Appendix A

Integration with Form-Oriented Analysis

In this appendix an example transformation of a system feature that is specified as a form-oriented analysis model into NSP technology is given. All model elements of form-oriented analysis that are used in this appendix are defined in the dissertation of Gerald Weber [178]. The system feature is modeled by a form chart with respect to a layered data model , i.e. a data dictionary and a semantic data model. A form chart can be transformed into server pages technology in several different ways, i.e. yielding several different system designs. There is one canonical transformation. This transformation maps each server action and each client page to one server page. The message types of server actions and client pages, which are specified in the data dictionary, become the web signatures. The server page of a server action hosts business logic implementation only and the server page of a client page has the only purpose to present data and create new data input capabilities. A server page of a server action prepares the data that has to be presented in the next step and branches the control flow with respect to the enabling conditions of the outgoing transitions. Forward engineering tools can be build that realize the canonical transformation by generating a complete executable prototype system for a given form chart and layered data model, whereas the prototype system provides protected regions in all kinds of generated server pages as well-defined hooks. The tool Gently [56][67] is such a forward engineering tool for generating Java Server Pages based systems. Again the Gently approach is presented in the dissertation of Gerald Weber [178].

The canonical transformation guarantees maximum reusability and low coupling and allows for an immediate exploitation of further form-oriented dialogue elements like e.g. dialogue constraints. However it yields not the most succinct kind of design. Design-decisions may lead to other justifiable transformations, for example to such transformations that foster high cohesion. Actually for example in this appendix a very straightforward transformation to NSP tech-

nology has been chosen, that always implements a server action and an adjacent client page as one server page.

The NSP concept of functional decomposition of server pages can be exploited to implement a server action's branching to client pages. The NSP concept of higher order server pages can be exploited to solve some typical design problems that often arise in web application architecture but can be identified early in form charts as has been explained in section 4.3 with listing 4.2 and Figure 4.4. However the aim of this chapter is just to give some confidence that the NSP technology is tightly integrated with the overall form-oriented approach by providing a somewhat larger example from a real system that is currently under development. The chosen transformation is already sufficient for this purpose. It can be seen how web signatures of a system dialogue become NSP web signatures.

The appendix proceeds as follows. First the vision of a software system is given. Then a single desired feature in this system is described informally and then specified using form-oriented analysis. In section A.2 the mapping of this specification to NSP server pages is given.

A.1 Problem Description

A.1.1 Vision

It is the vision to develop a combined CSCW/project management tool for the software engineering process EASE.

EASE (Education for Actual Software Engineering) [65] is a software engineering process model for higher education. It is the first such process that is designed from scratch, whereas it is oriented towards proven concepts from mature andragogical methodologies like Collaborative Learning [21], Action Learning [154], and Entraînement Mental [30].

The project management tool PEASE (Platform for EASE) is currently under development [93]. The project planning phase, comprising competitive product analysis [177] as a basis for a positive stop-or-go decision, has already been finished [92]. The PEASE platform is a web-enabled project management tool that is oriented towards groupware [82]. There are already a couple of such systems, the most prominent one is probably [136] which arose in the open source community [152]. A detailed explanation of EASE is given in [65]. PEASE supports all activities found in the EASE process architecture, especially the micro process of EASE, which can be loosely compared to the iteration planning of the Extreme Programming [11] software development approach. The EASE micro process is a fast iterative alternation of meetings and plannings. In every meeting tasks are found, sorted, discussed, and assigned to teams. Every week new teams are formed; small changing groups foster the communication between students. Thereby the project's progress is tracked. In this appendix a single feature of the future PEASE platform for managing tasks serves as example. The task manager feature allows removing tasks and deleting or adding team

members. It is most probably used after the assignment of tasks , in order to adjust some recently entered data.

A.1.2 Task Manager Feature Description

The task manager feature allows removing tasks, deleting team members from tasks, and adding new team members to tasks. At the entry page of the task manager feature the user is shown a single select menu of all current tasks. She can select a task and choose between two options, namely removing the task and editing the task[1]. For this purpose two submit buttons are offered. Pressing a submit button will be successful only if the user has selected at least one task. If the user has chosen to remove a task from the task list first a message page is presented to the user. The user is asked to acknowledge or otherwise to redeem that she actually wants to remove the chosen task. Based on the user's decision the task is removed and the dialogue returns to the task manager entry page.

If the user chooses to edit a task from the entry page's task list a page is shown that contains a link and a form. With the link it is possible to return to the entry page without changing anything. The form contains two multiple select menus. In the first menu the team members of the selected task are listed. Selected team members will be removed from the task on submit. In the second menu all project members that are not assigned to the task are listed. Selected project members will be added to the task as new team members on submit. On submit the necessary update operations are executed. Then a message page is presented. It contains the name of the current task and the updated list of team members and it invites the user to control the result. The user has the option to return to the entry page or to edit the task again.

A.1.3 Task Manager Feature Specification

The task manager feature that has been described in section A.1.2 is modeled by the form chart given in Figure A.1 and a layered data model given in Figure A.2. The persistent data is given by the semantic data model: every task has a name and a list of team members. The semantics of the data dictionary types, i.e. the message types, and their interplay with semantic data types is explained in [178]: every type introduced in the semantic data model is available in the data dictionary as opaque reference type, which consists of the keys to the persistent data objects.

We do not give a complete dialogue constraint specification. We pick the dialogue constraints for one page/server transition as an example. A page/server transition is annotated with an enabling condition, a client output constraint, and a server input constraint.

[1]The task manager feature is a feature in the sense of feature orientation [60]. That is the specification of the pages that make up the respective dialogue are partial specifications. In a complete system specification the single pages may offer more information and more interaction capabilities. For example the page just described will have a link for the registration of a new task in the complete system specification.

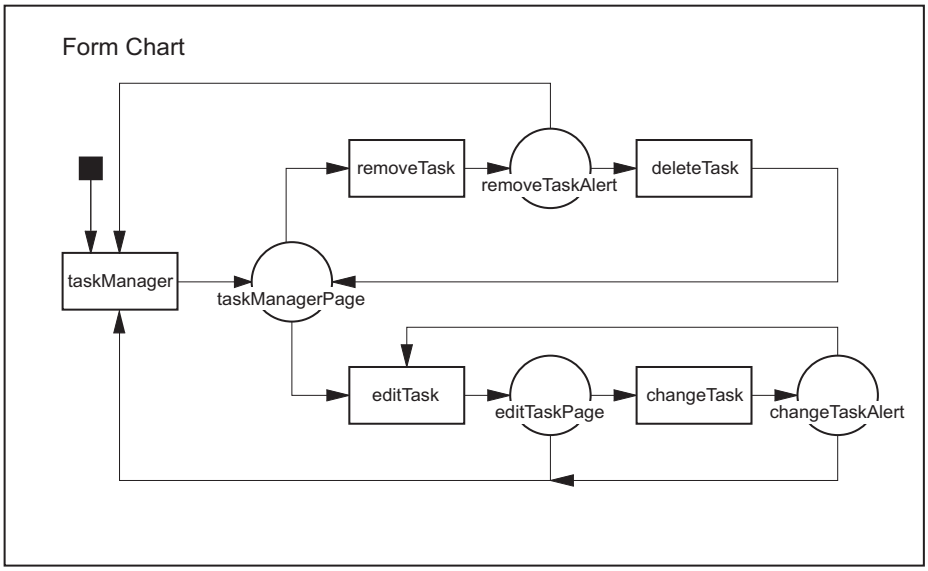

Figure A.1: Form Chart Diagram. The figure visualizes a task manager feature of a combined CSCW/project management tool.

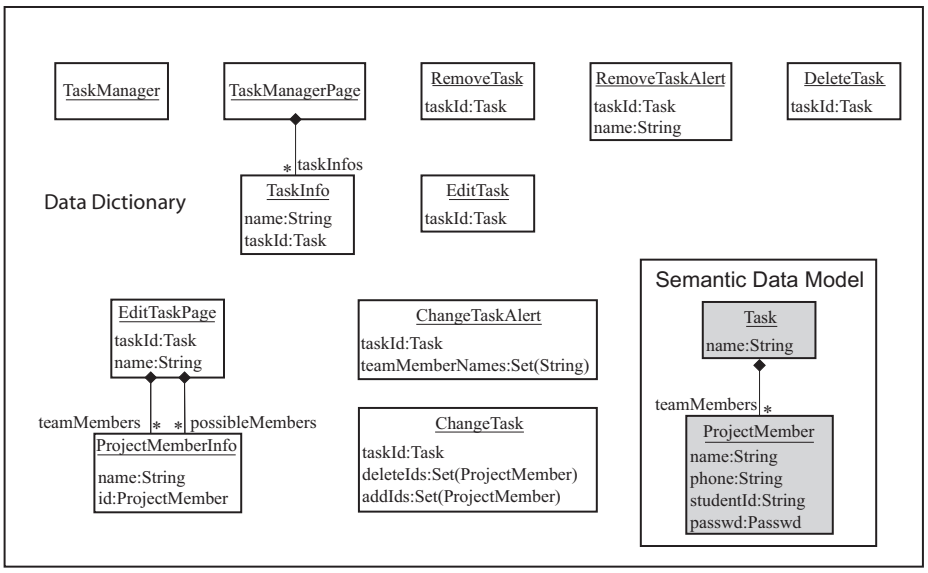

Figure A.2: Layered Data Model. The figure visualizes the form-oriented data model which underlies a task manager feature of a combined CSCW/project management tool. The data model consists of two layers, a data dictionary and a semantic data model. Data model types are used as opaque reference types in the data dictionary.

Listing A.1

```
editTaskPage to changeTask{
  enabling:
  clientOutput:
    changeTask.taskId=editTaskPage.taskId
    editTaskPage.teamMembers->includesAll(changeTask.deleteIds)
    editTaskPage.possibleMembers->includesAll(changeTask.addIds)
  serverInput:
    ChangeTask.deleteIds->notEmpty or ChangeTask.addIds->notEmpty
}
```

We consider the page editTaskPage in Figure A.1, which offers the user a form with two select menus for removing and adding team members. We consider the transition that is triggered by submitting the form, i.e. the transition to the changeTask server action. In listing A.1 the constraints for this transition are given. The enabling condition is empty. We consider the constraints of the client output specification next. First the current taskId must be passed unchanged to the server action. Then the team members that have been chosen to be deleted must actually belong to the list of team members presented to the user. The team members that should be added to the team are constrained analogously. The server input constraint states that the server action should only be executed if the user has selected at least one change.

A.2 Mapping A Form-Oriented Specification to NSP

The form-oriented feature specification given in section A.1.3 is implemented by the NSP server pages given in listings A.2 to A.5. Under the chosen transformation always a server action and the adjacent client page is implemented by one server page. The server page DeleteTask implements the effect of data deletion, but it does neither implement the retrieval of information for the client page taskManagerPage nor its presentation. For this purpose it reuses the implementation provided by the server page TaskManager by forwarding to it. The transformation of server actions and client pages is summarized in the informal equation A.1.

$$\begin{array}{rcl}
\texttt{taskManager taskManagerPage} & \mapsto & \texttt{TaskManager} \\
\texttt{removeTask removeTaskAlert} & \mapsto & \texttt{RemoveTask} \\
\texttt{deleteTaskManager taskManagerPage} & \mapsto & \texttt{DeleteTask} \rightsquigarrow \texttt{TaskManager} \\
\texttt{editTask editTaskPage} & \mapsto & \texttt{EditTask} \\
\texttt{changeTask changeTaskAlert} & \mapsto & \texttt{ChangeTask}
\end{array} \tag{A.1}$$

The message types of server actions are directly mapped to web signatures of NSP server pages.

Some message type parameters of client pages are mapped to formal parameters of a server page. Others are mapped to local variables of a server page or Java expressions. The mapping is pictured by the informal equation A.2[2].

```
TaskManagerPage.taskInfos.taskId ↦ TaskManager.taskIds
TaskManagerPage.taskInfos[i].name ↦ .getTaskName(TaskManager.taskIds[i])
RemoveTaskAlert.taskId ↦ RemoveTask.<param>taskId
RemoveTaskAlert.name ↦ .getTaskName(RemoveTask.<param>taskId)
EditTaskPage.taskId ↦ EditTask.<param>taskId
EditTaskPage.name ↦ .getTaskName(EditTask.<param>taskId)
EditTaskPage.teamMembers.id ↦ EditTask.teamMemberIds
  EditTaskPage.teamMembers.name[i]
  ↦ getProjectMemberName(EditTask.teamMemberIds[i])
EditTaskPage.possibleMembers.id ↦ EditTask.possibleMemberIds
  EditTaskPage.possibleMembers.name[i]
  ↦ getProjectMemberName(EditTask.possibleMemberIds[i])
ChangeTaskAlert.taskId ↦ ChangeTask.<param>taskId
ChangeTaskAlert.teamMemberNames[i] ↦ ChangeTask.teamMemberNames
```

(A.2)

Note how the NSP submit button concept is used for realizing several possible page/server transitions of a client page directly. For example the submit buttons in listing A.2 target different server pages.

The output constraints for project members of listing A.1 are easily fulfilled, because only select menus has been chosen to implement the lists of team members and possibly project members: the user cannot choose values that lead to a violation of the constraint. The server input constraint of listing A.1 is ignored in the current implementation. It has status TBD [95] per definition, i.e. only in a complete specification will be decided whether the server input constraint is ensured by client side scripting or additional dialogue.

[2] Formal server page parameters are marked by a juxtaposed <param>-tag in equation A.2. The methods in equation A.2 are class methods of the imported class CustomerBase. The class name CustomerBase is dropped. The semantics of the methods is implicitly specified by equation A.2, too.

Listing A.2

```
<nsp name="TaskManager">
  <html>
    <head><title>Task Manager</title></head><java>
    import myBusinessModel.CustomerBase;</java>
    <body><java>
      TaskId taskIds[]=CustomerBase.getTaskIds();</java>
      <form callee="editTask">
        <select param="taskId"><java>
          for (i=0;i<taskIds.length;i++){</java>
            <option>
              <value>
                taskIds[i]
              </value>
              <label>
                CustomerBase.getTaskName(taskIds[i])
              </label>
            </option><java>
          }</java>
        </select><br>
        <submit callee="removeTask"><label>Remove</label></submit><br>
        <submit callee="editTask"><label>Edit</label></submit><br>
      </form>
    </body>
  </html>
</nsp>
```

Listing A.3

```
<nsp name="RemoveTask">
  <param type="TaskId" name="taskId"></param>
  <html><head><title>Remove Task Alert</title></head>
  <body>
    Do you really want to remove task  
    <javaexpr>CustomerBase.getTaskName(taskId)</javaexpr>
      ? <br>
    <form callee="deleteTask">
      <submit callee="deleteTask">
        <label>YES</label>
        <hidden>taskId</hidden>
      </submit>
      <submit callee="taskManager">
        <label>NO</label>
      </submit>
    </form>
  </body>
</nsp>
```

Listing A.4

```
<nsp name="EditTask">
  <param type="TaskId" name="taskId"></param>
  <html><head><title>Edit Task</title></head>
  <body>
    You want to edit task  
    <javaexpr>CustomerBase.getTaskName(taskId)<javaexpr>
      .<br>
    <link callee="taskManager">Back to task manager.</link><br>
    <form callee="changeTask"><java>
      ProjectMemberId[] teamMemberIds = CustomerBase.getTeamMemberIds(taskId);
      ProjectMemberId[] possibleMemberIds =
        CustomerBase.getProjectMemberIdsUnequalTo(teamMemberIds);
      if (teamMemberIds.length>0) {</java>
        Remove the following team members from the task:<java>
        <select multiple param="deleteIds"><java>
          for (i=0;i<teamMemberIds.length;i++) {</java>
          <option>
            <value>
              teamMemberIds[i]
            </value>
            <label>
              <javaexpr>
              CustomerBase.getProjectMemberName(teamMemberIds[i])
              </javaexpr>
            </label>
          </option>
        </select>
      }</java>
      if (possibleMemberIds.length>0) {</java>
       Add the following team members to the task:<java>
        <select multiple param="addIds"><java>
          for (i=0;i<possibleMemberIds.length;i++) {</java>
          <option>
            <value>
              possibleMemberIds[i]
            </value>
            <label>
              <javaexpr>
              CustomerBase.getProjectMemberName(possibleMemberIds[i])
              </javaexpr>
            </label>
          </option>
        </select>
      }</java>
      <hidden param="taskId">taskId</hidden>
      <submit></submit>
    </form>
  </body>
</nsp>
```

Listing A.5

```
<nsp name="ChangeTask">
  <param type="TaskId" name="taskId"></param>
  <param type="ProjectMemberId[]" name="deleteIds"></param>
  <param type="ProjectMemberId[]" name="addIds"></param>
  <html><head><title></title></head>
  <body><java>
    CustomerBase.deleteTeamMembers(deleteIds);
    CustomerBase.addTeamMembers(addIds);
    </java>
    You changed the task  
    <javaexpr>CustomerBase.getTaskName(taskId)</javaexpr>
     .<br>
    Please check if the changes are correct !<br>
    The task are assigned the following team members:<br><java>
    String[] teamMemberNames = CustomerBase.getTeamMemberNames(taskId);
    for (i=1;i<teamMemberIds.length;i++){</java>
      <javaexpr>i</javaexpr>.
      <javaexpr>teamMemberNames[i]</javaexpr>
      <br><java>
    }</java>
    <form callee="taskManager">
      <hidden param="taskId">taskId</hidden>
      <submit callee="taskManager"><label>OK</label></submit>
      <submit callee="editTask"><label>Edit Task Again</label></submit>
    </form>
  </body>
</nsp>
```

Listing A.6

```
<nsp name="DeleteTask">
  <param type="TaskId" name="taskId"></param>
  <html><head><title></title></head>
  <body><java>
    CustomerBase.deleteTask(taskId);</java>
    <forward callee="taskManager"></forward>
  </body>
</nsp>
```