Chapter 6

Conclusion and Reflection

In this thesis we have discussed Form-Oriented Analysis, a new analysis method specifically designed for so called submit/response style applications. Form-Oriented Analysis introduces different artifacts, namely page diagrams, from storyboards and form charts, which differ in formality and expressiveness. The final and most expressive artifact, the form chart, was defined formally in UML. Furthermore Form-Oriented Analysis is designed as a firm basis for tool support, and the fully implemented Gently tool is presented in the Appendix B of this Thesis.

6.1 The Main Contributions of Form-Oriented Analysis

This thesis presents two kinds of contributions. First the contributions of Form-Oriented Analysis to the modeling process, i.e. contributions for the working developer, including the contributions by the Gently tool. We can summarize that Form-Oriented Analysis brings the following benefits for the analysis of submit/response style systems:

- It is an elaborated approach for modeling submit/response style systems with bipartite state machines.
- It gives precise semantics to form charts, which are constraint-annotated bipartite transition diagrams for modeling the user interface while unifying forms and links as strongly typed method calls. It defines decomposition and refinement concepts.
- The complete system specification is structured in four artifacts, the form chart, the dialogue constraints, a data dictionary and a semantic data model.

- Form-Oriented Analysis offers page diagrams and form storyboards for eliciting functional requirements and communicating them with domain experts.
- The relationship between Form-Oriented Analysis models and state-of-the-art proposals for software architecture of web based information systems is well understood, and gives rise to useful tools like the Gently Tool.

The second kind of contributions of Form-Oriented Analysis concerns theoretical advances achieved in the formal semantics of form charts. These advances go beyond the immediate application within Form-Oriented Analysis, however the application of these theoretical concepts in Form-Oriented Analysis are a more than sufficient justification of them. During the formal treatment of Form-Oriented Analysis we have encountered the following notable theoretic contributions.

- A specification of finite state behavior by a bipartite state machine. The finite state machine alternates between receptive states, which listen for the next event, and reactive states, which react to an event and are left automatically in order to return to an receptive state.
- The form chart as a formal artifact for the special purpose of specifying submit/response based systems by dialogue constraints.
- The operational semantics of state history diagrams as class diagrams derived from the shdframework.
- Opaque references for the data dictionary.

6.2 Further Work based on the Contributions

This thesis introduces a bipartite finite state model for a well-defined application area. Finite state models are one of the most important and basic modeling techniques used in theory and practice. The state chart formalism is not bipartite, Petri nets on the other hand use a bipartite graph to model a transition system. The bipartite structure of Petri nets has a different aim as we have argued earlier. However, our bipartite state transition model is useful, since it offers a natural separation of concerns: One class of state transitions models the input to the system, the other class models the reaction of the system. In further work we want to study, if bipartite state transition models provide valuable models in other areas, too.

The operational semantics of state history diagrams given in this thesis has the advantage to be tightly integrated with the system model and allows powerful combined reasoning about the finite model and the semantic data model at the same time. It is natural to study, whether this operational semantics provides added value in other areas of finite state modeling, too.

The notion of opaque references was introduced as a high level concept to model references in immutable messages, hence as an abstraction from ID types or from primary key fields. In future we are looking for a class diagram semantics, which offers opaque references as first class citizens. Furthermore opaque references may promise a valuable contribution for modeling automated interfaces, currently discussed as web services.

6.3 Reflection on Submit/Reponses Style Systems

Throughout the thesis we have seen, how submit/response style systems can be modeled in a semantically well understood framework. We have seen, how to specify such a system along its usage through a formbased interface, which is easily combined with state-of-the art modeling techniques like a class diagram and a constraint language. Now at the end of the thesis, after a formal treatment of submit/response style interfaces, we feel free to discuss broader aspects of such interfaces. We want to recall the properties of such interfaces and discuss possible extensions. Some advanced concepts especially concerning active content we have discussed earlier in the thesis. Here we want to discuss, whether there is still a natural place and a need for submit/response style interfaces. Such interfaces are often considered as bare metal legacy technology. we however will argue that there may well be reasons, why submit/response style interfaces are here to stay because of subtle cognitive advantages.

We want to reflect on the main properties of submit/response style interfaces.

- Submit/response style interfaces collect user input in the style of input forms. Input forms can be understood as a metaphor, namely as a direct translation of paper forms into human-computer interfaces. Input forms are composite structures of input elements.

- The notion submit/response style refers to the screen update policy, namely that all screen updates have to be triggered by a dedicated user action. The user actively submits data or a query, instead of watching a constantly updated view.

6.3.1 Cognitive Advantages

Form-based interfaces have clear advantages for the self-explanatory character of a system. The usage of the system is intuitive, since it is guided by a paper form metaphor. Notable is however the importance of the submission process, because of which we want to characterize the metaphor as *submission form metaphor*. The difference between temporary input and submission, or "sending", is intuitive and fosters the user's understanding of the system. The form-based metaphor has a multi-tier structure in its own, without fixing an implementation. The two classes of interactions structure the work of the user into the work intensive frequent page interactions and the punctual and atomic interactions of the "serious" kind, namely the page changes which also happen to be the conclusion and separation of logically disjoint bunches of work. The submit/response style character brings the user in command of the timing of her system usage. It protects her from irritating disruption of her work by incoming information.

In form-based interfaces the submission of a form is an operation that has exactly the semantics indicated by the metaphor. In computer science terms we have compared the submission of an actual parameter list with a method name. The submission form metaphor views interaction with the system as filling out virtual paper forms and submitting them to a processing instance, which represents the core system.

The metaphor has the qualified name submission form metaphor, because other form interface types can be found as well. E.g. desktop databases as they are found in office suites allow form style interfaces, which possess page navigation buttons. Input in this form immediately changes the model. We call such a form style interface a *formlike view* Applications using formlike views are in principle required to have synchronous views on the data: if two formlike views currently open show the same data, and the data are changed in one formlike view, then the other formlike view is immediately updated. Many implementations however have to stick to polling mechanisms, which leads to latency effects in the update process. Well-known and even worse examples are file managers, which recognize state changes frequently only after manual refresh. It is important to recognize, that the necessary refresh is in this case a bad implementation, while the reload is a feature in the case of the submit/response style applications.

In desktop databases the model state is the persistent state. Other applications with formlike views have nonpersistent state, e.g. spreadsheets.

The submission form metaphor has the advantage of having a clear semantics. The two-staged state change due to the two-tiered model is integral part of the metaphor. This is quite in contrast to e.g. the important desktop metaphor: Consider the important drag and drop feature, which is at the very heart of the desktop metaphor. Drag and Drop means regularly either copy or move, hence can lead to two different effects.

The submission form metaphor is accompanied by the *response page principle* for showing reponses of the core system. The submission of a form is a page change, i.e. the page that hosted the form is hidden and a new page is shown. This new page is the response from the server. The response page has three important functions:

- Notifying the user of the of the submitted form.
- Showing new information to the User.
- Offering new interaction options.

The immediate status of the submitted form is the systems immediate response to the form. Depending on the business logic this may or may not be the completion of the form processing.

- Consider the entry of a new date in a web calendar tool. The response page is the new calendar view with a short notification message. The form has been completely processed.
- Consider the submission of an order in an online brokerage system. The response page is a notification of reception. The execution of the order however is taking place asynchronously.
- Consider the submission of an email in a Mail account on the Web. The response page logically is only a notification of some overall validity of the submitted data, e.g. the recipient's address contains an at-sign. The completion of the intended effect, namely the delivery of the email, is not acknowledged at all.

6.3.2 Advanced Form Concepts

There are advanced form and widget concepts, which are not ubiquitous in current types of submit/response systems, but which show, that form based interfaces can have a comfortable appearance and functionality.

Strongly Typed Direct Input Widgets

Direct input widgets are in the first place textual input fields. Strongly typed input widgets allow only input of correct primitive type. Strongly typed widgets are directly related to formatted widgets. Examples are fixed length, integer, fixed point, date, IP numbers. Certain special formatting widgets can be seen as widgets for complex data types. E.g. IP address widgets can be built by four integer widgets. In this case it is desirable, that the framework allows to define the dot key as an alternate key for focus shift: hence, if the user enters the IP address in the keystroke sequence with dots, the IP address is correctly placed across the four widgets.

Auto Complete Widgets

If the user is required to enter a selection from a large set, neither radio buttons nor select lists are applicable. An example is the input field for a train station. Here an adequate input widget is an auto complete widget. If the user enters a character sequence, a list with all choices with this sequence as a prefix is offered.

6.4 Summary

Submit/response style applications are a widespread class of important systems. We have seen that focusing on this system class allows us to define a rich analysis method, which allows the convenient specification of such applications. We have also experienced that this analysis method yields valuable insights into the semantic structure of such systems, as well as interesting contributions to modeling in general.