Chapter 2

A Motivating Case Study

In this chapter we demonstrate Form-Oriented Analysis for an example application. We will see, how Form-Oriented Analysis yields a simple and intuitive overview as well as a formal description of the intended system. Our example system is a form-based seminar registration system as it is suited for a single course. The screen shots of the system are shown in Figure 2.1. The front page shows the students registered so far and contains links to the different interaction options. New students can register themselves. From the homepage, students already registered can change or delete their registration. Each link leads to a new page specific for the chosen option.

This example system is already suited to explain the main properties of a submit/response style system and to compare it with other system types. The first important property of submit/response style systems is that they follow a strict two-staged interaction paradigm. Submit/response style systems present to the user at each point in time a page which offers two kinds of interaction options: page interaction and page change. The registration page for our example offers a back link as well as input fields for text input, which comprise a form together with the submit button. The back link and the submit button on the one hand and the input fields on the other hand differ in the effect of input. Input into the input fields is only preliminary until the submit button is pressed, it cannot change the system state permanently. Links and a submit button change the page with an atomic interaction. They are the only interactions that change the system state permanently. Hence the difference between both kinds of interaction options is that page interaction is fine grained, volatile and reversible, while page change affects the system state and consists of singular atomic actions. Supposed the user has chosen to insert a new address record and has already filled out all fields of the new record, but she has not submitted the new address. Then in our system there would be no new address record in the non-transient system state up to this time. Only if she submits the record to the system, the new address record would be entered in the database. But if she presses the back button instead, the input is lost. Page interactions are well understood and can be offered by an application independent browser, which

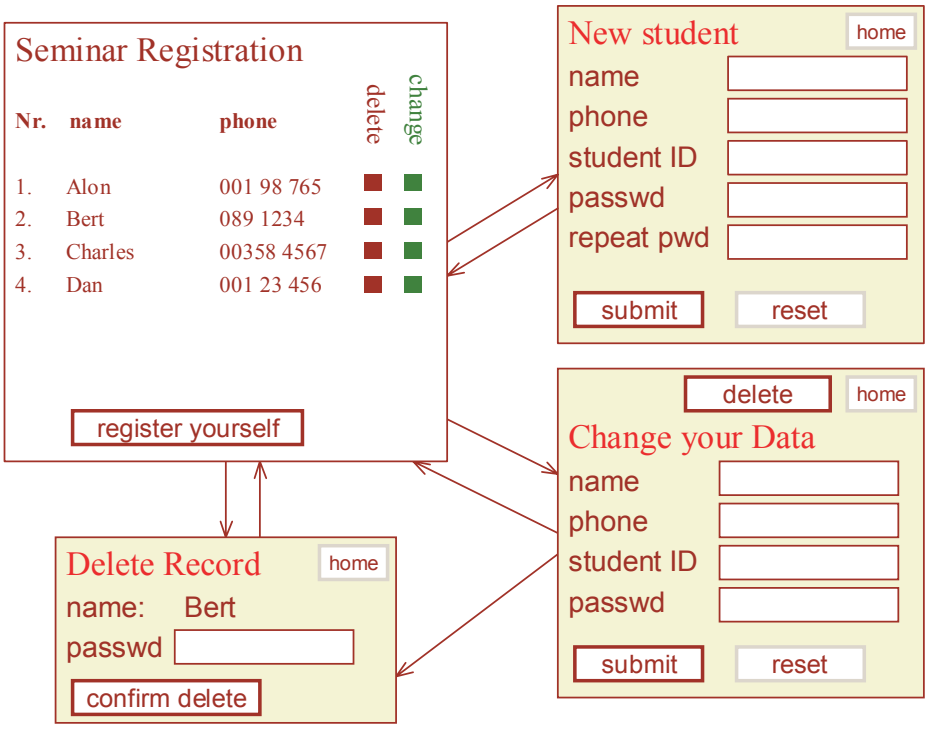

Figure 2.1: Overview of the screens of an online seminar registration system.

is able to interpret a declarative page description and provide a sufficiently powerful standard page interaction paradigm. This explains the success story of concrete technologies like the HTML form concept together with the web browser as a standard ultra-thin client. This two-staged interaction paradigm is an important difference of a submit/response style system to other types of systems. Most telling is here a comparison with a system based on a so-called desktop database. Such tools can be found in many office suites. They are typically not multi-user systems, therefore our example is not typical for such an application, but a similar single user application for desktop databases could use a similar screen layout, e.g. for a personal address list. But in such an application a new database record is created as soon as the user goes to the edit mask. Furthermore, each field entry is entered immediately into the persistent system state. The interaction in these systems is hence single staged. Another important difference is the update policy for pages. In our submit/response style interface, the list shown to the user is not updated until she revisits the page. In desktop databases on the other hand, views of the complete table may be updated immediately.

Submit/response style systems form an important and distinct system class. They are a formal defined class of form-based, request/response systems. Form-Oriented Analysis is an analysis method tailored for submit/response based systems. The structure of the analysis artifacts reflects the two-staged interaction paradigm. In the following we will develop the different artifacts of Form-Oriented Analysis for the given example application. These artifacts will be page diagrams, form storyboards and finally the combination of form chart and model. Concerning formality they range from informal to formal.

2.1 Page Diagrams

Naïve screenshot diagrams can be found frequently in practice, showing pages and connections between them. We have already introduced a naive screenshot diagram of our system in Figure 2.1. Screenshot diagrams, also called GUI prototype, can have various meaning concerning the connections between pages. In our approach we see pages as states and connections between pages as possible state transitions. We discuss other approaches with different semantics in Chapter 3, e.g. WebML [139], a visual language specifically for designing Web Applications. We now introduce our notion of page transition diagrams, which we call *page diagrams*. This diagram type is intended to introduce the modeler into the notion of submit/response style dialogues in a hands-on fashion. It is especially suited as a starting point for the discussion with the domain expert about the intended task of the system. The conceptual insights of the page diagram will be later fully expressed in the form storyboard. A page diagram as shown in Figure 2.2 describes a system as consisting of a finite set of pages, each page offering a finite number of interaction options. A finite description of the interaction options is the conceptually preferable notion in contrast to an unspecified potentially infinite description as it is shown in Figure 2.3. All

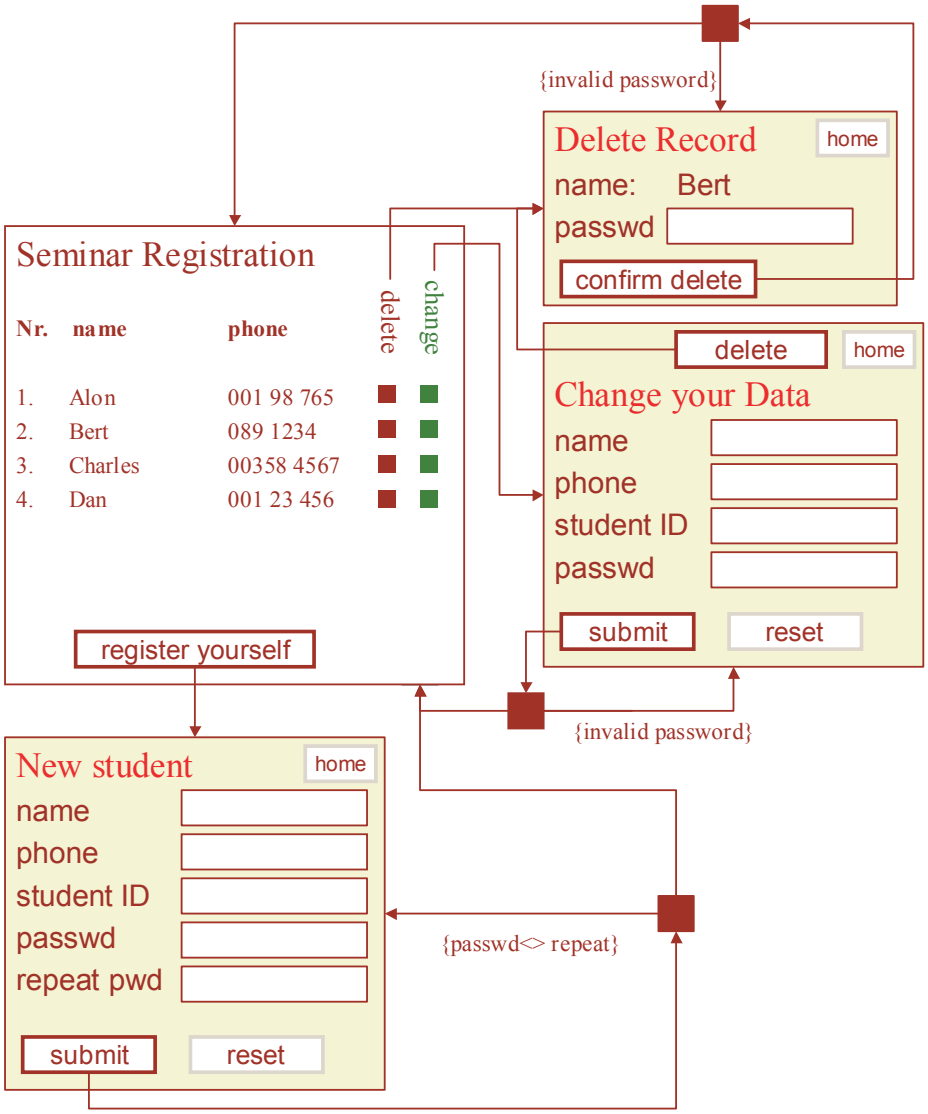

Figure 2.2: Page diagram of the online seminar registration system.

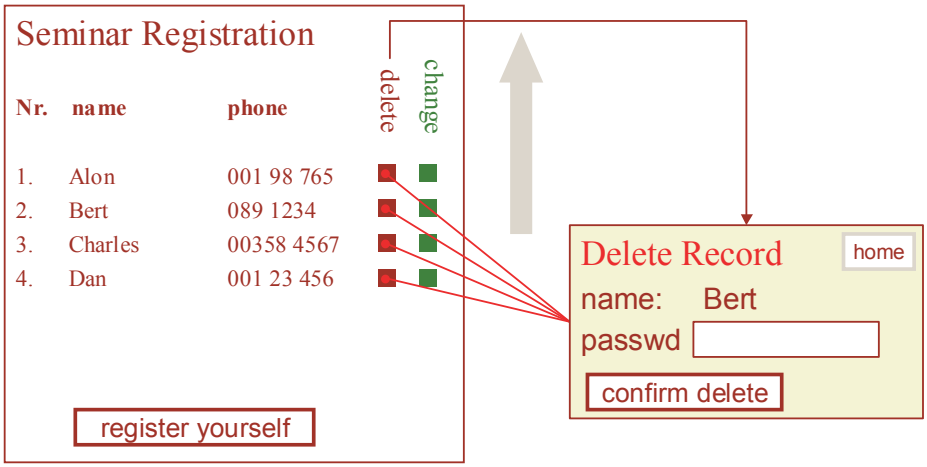

Figure 2.3: A list of options forms a single conceptual option

delete options of the list together form a single conceptual delete option. In page diagrams the pages represent at the same time the set of states of the system. The page diagram is therefore explicitly identified as a state transition diagram. At this point we see the impact of the two-staged interaction paradigm: Only the page changes are modeled as state transitions in the page diagram. The page interaction obviously is implicit in each single page prototype. We want to conceive our page diagram as being executable by a high level kind of a browser. This browser is not a technological browser, but an abstract machine executing our specification. The state transition diagram is a coarse grained state transition since it depicts only page change as state change. This is a major difference in contrast to common user interface modeling with state transitions [54] [76], where the fine-grained page interaction is modeled.

But the most crucial observation we have to make, which immediately opens the conceptual path to form storyboards, is the observation that system response may be conditional: a single page change triggered by the user can result in different response pages, depending on the system state. A typical case is that either the regular response page or an error page is shown. Hence page diagrams as an interface prototype notation offer also conditional transitions. An example is the submission of a registration form in the diagram. The transition has a branching point depicted by a square from which different branches lead to different response pages. These branches are annotated with the conditions under which they occur. In which sense such a system can be formally still seen as a state transition diagram will be explained in the context of form storyboards.

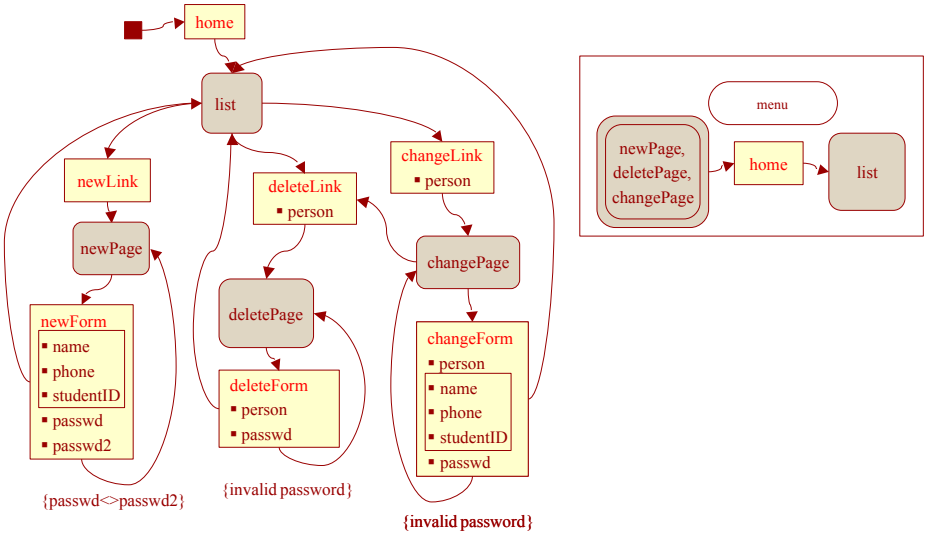

Figure 2.4: Form storyboard for the seminar registration system

2.2 Form Storyboards

Form storyboards take the understanding of submit/response style applications a step further by introducing into the system model one of the major conceptual cornerstones of Form-Oriented Analysis: in form storyboards the system is seen as a bipartite state transition diagram. The bipartite state machine is the basic structure of form storyboards as can be seen in Figure 2.4. In this view the system is alternating between two kinds of states. The first kind of states corresponds to the pages of the system as before. These states are called client pages, they are depicted by rounded rectangles. The system remains in such a client page state until the user triggers a page change. In that moment the record with her previous input is sent to the system. The type of this record, as one can see in Figure 2.4, is specified within the rectangles depicting the second kind of states. They represent the systems action in response to a page change and are therefore called server actions. These states are left automatically by the system and lead to a new client page.

Form storyboards are designed for communication between domain experts and IT experts, but under utilization of the conceptual achievements of Form-Oriented Analysis.

The modeling of the system as a bipartite state transition diagram brings a number of benefits for modeling as we will see in the further development throughout the thesis.

At this point we encounter the first benefit of this bipartite model. Form storyboards give a formal model for the conditional page transition. The branching points in the conditional transitions of the page diagrams are identified with

server actions in the form storyboard. The branching transitions are split into a transition going from the client page to the server action and transitions going from the server action to the client pages. The first type of transitions is called page/server transitions, the second kind of transitions is called server/page transitions.

Since the form storyboard is a bipartite state transition diagram, each unconditional transition in the page diagram which leads from a page to another page has to be transformed into two transitions with an intermediate server action. In the example form storyboard we also see a so called menu feature, which gives a terse description of the fact that a link to the home page is contained on all pages except the home page itself.

2.2.1 Typed calls to the system

The bipartite structure of form storyboards clarifies the difference between pages and forms, which can be seen as the motivation for the term form storyboard. Let us consider the page of the example application with title "Change your Data", which is named *changePage* in the form storyboard. This page is dominated by the form for changing the prefilled personal data. However, the page and the form are different concepts, as can be seen by the fact that the page also offers another option for deleting the record. Each client page offers a finite number of interaction options in our conceptual view already introduced for page diagrams. Each interaction option corresponds to a page change.

Page change is seen as the submission of a data record together with a command to the system. Hence page change can be seen as a method call by the user. This method call is editable. Its actual parameters are edited by the kind of interaction we have called page interaction, either by textual input, also called direct input, or by selecting from a list. In form storyboards, the signature of a server action is written in the rectangle in order to resemble a form.

The important web technology often uses links to choose a certain parameter. Consider again the main page of our seminar registration system. The student can delete himself by clicking the delete square beside his entry. This could be realized by a link. At this point we can illustrate the difference between this technical perspective and the high level perspective taken by our Form-Oriented Analysis method. The difference between links and forms is a purely technical one, residing on the implementation level of the system. On the analysis level on the other hand such a link is simply a selection. The analysis level is not concerned with the technical realization, not even with the layout, i.e. the selection could be realized as a list of links as well as a list of radio buttons or a pull down list. In other words, a list of links is equivalent to a radio button group with a submit button. So analysis deals with the logical effect of the interaction option.

Since in Form-Oriented Analysis page change is seen as an editable method call, the server action represents the state of method execution. In form storyboards each server action can have more than one ingoing edge, representing

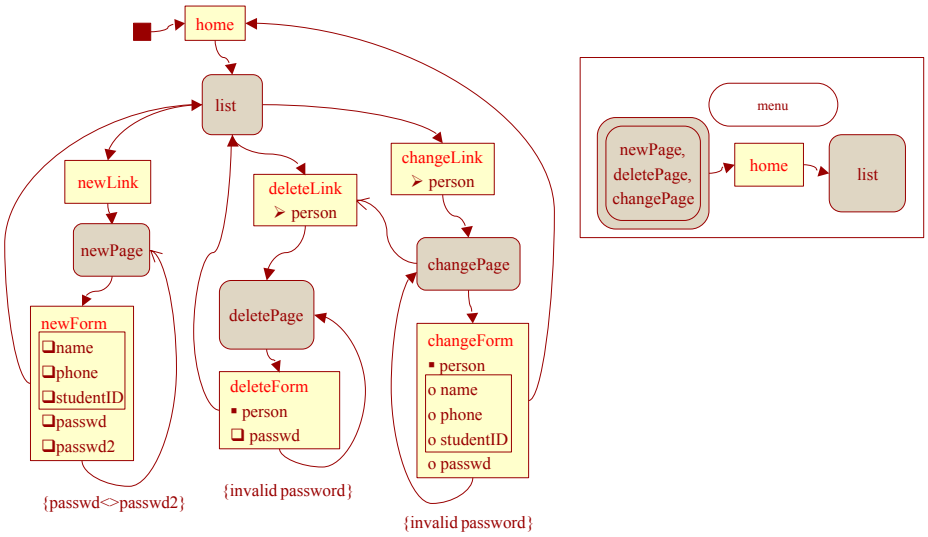

Figure 2.5: Form storyboards can be annotated with interaction information

calls from different client pages to the same method. All ingoing edges to the same server action must provide the same parameters. In form storyboards the parameters of a server action are specified within the square representing the server action. The parameters provided by a form and submitted to a server action are considered as a single record data type which is called the superparameter of the server action.

Page/server transitions represent the submission options that are offered by the page i.e. the forms offered by the page.

In order to support form storyboards as a tool for efficient communication with the domain expert, form storyboards can be annotated with information about the interaction type used for submitting the form parameters. Figure 2.5 shows the form storyboard with annotated interaction information. It differs from the form storyboard version in Figure 2.4 in the icons preceding the parameters. These icons explain, how the parameter is provided. The arrowhead icon indicates a selection or a link list. The square indicates a text input field. The empty bubble indicates a text input field with default value. The dot indicates a hidden parameter.

The form storyboard provides obviously a more abstract view of the system interface than the page diagram, however the form storyboard can still be seen as an intuitive high-level prototype of the system. In the form storyboard each client page together with its accessible server actions give a representation of the page with its interaction options, as it is shown in Figure 2.6. Therefore each such subgraph is called a *page image*. Page images can share server actions, as can be seen in the example of the deleteLink server action. This server action is contained in the page image of the main page as well as in the page image of

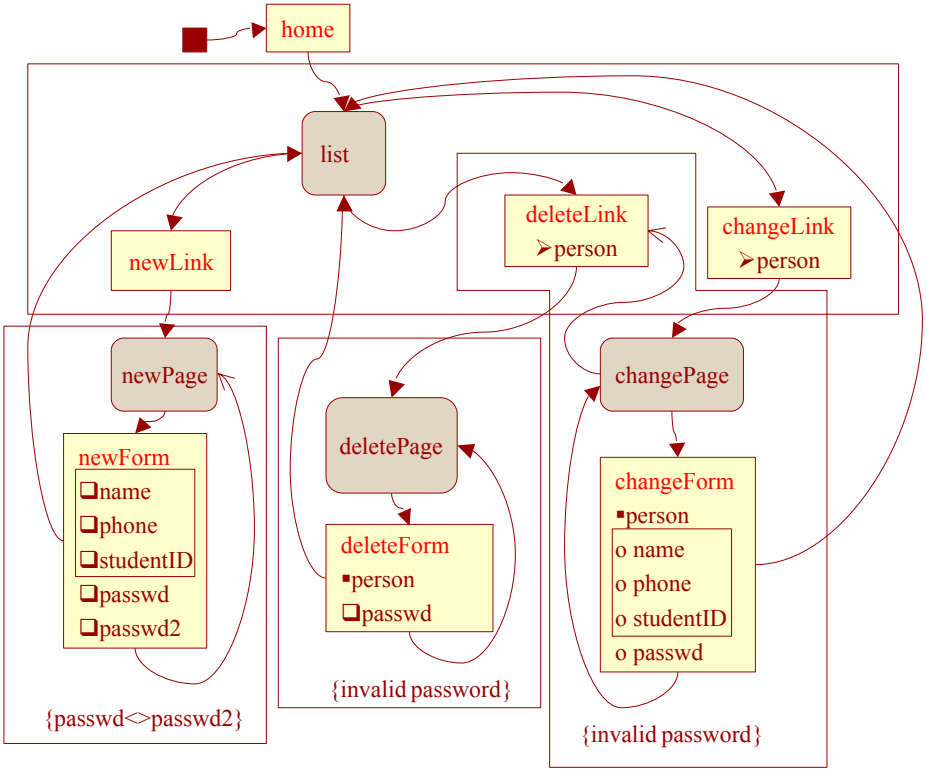

Figure 2.6: All page images of the system.

the change dialogue.

A server/page transition indicates that the server action may produce a response page of the addressed type as a result of its action. Therefore the server/page transitions must be annotated by the condition under which this transition is chosen. These conditions must be mutually exclusive or must be ordered. One outgoing transition of a server action may carry no condition and then represents the default case, also called the else case. In the example the transition from the newForm server action to the home page is the default case and therefore carries no condition. The other server/page transition carries the condition that the password is not correctly retyped.

Form storyboards are labeled directed multigraphs. Two states can be connected by more than one edge in the same direction. In the case of page/server edges, this represents several distinct submission options. An example may be an online shop that has on each page two types of offers: featured articles with in-depth information and an article list with only a short info. The purchase option addresses the same server page, but the modeler wants to distinguish these two classes of purchases.

2.2.2 Patterns for submit/response style systems

Before we introduce the composition mechanism for form storyboards we make some reflections about the overall structure of the system. In the form storyboard of our system it is clearly visible that the system offers three different interaction strands, namely for insertion, deletion and change of enrolments. Each of these dialogue strands is shown in a column-like fashion in the form storyboard. Furthermore each of these interaction strands obviously consists of three dialogue states which are connected with the home page to an interaction circle. These three interaction strands follow a typical pattern for submit/response style systems, which we call the link-page-form pattern. This interaction pattern begins with a server action that represents the start of the interaction. In the case of the server action changeLink the user selects the data record under consideration by choosing his own enrolment. The link leads to the page on which the actual interaction is performed. This client page is typically characterized by the fact that it is dominated by a single form. The server action representing this form therefore is the third dialogue state involved. This pattern is so commonplace, as one can see in our example system, that a special naming convention for the dialogue states is helpful. As one can see in the example, the naming convention is to take the name of the functionality and to append the role name within the pattern e.g. we get changeLink, changePage, changeForm. This pattern applies for such actions which involve two page changes. Simpler interactions can of course use only a single server action for completion, e.g. if a system has a delete function which needs only a single click without confirmation dialogue.

2.2.3 Feature Composition

Form-Oriented Analysis offers a powerful composition mechanism called feature composition, which is chiefly used for form storyboards and the form charts introduced in the next section. Feature composition allows the composition of the complete system form storyboard from partial descriptions of the system called feature storyboards or simply features. A feature storyboard is a form storyboard that describes a part of the system. The graph underlying a form storyboard has been defined as a bipartite directed labeled multigraph, and the composition mechanism for features is basically graph union. If two features are composed to a single form storyboard the node and edge sets are united. Nodes of same name are identified, edges between the same nodes of the same name are identified as well. The composition of features is shown in Figure 2.7. There are some rules that two features have to follow when they are to be composed. These rules are explained in Chapter 4 together with techniques for adapting features for composition. Feature composition is an astonishingly powerful composition mechanism given its simple semantics.

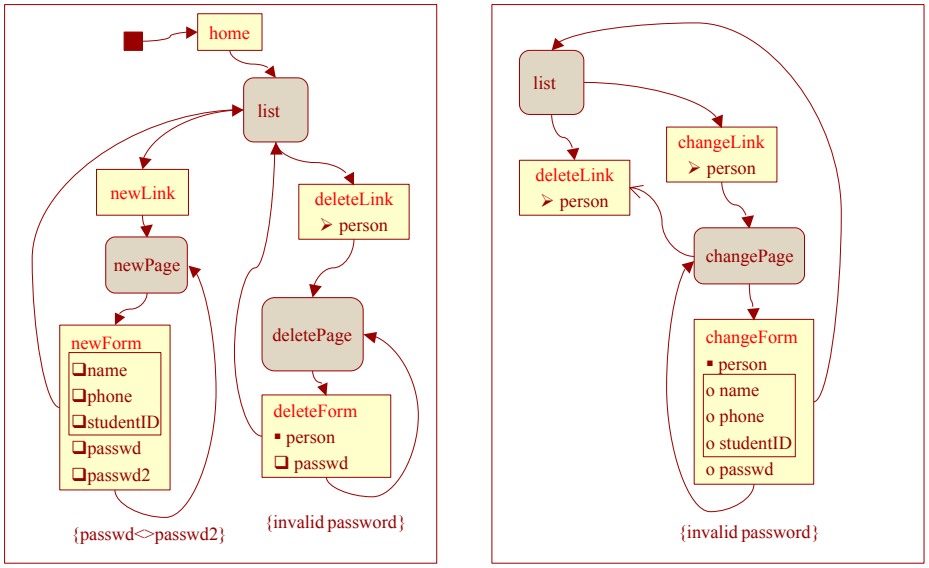

Figure 2.7: Two features which give together the complete system.

2.2.4 Communicating Form Storyboards

Form storyboards are designed for communication and joint development between domain experts and IT professionals. Therefore form storyboards present a self contained system view comparable to page diagrams. At the same time form storyboards achieve a much higher level of abstraction than page diagrams. Nevertheless form storyboards may be easily communicated with the domain expert.

2.3 Form Charts and Model

The final and semantically most expressive diagram of Form-Oriented Analysis is the form chart. The form storyboard already expresses the key elements of the submit/response paradigm. The form chart has now the task to make the analysis model amenable to formal constraint writing and coupling to the semantic data model, and it is therefore accompanied by two other diagrams, first the semantic data model and second the data dictionary mediating between both. Furthermore a textual document containing formal constraints has to be seen as attachment to the form chart. The document bundle consisting of form chart with attached constraints, data dictionary and semantic data model comprise the *form-oriented specification* of the system. The form chart will be precisely explained in Chapter 4, together with the dialogue constraints. A complete specification of a system is often a valuable goal, but in many cases it may not be practically achievable. Our method allows the modeler to create a

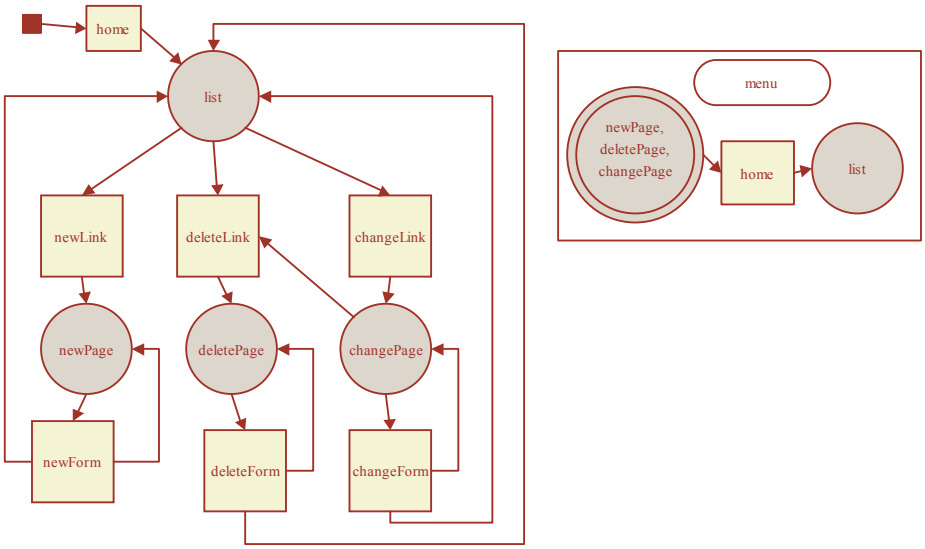

Figure 2.8: Form chart of the seminar registration system.

complete specification, but of course it is usable for partial specification as well and therefore gives the modeler the freedom to choose the degree of precision which seems appropriate for the project.

The form chart is the successor diagram to the form storyboard. The form chart as shown in Figure 2.8 contains the bipartite state machine. Server actions are depicted as rectangles and client pages are depicted as bubbles. In the form chart only the names of the states and transitions appear. The form chart also contains the start marker.

Hence the form chart does not contain the signatures of the server actions as they were depicted in the form storyboard. Instead, the signatures are contained in the second new artifact type, the data dictionary, shown in Figure 2.9. The data dictionary contains types and is therefore a class diagram in the terms of modern modeling languages like the UML. However, the data dictionary types are of a special kind of data types, namely algebraic data types. Instances of these types are immutable values. The types can have structure, but only a hierarchical structure, namely composition. They represent sent messages, comparable to written and sent documents. Remaining in that metaphor, once you have sent a letter the content is unchangeable. In the data dictionary there must be a message type for each form chart state, and it must have the same name, except that the initial letter is lower case in the form chart, but upper case in the data dictionary.

The message represents the signature of the state of same name. Each time this state is entered, a new message of this type has to be provided. we have already encountered signatures of server actions in the form storyboard, but in the form chart we specify also signatures for the client pages. These client

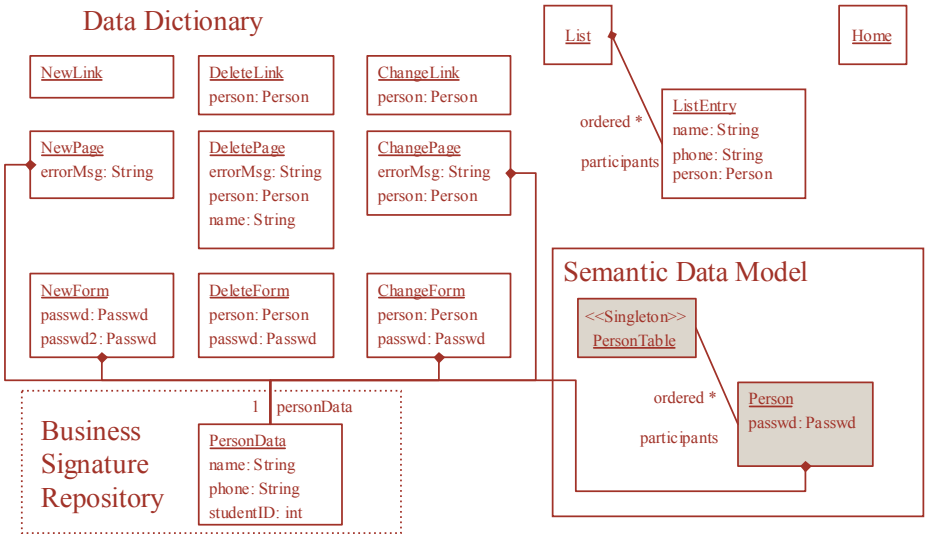

Figure 2.9: Semantic data model and data dictionary

page signatures represent the information shown on the page. The page content is immutable. A page shows the same content to the user until she triggers a page change and therefore gets a new page, although possibly from the same type. Page interaction, i.e. user input in forms is not considered a change of the page content, but preparation of a new message. The fact that now the data dictionary contains the information shown on pages as well as the information sent back to the system as part of a page change is important with respect to the specification of so called dialogue constraints. Indeed one of the main advantages of form charts is that it allows more elaborate constraint writing than the form storyboard. We want to be able to express e.g. that the data record the user chooses for deletion must have been presented on the page. Such a constraint is called client output constraint. It is written in the following style.

```
list to deleteLink {
clientOutput:
source.participants.person->includes(target.person)
}
```

As we see in this example, we need the signature of the client page as well as the signature of the server action, called source and target, in order to set both in relation to each other. There are a number of different types of dialogue constraints, and they form together the dialogue constraint language, DCL for short. The DCL constraints are typically written in an attachment of the form chart, although in principle they can be written into the form chart diagram it-

self. The last diagram in the bundle that forms the specification is the semantic data model. This diagram is the conceptual data model that specifies the system state. Pure data record specifications which are needed by both, the semantic data model as well as the data dictionary, are put in a separate part of the data dictionary, the business signature repository. In our example the semantic data model is rather simple and consists mainly of the class holding the student information. The semantic data model is connected with the data dictionary again through the dialogue constraints, but also through so called opaque references. Take the client page `list` as an example. The message type `List` contains a list of different `ListEntry` objects that contain only the information presented to the user. This `ListEntry` submessage contains an attribute of type`Person`, the class from the semantic data model. Such a reference from the data dictionary to the semantic data model is called opaque reference. As one can see, if one follows the message types associated e.g. with the delete subdialogue, this reference is passed along the dialogue and hence specifies, which person object is subject to the deletion process. The reference is passed along the form chart, yet the reference is opaque in the sense that only through certain operations that again access the semantic data model the content of the person object can be accessed. The whole semantic data model forms a single data abstraction module with possibly as many opaque reference types as it contains classes. The opaque references are therefore the border of the data dictionary. The reference itself is part of the message, but not the referenced object. Therefore the object can change without violating our demand, that messages are unchangeable.

2.4 Summary

We want to sum up the lessons learned from our hands-on introduction to Form-Oriented Analysis. We have seen the following artifacts of Form-Oriented Analysis applied to an example application.

2.4.1 Page Diagrams

Our page diagrams foster understanding of the following issues:

- Submit/response based interfaces are a distinct class of interfaces.
- User interaction in these interfaces is two-staged, consisting of volatile page interaction and page change affecting the system state.
- Page interaction is well understood and can be offered as an application independent browser.
- The system response is conditional. Therefore the state transition from a page can lead to an intermediate state that leads again to alternative response pages.

- An adequate representation of the system by finite page states and finite interaction options has to be found. The conditional system response is based on the system state beyond this finite state model.

Page Diagrams may contain mock-up prototypes of pages as shown in the example in Figure 2.2. Submit/response based interfaces should not be confused with formlike interfaces without a two-staged paradigm. Examples for the latter kind of interfaces are desktop databases.

2.4.2 Form Storyboards

In form storyboards each page change leads to an intermediate state, therefore the state transition diagram is bipartite. The page states are called client pages, the intermediate states are called server actions. The transitions leaving a client page are called page/server transitions or options; the transitions form server actions to pages are called server/page transitions. In form storyboards, server actions have a record parameter type which is inlined into the graphical element representing the server action.

Form storyboards foster understanding of the following issues:

- The state transition diagram is bipartite. A client page is left by a form submission and leads to an intermediate server action. Server actions are left automatically depending on the system state beyond the finite state model. Hence in a dialogue client pages and server actions alternate.
- Forms have a record type which is the parameter signature of a server action. The same server action can be called from different pages.
- As a consequence, the system interface offers a set of state transitions with typed parameter list, but each state transition may be accessible only from certain client pages.
- Subsumption of links under forms. Links and forms are editable or selectable method call templates.
- The system response contains the new interaction options.
- A page state together with its accessible server actions is a high-level page representation, the so-called page image. The form storyboard is a high level prototype.
- Form storyboards offer a simple composition mechanism, namely graph union tailored to form storyboards. Vice versa, the complete form storyboard can be decomposed into smaller subdiagrams called features. The composition mechanism is called feature composition.

Form storyboards are based on the submission form metaphor. The user fills out forms and submits them to the system. In submit/response systems, forms are volatile until they are submitted.

Form storyboards are intended for communication of the main conceptual achievements of Form-Oriented Analysis with domain experts. Hence form storyboards are less prescriptive than form charts, in order to allow adaptive levels of formality. Especially form modeling with opaque references is not required, as it will be in form charts.

2.4.3 Form Charts

Form charts are the final and strict notational concept for Form-Oriented Analysis. They are given by rigorous semantics and rules of usage.

- The user interface is set in relation to an analysis model consisting of persistent and session data. The connection is established through a data dictionary.
- Object selection by opaque references. Pages can offer collections of objects and the user can select objects from the collection and pass them back. No primary keys are passed across the state transition dataflow.
- Rich annotation of the state transition diagram by so called dialogue constraints given in the dialogue constraint language DCL.
- In Form-Oriented Analysis the responsibility for a system functionality is not artificially delegated to participating objects. Instead system functionality is modeled in procedural style and technically delegated to the class representing the parameter list.