DomainObjects>
</Activity>
```

8.8.5.2. Name des Elements

DomainObject

8.8.5.3. Attribute :

Name	der Name des Slots	Obligatorisch
Class	die das Objekt implementierende Java-Klasse	Obligatorisch

Keep	TRUE/FALSE Wird ein Geschäftsprozeß neu gestartet, so wird ein im gerade beendeten Geschäftsprozeß eventuell vorhandener Slot gleichen Namens in den Workspace des neuen übernommen	Optional

8.8.6. Beschreibung der Userinterface-Elemente

Activity-Einträgen kann optional ein Userinterface-Eintrag mitgegeben werden, der den ViewPort (als Teil des MVC-Frameworks) und damit die GUI beschreibt. Diese Einträge müssen im Userinterfaces-Teil deklariert werden als eine Liste von Userinterface-Einträgen .

8.8.6.1. Beispiel

```
<Userinterface name="View1" class ="de.dekra.ui.minihu.View1"/>
<Userinterface name="View2" class ="de.dekra.ui.minihu.View2"/>
```

8.8.6.2. Name des Elements

Userinterface

8.8.6.3. Attribute :

Name	der Name des UI-Elements, der zum Activity-Eintrag deklariert wurde	Obligatorisch
Class	die das Objekt implementierende Java-Klasse	Obligatorisch

8.9. Workspace

8.9.1. Definition

Jeder Activity wird bereits bei der Definition als Usecase eine bestimmte, genau definierte Menge an Geschäftsobjekten zugeordnet. Nur diese Geschäftsobjekte stehen im Zugriff einer Activity. Der Datencontainer oder Namingservice, über den diese Objektinstanzen verwaltet werden, wird lokaler Workspace genannt. Dieser lokale Workspace steht nicht direkt im Zugriff der Activity, sondern wird intern von der Activity verwaltet.

Die Vereinigungsmenge aller dieser lokalen Workspaces bildet den globalen Workspace, der dem gesamten Geschäftsprozeß zugeordnet ist.

Die Geschäftsprozesse bilden die Activities als einen Baum ab, bei dem jedes Blatt genau einer Activity zugeordnet ist, jeder Knoten einem Geschäftsprozess, der selber Activities enthält[1].

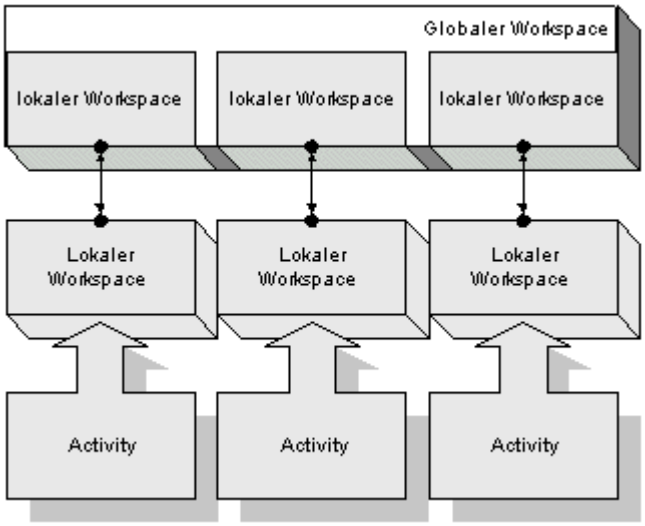


Abbildung 134: Globaler Workspace (globalerworkspace.gif)

Jeder Activity und jedem Geschäftsprozess ist genau eine WorkspaceDefinition zugeordnet, die bei Aktivieren der Activity benutzt wird, um einen echten Workspace zu instanziieren.

Der oberste Workspace wird Masterworkspace genannt.

Der Workspace verwaltet Objekte, die über logische Namen ansprechbar sind. Diese logischen Namen entsprechen den beim Deployment der Configuration mitgeteilten Namen.

Diese (ggfs. leeren) Einträge werden Slots genannt.

Beim Zuordnen eines Workspaces (Creator.loadWorkspace(....)) zu einer Activity wird vom Creator jeder Slot des darüberliegenden Workspaces in den neuen Workspace kopiert. Die in den Slots enthaltenen Objekte sind nicht identisch mit denen des darüberliegenden.(logische Schnittmengenoperation)

Beim Beenden der Activity werden die Elemente des lokalen Workspaces in der darüberliegenden zurückkopiert

```
Creator.unloadWorkspace(....))(logische Vereinigungsmengenoperation)
```

Sobald das System unter einem Transaktionsmonitor läuft, vereinfacht sich diese Operation, indem der Transaktionsmonitor die Kopieroperationen übernimmt. Für den Creator stellt sich dies als Zuordnen und Commiten eines Transaktionskontexts dar.

[1] in der Regel wird von diesem Feature allerdings kein Gebrauch gemacht, es dient lediglich dazu, präventive Bedenkenträger, die unbedingt etwas kompliziertes erwarten, zu befriedigen. Der Standardfall ist : ein Geschäftsprozess und n Activities.

8.9.2. API (von Creator)

Der Manager selbst hat keine Schnittstellen zum Workspace, die Erzeugung und Konfiguration ist Verantwortlichkeit des Creators.

Der Creator beinhaltet 4 Schnittstellen zum Workspace :

8.9.2.1. Erzeugen und Zusammenstellen eines Workspaces

```
public Workspace createWorkspace();
```

erzeugt einen neuen, leeren Workspace

```
public void loadWorkspace(String name,Workspace ws,Workspace parent,Properties prop);
```

führt die Schnittmengenoperation vom ParentWorkspace zum aktuellen Workspace aufgrund der Informationen in prop durch

```
public void unloadWorkspace(String name,Workspace ws,Workspace parent,Properties
                prop);
```

führt die Vereinigungsmengenoperation vom aktuellen Workspace zum ParentWorkspace aufgrund der Informationen in prop durch

8.9.2.2. Persistenz-Support

```
public Context loadWorkspace(InputStream is, Workspace ws);
public void dumpWorkspace(OutputStream is,Context currentCTX);
```

Schreibt einen Workspace in einen Stream bzw lädt einen Workspace aus einem Stream. Der Systemzustand wird in einem Context abgelegt und daraus wieder restauriert.

8.9.3. API (von Activity)

Die Zugriffe auf diese Workspaces erfolgen nur innerhalb der jeweiligen Activity ohne Zugriffe auf externe Systeme oder Services.

Das Lesen von Objekten aus dem Workspace erfolgt mittels Aufruf von

```
Get(„Logischer_Name“)
```

wobei logischer Name der im Usecase definierte Eintrag im Workspace ist.

Das Einstellen von Daten in den Workspace erfolgt mittels

```
put(„Logischer_Name“, Object)
```

wobei logischer Name der im Usecase definierte Eintrag im Workspace ist.

Das Abfragen von Objekten erfolgt mittels

```
iterator ();
```

Der Workspace wird vor

```
canStart()
```

der Activity zur Verfügung gestellt.

Die Änderungen werden nach

```
canCommit
```

in den globalen Workspace eingestellt.

Wird die Activity nicht ordnungsgemäß beendet, so findet das Einstellen in den Workspace nicht statt, die Ergebnisse werden verworfen.

8.10. Slots

8.11. Transactions (committing, persistence, etc.)

8.12. SDI

8.13. Services

9. Tools

9.1. Tool Center

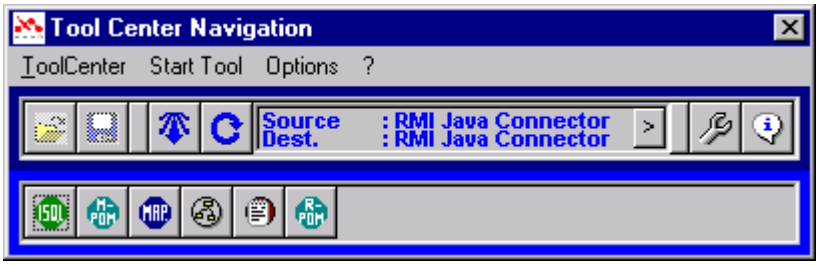


Abbildung 135. Tool Center

9.2. Object Model Browser

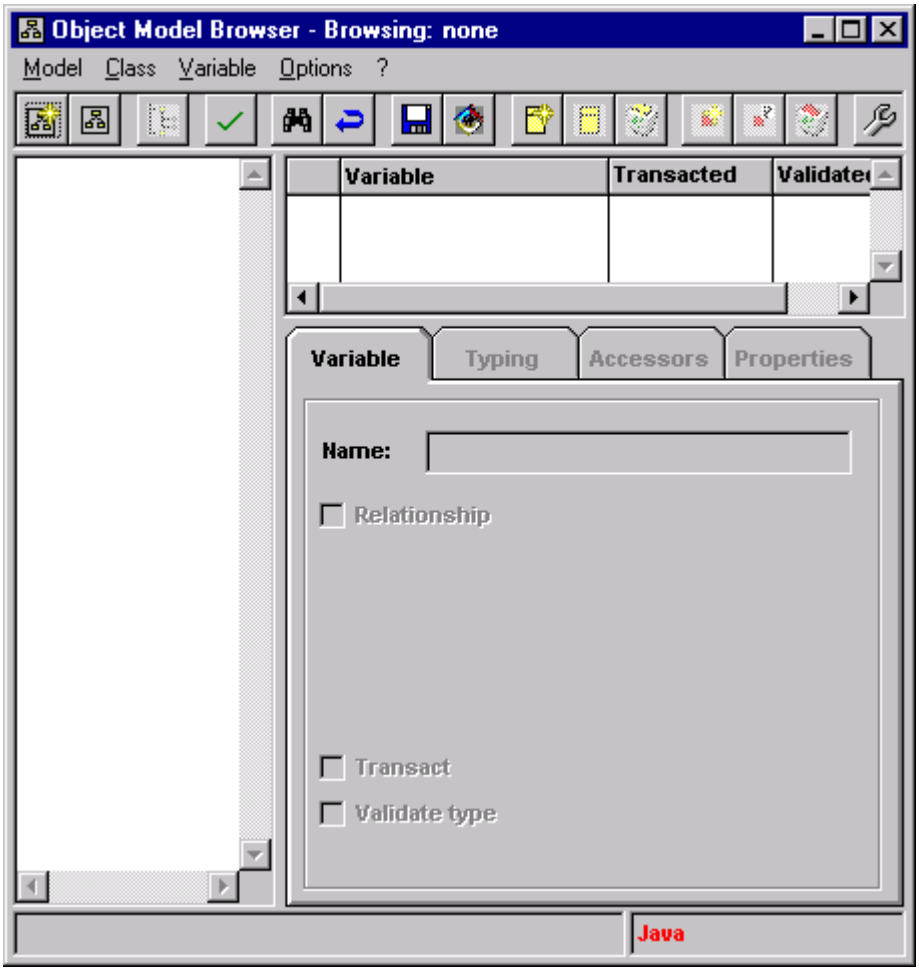


Abbildung 136. Object Model Browser

9.3. Model->POM Generator

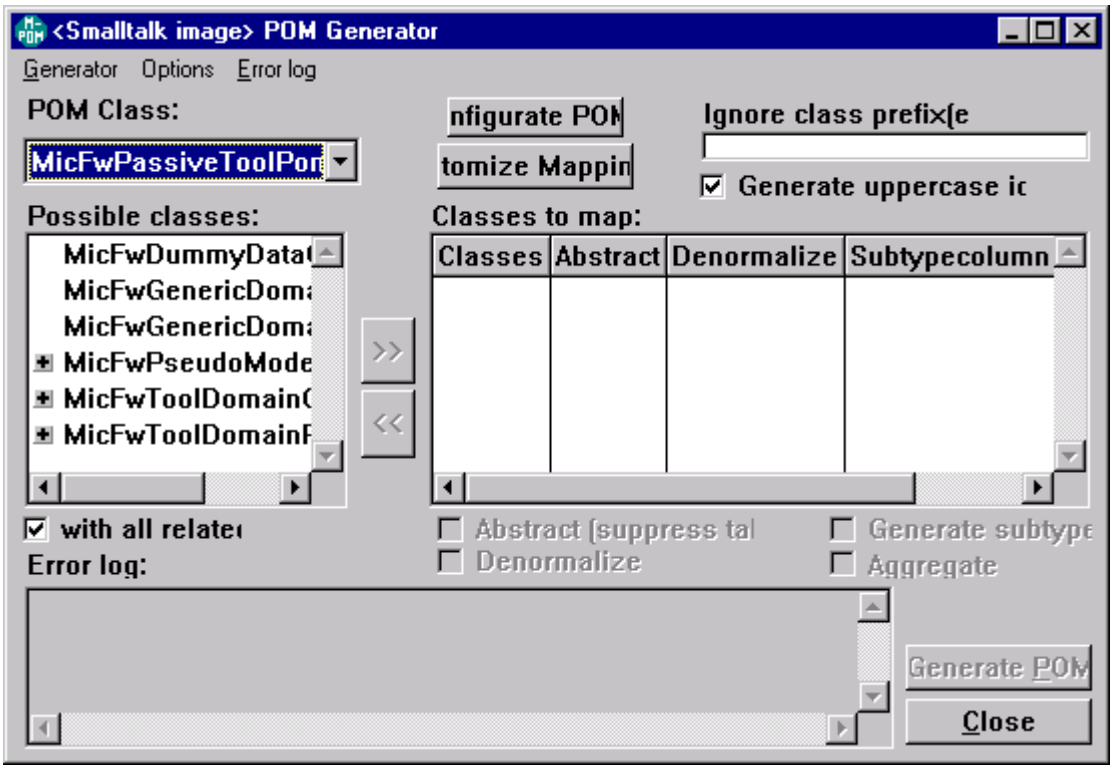


Abbildung 137. Model -> POM Generator

9.4. Model DB Mapping Tool

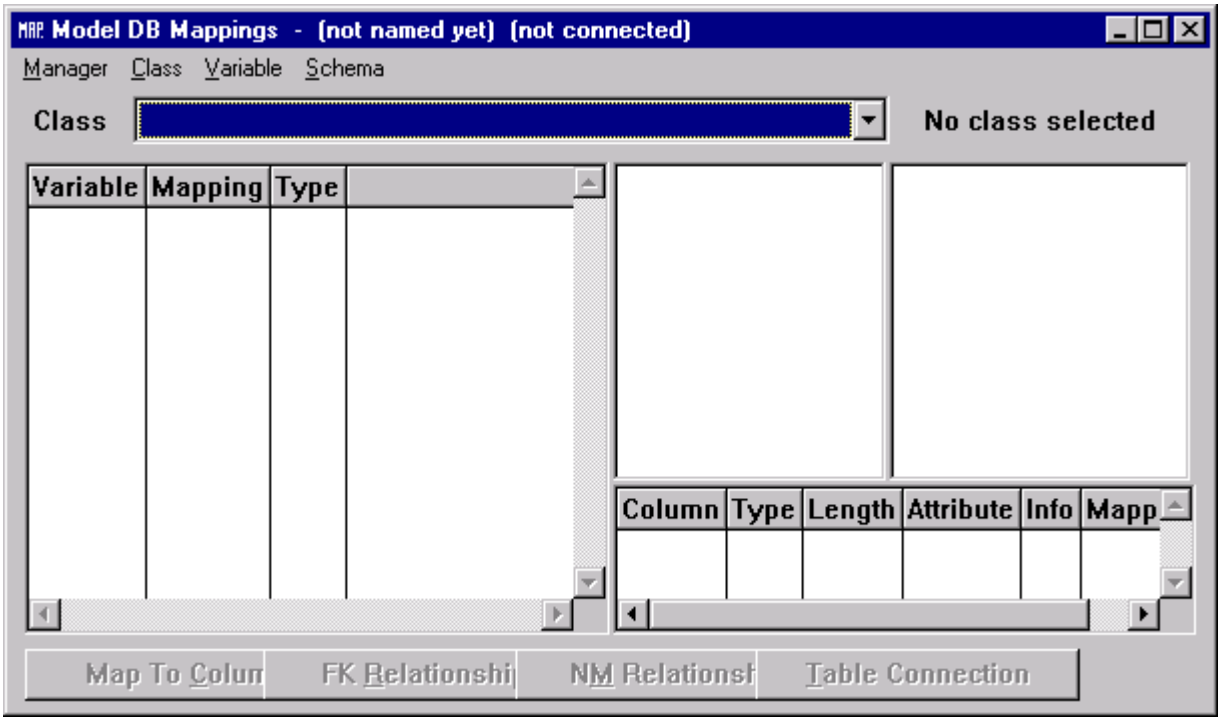


Abbildung 138. Model DB Mapping Tool

9.5. Interactive SQL

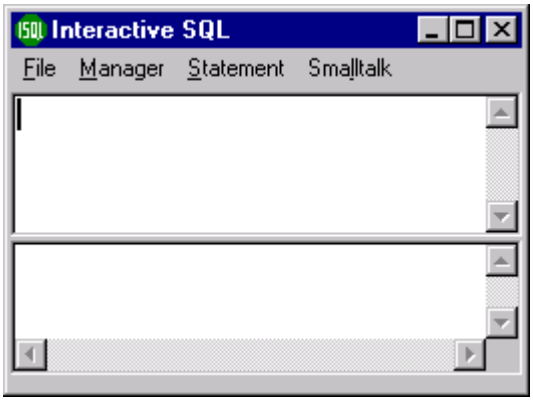


Abbildung 139. ISQL

9.6. Runtime POM Generator

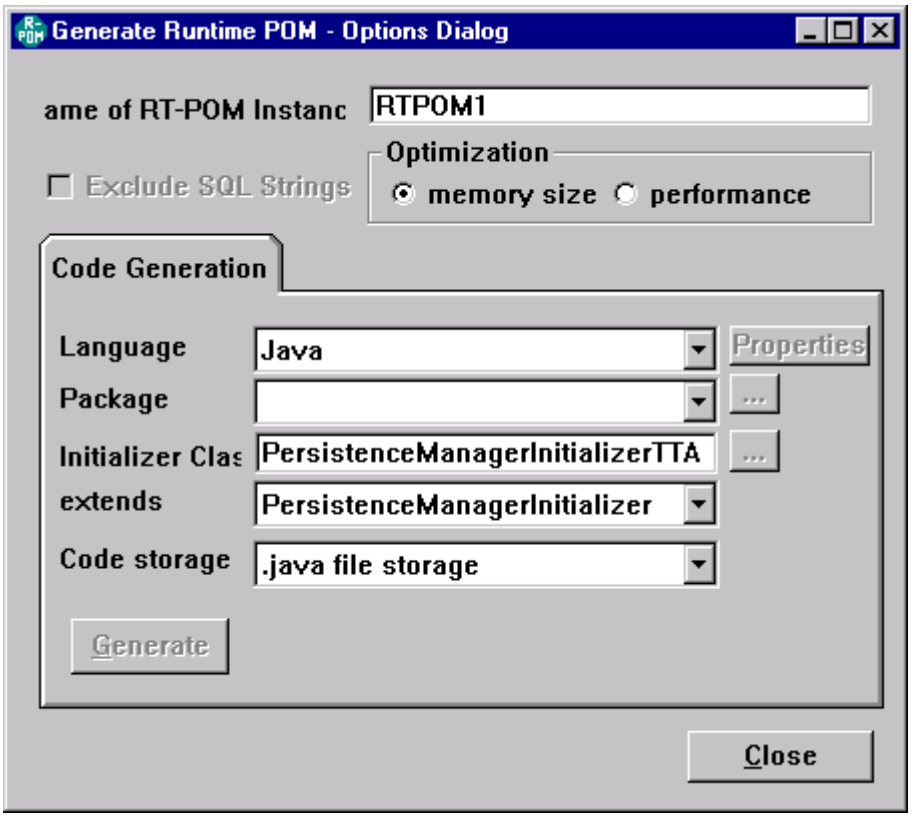


Abbildung 140. Runtime POM Generator

9.7. Logger

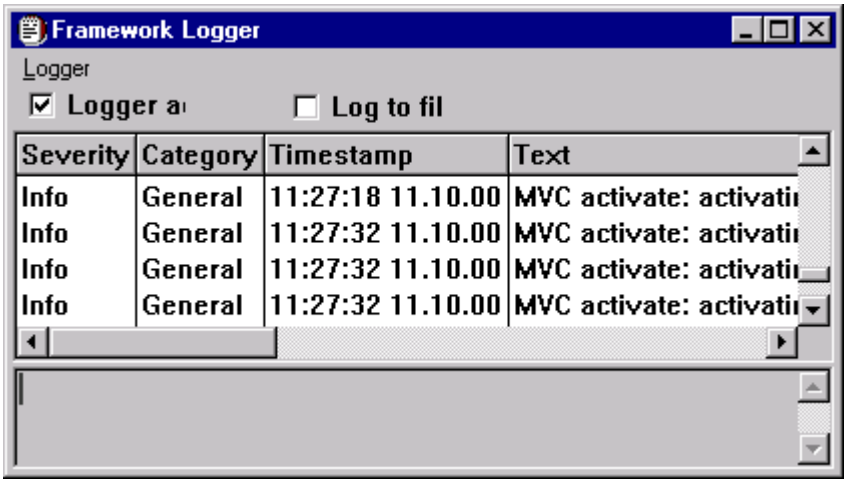


Abbildung 141. Logger

10. API

10.1. Activity

10.1.1. Overview

Auf den folgenden Seiten werden die einzelnen Methoden wie z.B. „public void addBusinessRule(String objkey, String attributeKey„ in Bezug auf ihre Aufgaben, Parameter und ihren Rückgabewerten besprochen.

NOTE: Die Programmierung im Activity-Frameworks geschieht innerhalb der Methoden der Klasse Activity :

```
// Generated by Together
import java.beans.*;

public class Activity {
  public void addBusinessRule(String objkey, String attributeKey,
              PropertyChangeListener pcl)
  public void addActionListener(String objkey, ActionListener acl);
  public void addVetoableBusinessRule(String objkey, String attributeKey,
              VetoableChangeListener vcl)
  public Iterator iterator()
  public Object get(String key)
  public boolean hasObject(String key)
  public boolean hasSlot(String key)
  public void put(String key, Object item)
  public boolean canComplete()
  public boolean canComplete(int nPartialMode)
  public void doStart()
  public boolean canStart()
  private PropertyChangeSupport listeners = new PropertyChangeSupport();
  private VetoableChangeSupport vlisteners = new VetoableChangeSupport();
}
```

10.1.2. public void addBusinessRule(String objkey, String attributeKey, PropertyChangeListener pcl)

10.1.2.1. Aufgabe

Durch Aufruf dieser Methode wird ein PropertyChangelistener in der Activity definiert, dessen Methode

```
public void propertyChange(PropertyChangeEvent evt)
```

mit folgendem PropertyChangeEvent gefeuert wird :

`Object getNewValue()`	Der neue Wert des Attributs
`Object getOldValue()`	Der alte Wert des Attributs
`Object getPropagationId()`	Nicht unterstützt
`String getPropertyName()`	Der Name des Attributs
`Object getSource()`	Das Geschäftsobjekt
`Void setPropagationId(Object propagationId)`	Nicht unterstützt

10.1.2.2. Parameter

`String objkey`	Der logische Name des Objekts
`String attributeKey`	Der logische Name des Attributs

`PropertyChangeListener pcl`	Der Listener , idealerweise eine Inner Class

10.1.2.3. Rückgabe

-.-

10.1.3. public void addActionListener(String objkey, ActionListener acl)

10.1.3.1. Aufgabe

Durch Aufruf dieser Methode wird ein ActionListener in der Activity definiert, dessen Methode

```
public void actionPerformed(ActionEvent evt)
```

mit folgendem ActionEvent gefeuert wird :

`Object getActionCommand()`	Der Name der Action

10.1.3.2. Parameter

`String objkey`	Der logische Name des Objekts
`ActionListener acl`	Der Listener , idealerweise eine Inner Class

10.1.3.3. Rückgabe

-.-

10.1.4. public void addVetoableBusinessRule(String objkey, String attributeKey, VetoableChangeListener vcl)

10.1.4.1. Aufgabe

Durch Aufruf dieser Methode wird ein vetoableChangelistener in der Activity definiert, dessen Methode

```
public void vetoableChange (PropertyChangeEvent evt) throws PropertyVetoException
```

mit folgendem PropertyChangeEvent gefeuert wird :

`Object getNewValue()`	Der neue Wert des Attributs
`Object getOldValue()`	Der alte Wert des Attributs
`Object getPropagationId()`	Nicht unterstützt
`String getPropertyName()`	Der Name des Attributs
`Object getSource()`	Das Geschäftsobjekt
`Void setPropagationId(Object propagationId)`	Nicht unterstützt

Durch werfen einer PropertyVetoException wird die Zuweisung unterbunden. Der den Exception mitgegebene Parameter wird über das GUI-Subsystem (wenn vorhanden) visualisiert.

10.1.4.2. Parameter

`String objkey`	Der logische Name des Objekts
`String attributeKey`	Der logische Name des Attributs, null für alle Attribute
`PropertyChangeListener pcl`	Der Listener , idealerweise eine Inner Class

10.1.4.3. Rückgabe

-.-

10.1.5. public Iterator iterator()

10.1.5.1. Aufgabe

Liefert einen Iterator zurück, mit dem über alle im Workspace der jeweiligen Activity vorhandenen Slots iteriert wird.

10.1.5.2. Parameter

-.-

10.1.5.3. Rückgabe

Elemente des Iterators sind Instanzen der Klasse Pair :

```
public class Pair
{
    public String key;
    public Object value;
}
```

Ist ein Slot vorhanden, aber noch ohne Wert, so wird in value <null> zurückgegeben.

10.1.6. public Object get(String key)

10.1.6.1. Aufgabe

Liefert das Objekt zurück, dass in den Slot unter dem Namen <key> eingestellt wurde.

Erst unmittelbar vor dem Aufruf von canStart und nicht bereits im Konstruktor steht der Activity der Workspace zur Verfügung.

10.1.6.2. Parameter

String key : der Name des Slots;

10.1.6.3. Rückgabe

Der Inhalt des Slots, null wenn slot leer oder nicht vorhanden.

10.1.7. public boolean hasObject(String key)

10.1.7.1. Aufgabe

Zum feststellen, ob der Slot mit dem Namen key mit einem Wert belegt ist.

Erst unmittelbar vor dem Aufruf von canStart und nicht bereits im Konstruktor steht der Activity der Workspace zur Verfügung.

10.1.7.2. Parameter

String key : der Name des Slots;

10.1.7.3. Rückgabe

false, wenn slot leer oder nicht vorhanden.

true, wenn slot mit Wert gefüllt.

10.1.8. public boolean hasSlot(String key)

10.1.8.1. Aufgabe

Zum feststellen, ob der Slot mit dem Namen key für die aktuelle Activity im Zugriff ist.

Erst unmittelbar vor dem Aufruf von canStart und nicht bereits im Konstruktor steht der Activity der Workspace zur Verfügung.

10.1.8.2. Parameter

String key : der Name des Slots;

10.1.8.3. Rückgabe

false, wenn slot nicht vorhanden.

true, wenn slot vorhanden.

10.1.9. public void put(String key, Object item)

10.1.9.1. Aufgabe

Stellt ein Objekt in den Workspace ein.

Erst unmittelbar vor dem Aufruf von canStart und nicht bereits im Konstruktor steht der Activity der Workspace zur Verfügung.

10.1.9.2. Parameter

String key : der Name des Slots;

Object item : das einzustellende Objekt

Wenn item null ist, so wird der entsprechende Slot geleert.

10.1.9.3. Rückgabe

-.-

10.1.10. public boolean canComplete()

10.1.10.1. Aufgabe

Callback, wird aufgerufen, wenn die Activity beendet werden soll, zur Rückfrage, ob dies möglich ist.

10.1.10.2. Parameter

-.-

10.1.10.3. Rückgabe

true : die Activity kann beendet werden, die fachlichen Anforderungen sind erfüllt.

false : die Activity kann nicht beendet werden, die fachlichen Anforderungen sind nicht erfüllt.

10.1.11. public boolean canComplete(int nPartialMode)

10.1.11.1. Aufgabe

Callback, wird aufgerufen, wenn die Activity temporär eingefroren werden soll werden soll, zur Rückfrage, ob dies möglich ist.

Dieser Aufruf zieht keinen doComplete nach sich, die internen Daten des lokalen Workspaces werden nicht in den Globalen überführt.

10.1.11.2. Parameter

int nPartialMode : ACTIVITY.SLEEPONLY

Ist immer gesetzt, reserved for future use.

10.1.11.3. Rückgabe

true : die Activity kann temporär eingefroren werden, die fachlichen Anforderungen sind erfüllt.

false : die Activity kann nicht temporär eingefroren werden, die fachlichen Anforderungen sind nicht erfüllt.

10.1.12. public void doStart()

10.1.12.1. Aufgabe

Wird aufgerufen, um die Geschäftsregeln zu initialisieren, also Aufrufe von addBusinessRule etc abzusetzen. Die eigentliche Verarbeitung erfolgt dann innerhalb der Listener.

Diese Listener sollten nicht bereits in canStart angebunden werden. Im Falle, dass eine GUI angebunden wird, ist sichergestellt, dass die Anbindung der GUI erst nach diesem Aufruf erfolgt.

10.1.12.2. Parameter

-.-

10.1.12.3. Rückgabe

-.-

10.1.13. public boolean canStart()

10.1.13.1. Aufgabe

Wird aufgerufen, um die preconditions abzubilden. Erst unmittelbar vor dem Aufruf von canStart und nicht bereits im Konstruktor steht der Activity der Workspace zur Verfügung.

10.1.13.2. Parameter

-.-

10.1.13.3. Rückgabe

true : die Vorbedingungen sind erfüllt.

false : die Aktivität wünscht, nicht gestartet zu werden. Dies muss nicht unbedingt eine gescheiterte Vorbedingung sein, es kann auch bedeuten, dass keine Notwendigkeit besteht, die Aktivität zu starten, da sie ihre Aufgabe bereits erledigt hat.

10.1.14. public void done()

10.1.14.1. Aufgabe

Wird aufgerufen, um von der Aktivität aus den Wunsch zu signalisieren, dass die Aktivität beendet zu werden wünscht. Initiiert die Abschluß-Sequenz von canComplete und doComplete.

10.1.14.2. Parameter

-.-

10.1.14.3. Rückgabe

-.-

10.1.15. public void failed()

10.1.15.1. Aufgabe

Wird aufgerufen, um von der Aktivität aus den Wunsch zu signalisieren, dass die Aktivität abgebrochen zu werden wünscht. Initiiert nicht die Abschluß-Sequenz von canComplete und doComplete. Die Aktivität wird sofort abgebrochen. Als Folgeaktivitäten werden die in der Konfigurationsdatei unter dem Tag abend activity = Activity name abgearbeitet.

10.1.15.2. Parameter

-.-

10.1.15.3. Rückgabe

-.-

10.1.16. public void failed(String msg)

10.1.16.1. Aufgabe

Wird aufgerufen, um von der Aktivität aus den Wunsch zu signalisieren, dass die Aktivität abgebrochen zu werden wünscht. Initiiert nicht die Abschluß-Sequenz von canComplete und

doComplete. Die Aktivität wird sofort abgebrochen. Als Folgeaktivitäten werden die in der Konfigurationsdatei unter dem Tag abend activity = Activity name abgearbeitet.

10.1.16.2. Parameter

String msg : wird über das GUI-Subsystem (wenn vorhanden) visualisiert.

10.1.16.3. Rückgabe

-.-

10.1.17. public void failed(Exception exc)

10.1.17.1. Aufgabe

Wird aufgerufen, um von der Aktivität aus den Wunsch zu signalisieren, dass die Aktivität abgebrochen zu werden wünscht. Initiiert nicht die Abschluß-Sequenz von canComplete und doComplete. Die Aktivität wird sofort abgebrochen. Als Folgeaktivitäten werden die in der Konfigurationsdatei unter dem Tag abend activity = Activity name abgearbeitet.

10.1.17.2. Parameter

Exception exc : wird über das GUI-Subsystem (wenn vorhanden) visualisiert.

10.1.17.3. Rückgabe

-.-

10.2. Manager

10.2.1. Interface ActivityGuiEventSupport

10.2.2. Interface ActivityNotificationSupport

10.2.3. ActivityWorkflowSupport

10.2.4. Interface ActivityTransactionalSupport

10.3. Configuration

10.3.1. Interface Configuration

10.4. Creator

10.4.1. Interface Creator

10.5. Workspace

10.5.1. Interface Workspace

10.6. Context

10.6.1. Interface Context

10.7. Technische Beschreibung der Implementation des ActFW (API)

10.7.1. Manager

10.7.1.1. ActivityGuiEventSupport

10.7.1.2. ActivityNotificationSupport

10.7.1.3. ActivityWorkflowSupport

10.7.1.4. ActivityTransactionalSupport

10.7.2. Configuration

10.7.3. Creator

10.7.4. Activity

10.7.5. Workspace

10.7.6. Context

10.8. API reference (ask, alert, tell, prompt, select) ??

11. Trouble shooting

12. FAQ

13. Glossary

14. Index

000926TTA: note: a bug in the index. i will insert later.

15. Abbildungverzeichnis

Abbildung 1. VA classes 11
Abbildung 2. DSE classes 12
Abbildung 3. Options / Resources 12
Abbildung 4. Model Integrator dialog (117) 13
Abbildung 5. Model Integrator Preferences dialog (185) 13
Abbildung 6. Browser command settings in the Model Integrator Preferences dialog (186) 13
Abbildung 7. Use cases (911) 17
Abbildung 8. Activities (913) 17
Abbildung 9. Activity Suchen diagram (912) 18
Abbildung 10. Activity KundeBearbeiten diagram (914) 19
Abbildung 11. Activity LkwBearbeiten diagram (915) 20
Abbildung 12. Domain objects (907) 21
Abbildung 13. Object accessors created with OMB (906) 21
Abbildung 14. Database tables for domain objects (181, 182) 21
Abbildung 15. RT-POM interface between objects and database tables (908) 22
Abbildung 16. Suchen dialog (919) 22
Abbildung 17. KundeBearbeiten dialog (920) 23
Abbildung 18. LkwBearbeiten dialog (921) 23
Abbildung 19. MVC components (909) 24
Abbildung 20. MVC actor input interactions (910) 25
Abbildung 21. Model Integrator dialog (117) 30
Abbildung 22. ToolCenter and OMB dialogs 31
Abbildung 23. OMB options / Accessor prefixes (118) 32
Abbildung 24. OMB default package name (119) 32
Abbildung 25. Class specification window 33
Abbildung 26. Specification of class Kunde 33
Abbildung 27. Package specification (120) 33
Abbildung 28. Kunde class in OMB 33
Abbildung 29. Lkw class in OMB 34
Abbildung 30. Enter name of new instance variables (121) 34
Abbildung 31. Variable id for Kunde in OMB 34
Abbildung 32. Transact checkbox 34
Abbildung 33. Typing for variable id 35
Abbildung 34. Accessors for variable id (124) 35
Abbildung 35. Specify id as key and persistent (122) 35
Abbildung 36. Complete Kunde variables (123) 36
Abbildung 37. Complete Lkw variables (125) 36
Abbildung 38. Specification of relationship type for variable 37
Abbildung 39. Source class maximum/minium settings 38
Abbildung 40. Settings for relationship from myKunde (126) 38
Abbildung 41. Imported model classes Kunde, Lkw in VA (127, 128) 39
Abbildung 42. POM Generator (129) 40
Abbildung 43. Select POM class 40
Abbildung 44. Configuration POM dialog 41
Abbildung 45. Database description (130) 41
Abbildung 46. Statement executor (131) 42
Abbildung 47. Persistent Property Policy 42
Abbildung 48. Select classes in POM Generator (132) 43
Abbildung 49. Model DB Mappings 43
Abbildung 50. Mappings class Kunde (133) 44
Abbildung 51. Mappings table KUNDE (134) 44
Abbildung 52. Mappings table LKW (135) 44

Abbildung 53. Mappings class Lkw (136) 45
Abbildung 54. Mappings table LKW (137) 45
Abbildung 55. Mappings table KUNDE (138) 45
Abbildung 56. Enter RTPOM info (139) 46
Abbildung 57. RTPOM generation result (140) 47
Abbildung 58. Source file import (141) 47
Abbildung 59. SmartGuide for file import (142) 48
Abbildung 60. Oracle8i lite Navigator main dialog (143) 49
Abbildung 61. Enter database name (144) 49
Abbildung 62. Database in Oracle list (145) 49
Abbildung 63. Database properties (146) 50
Abbildung 64. ODBC Datenquellen Administrator 50
Abbildung 65. Oracle ODBC setup (149) 50
Abbildung 66. Choose tables for DDL export 51
Abbildung 67. Schema export dialog (ERROR) 51
Abbildung 68. Exported DDL text 52
Abbildung 69. ISQL dialog 52
Abbildung 70. ISQL dialog with copied SQL text 52
Abbildung 71. Choose Persistence Manager dialog (147) 53
Abbildung 72. ISQL dialog header (not connected) (148) 53
Abbildung 73. Datenquelle auswählen (150) 53
Abbildung 74. SQL in ISQL (151) 53
Abbildung 75. Ready and commit statements (ISQL) 53
Abbildung 76. New tables in database ZZZDB (152) 54
Abbildung 77.Empty table KUNDE (153) 54
Abbildung 78. Empty table LKW (154) 54
Abbildung 79. Data for tables (155, 156) 54
Abbildung 80. Panel de.dekra.tutorial.view.ZZZPanelSuchen (162) 61
Abbildung 81. SmartGuide for de.dekra.tutorial.view.ZZZPanelSuchen (164) 62
Abbildung 82. Attributes for de.dekra.tutorial.view.ZZZPanelSuchen (165) 62
Abbildung 83. JLable's for de.dekra.tutorial.view.ZZZPanelSuchen (157) 62
Abbildung 84. ControlConnectDescriptor dialog (158) 63
Abbildung 85. controlConnectDescriptor setting (159) 63
Abbildung 86. ZZZPanelSuchen (160) 64
Abbildung 87. controlConnectDescriptor setting (161) 64
Abbildung 88. Complete de.dekra.tutorial.view.ZZZPanelSuchen (162) 64
Abbildung 89. controlConnectDescriptors for ZZZPanelSuchen (919) 65
Abbildung 90. Panel de.dekra.tutorial.view.ZZZPanelKundeBearbeiten (920) 65
Abbildung 91. Panel de.dekra.tutorial.view.ZZZPanelLkwBearbeiten (921) 66
Abbildung 92. ZZZFrame content pane layout setting (168) 66
Abbildung 93. Choosing Bean DseJPanel (170) 67
Abbildung 94. DseJPanel layout setting (171) 67
Abbildung 95. ZZZPanelSuchen settings in ZZZFrame (172) 67
Abbildung 96. ZZZPanelKundeBearbeiten settings in ZZZFrame (173) 68
Abbildung 97. ZZZPanelLkwBearbeiten settings in ZZZFrame (174) 68
Abbildung 98. Projects for class path 72
Abbildung 99. ZZZMain dialog (175) 73
Abbildung 100. Search for Kunde with id=1 results (176) 73
Abbildung 101. ZZZPanelKundeBearbeiten (result of pressing Edit button) (177) 74
Abbildung 102. Saved changes (178) 74
Abbildung 103. Saved changes to Kunde in the database (179) 74
Abbildung 104. Direct to ZZZPanelLkwBearbeiten (180) 74
Abbildung 105. TestSuiteBrowser main dialog 77
Abbildung 106. Available test suites 77
Abbildung 107. Selecting activity.configuration.xml file for test 77
Abbildung 108.Test passed 78

Abbildung 109. Selecting multiple test cases 78
Abbildung 110. Results of multiple tests 78
Abbildung 111. New test in TestSuite 79
Abbildung 112. ZZZObjectModel1.ome in OMB(187) 80
Abbildung 113. Kunde with new variable name2 (188) 80
Abbildung 114. New Kunde and Lkw versions in VA (189) 80
Abbildung 115. RTPOM Generator settings for ZZZRTPOM2 (190) 81
Abbildung 116. KUNDE database with new table NAME2 (191) 82
Abbildung 117. ZZZPanelSuchen with label, textfield for name2 (192) 83
Abbildung 118. ZZZPanelKundeBearbeiten with label, textfield for name2 (193) 83
Abbildung 119. ZZZMain main dialog with new textfield (194) 83
Abbildung 120. Text for name2 succesfully entered (195) 84
Abbildung 121. ControlConnectDescriptor for fullname (196) 85
Abbildung 122. fullname in KundeBearbeiten view (197) 87
Abbildung 123: Manager Subsystem (managersubsystem.gif) 94
Abbildung 124: Manager All (managerall.gif) 95
Abbildung 125: Triade (triade.gif) 101
Abbildung 126: Wer Wo Was (werwowas.gif) 102
Abbildung 127: Aussen - Innen (ausseninnen.gif) 104
Abbildung 128. Activity Diagramm als XML 110
Abbildung 129. UseCase als Activity 110
Abbildung 130. Lokale und globale Workspaces 112
Abbildung 131. Lebensdauer eines lokalen Workspaces 113
Abbildung 132: Übergang (übergang.gif) 115
Abbildung 133: Activity done (activity.done.gif) 116
Abbildung 134: Globaler Workspace (globalerworkspace.gif) 123
Abbildung 135. Tool Center 130
Abbildung 136. Object Model Browser 131
Abbildung 137. Model -> POM Generator 132
Abbildung 138. Model DB Mapping Tool 133
Abbildung 139. ISQL 134
Abbildung 140. Runtime POM Generator 135
Abbildung 141. Logger 136