Chapter 3

Related Work

In the last chapter we have gained a first impression on how we can obtain an intuitive analysis model of a typical submit/response style system by the use of Form-Oriented Analysis. In this chapter we give a comparison with other analysis and modeling techniques that are applicable. We will see, how in a number of aspects Form-Oriented Analysis can be seen as a consequent extension and continuation of existing techniques, including most recent proposals. We will also discuss, in which aspects Form-Oriented Analysis goes beyond these approaches and solves typical problems encountered in modeling submit/response based systems. In this chapter we give a survey of related proposals from a number of domains. First we will discuss the two most important analysis techniques, Structured Analysis and Object-Oriented Analysis. Afterwards we deal with more particular modeling techniques e.g. modeling techniques for user interfaces.

3.1 Structured Analysis

Early works on Structured Analysis are Gane and Sarson [126] and DeMarco [63]. Structured Analysis is a very successful approach to both business modeling and system modeling and even today, it is in extensive use in the industry. It combines hierarchical data flow diagrams, sum-of-product data specification, local functionality specification and later [66] entity-relationship diagrams. The method is deliberately ambiguous with respect to the semantics of the several notational elements of the data flow diagrams and therefore heavily relies on the intuition of the modeler. The method is undoubtedly well suited for form based tasks, as can be seen by similar early contributions [148], and it especially works well for the analysis of combined manual and automatic processes.

Structured Analysis defines several analysis-level artifacts, namely Data Flow Diagrams (DFD), Data Dictionaries, Pseudocode (called Structured English), Decision Tables and Decision Trees. They together form the structured specification called target document. Conditions for the mutual correctness of the diagrams are defined as so called balancing rules. Balancing rules cover type correctness rules as well as name space rules. Modern Structured Analysis furtermore uses Entity-Relationship Diagrams [66] for the semantic data model, which are balanced against the data dictionary.

3.1.1 Leveled Data Flow Diagrams

A DFD describes the data flow in the system between processes and other instances. A data flow diagram is a directed graph. Edges and nodes are labeled. There are three kinds of nodes: processes (bubbles), stores and terminators. The edges are called flows. DFD's are typed, the type system is given by the data dictionary concept. The labels on the edges and on stores represent entries in the data dictionary. Terminators represent incoming or outgoing data flows. DFD's can be decomposed, but only processes are subject to decomposition. A single process can be replaced by a complete DFD called the subdiagram. The dependency between edges connected to the process and the terminators in the subdiagram is explained in a balancing rule: The data flows connected to a process must be constructed from the data flows connected to terminators of the subdiagram for that process. DFD's can be conceived as a concurrency model. The edges of the data flow diagram represent pipes capable of messages with a fixed type specified in the data dictionary. If a process has two ingoing data flows, the diagram does explicitly not specify e.g. whether the process needs each time messages from both ingoing flows or only one message from an arbitrary ingoing flow [128]. This ambiguity is designed to focus deliberations on the aspects of pure typed flow of data. The decomposition of DFD's is functional decomposition. Since the processes are decomposed by subdiagrams, this resembles the specification of subprograms in programming languages. However, the sub-DFD's cannot be reused in different places of the specification. The hierarchical decomposition is hence only targeted at the partitioning of the specification.

3.1.2 Data Flow Diagrams vs. Form Charts

Data flow diagrams may seem superficially similar to form charts, but are completely different. While a form chart is bipartite, a data flow diagram is not. In a data flow diagram the different types of nodes may be combined freely. Moreover, the semantics of a dataflow diagram is different from the form chart semantics. Especially the data flow diagram is no state diagram; on the contrary it exhibits parallelism and nondeterminsm as explained above. There are no strict temporal relations between ingoing and outgoing messages in a DFD. In a form chart obviously a state must be entered before it can be left exactly once on exactly one outgoing edge. Furthermore the decomposition mechanisms are completely different. While in DFD's a process can be decomposed into a complete dataflow itself, in form charts such a decomposition is neither intended nor possible. Instead the powerful feature decomposition of form charts is used,

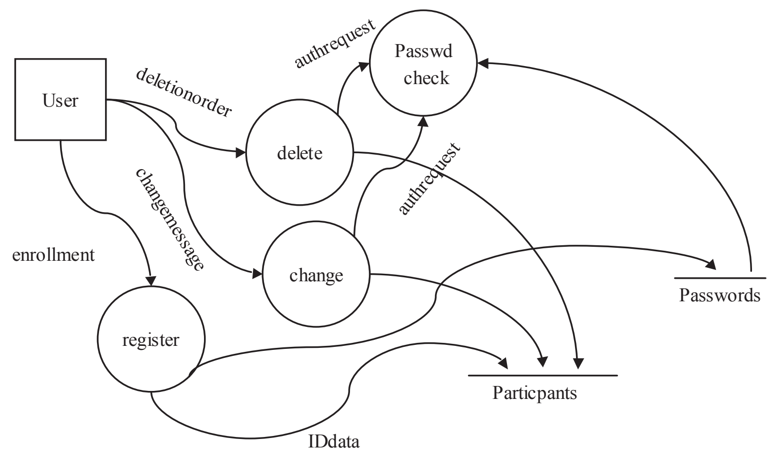

Figure 3.1: Data flow diagram of the seminar registration system

which has been defined as graph union. Figure 3.1 shows a data flow diagram for the seminar registration system.

3.1.3 Data Dictionary

The data dictionary is a list of hierarchical data type definitions. Formally this means that data dictionary entries are algebraic data types. Data dictionary types represent messages sent along flows or stored in data stores. The data dictionary in Form-Oriented Analysis is essentially the same as in Structured Analysis with minor changes. Data dictionary entries are immutable values. A change to an entry in a store has to be understood as an exchange of whole messages.

The data dictionary of Form-Oriented Analysis differs chiefly in two points, first in the dealing with EITHER-OR type construction, secondly through the concept of opaque references. Concerning the dealing with EITHER-OR type construction, there is no special structural concept necessary in Form-Oriented Analysis, instead the EITHER-OR is an Xor constraint on a compound type, expressing that only one of the components can be given at a time. The second difference is with respect to references to stored entities. In Structured Analysis there is most remarkably no explicit concept for keys. Keys are just primitive values. In Form-Oriented Analysis, the key concept is made explici in form of the opaque references.

3.1.4 Object-Oriented Extensions of Structured Analysis

Given the success of Structured Analysis, some work has been done in constructing object-oriented extensions of Structured Analysis. The Object-Process Methodology OPM [112] [113] deliberately avoids the partition of different modeling artifacts in different diagrams and uses a joint representation of processes and object-oriented information. It sets itself in contrast to object-oriented modeling techniques as they are discussed in the next section. In the FOOM methodology [115] [116] an object-oriented extension of Structured Analysis is given in such a form that two initial diagram types are used, a class diagram and a so called OO-DFD. The authors of FOOM claim to gain superior results in a comparison with OPM [117]. These methods aim at employing the advantageous elements from Structured Analysis as well as from object-orientation in a single method.

3.1.5 Structured Analysis and Form-Oriented Analysis

As we have pointed out earlier, Form-Oriented Analysis is an artifact oriented approach, which brings its chief innovations in the semantics of form charts. Form charts can be used immediately in Modern Structured Analysis simply as a further document type, since the form chart relates itself to a semantic data model and a data dictionary as it is already used in Modern Structured Analysis. The semantic data model for form charts is in this thesis presented as an UML class diagram, but it uses only such features, which can be directly translated into Entity-Relationship Approach. In that sense Form-Oriented Analysis is a contribution to Structured Analysis, too.

3.2 Object-Oriented Analysis

Object-Oriented Analysis (OOA) has become from 1990 onwards a widely discussed analysis technique [83]. There is a remarkably huge number of different OOA methods. OOA is based on the intuition that the object-oriented metaphor, which was at this time already firmly introduced in the programming languages world can serve as a viable metaphor for analysis as well.

3.2.1 Origin of Object-Orientation and its Application in Design

Object-orientation itself was introduced by the language Simula [131][129][130], later the language Smalltalk developed at Xerox PARC took up the object-oriented metaphor. Object-orientation is indeed successful on the design and implementation level. Let us first recall some facts about Object-Oriented Design (OOD) before we discuss OOA in depth. OOD is especially relevant with respect to the justification of OOA, since one of the main goals in a pervasive object-oriented development process is to create synergies through the tight paradigmatic integration of the different development steps. Several principles that describe the advantages of OOD have been identified [143] [144] [94]. One important principle is the Hollywood Principle *don't call us, we call you*, which refers to the object-oriented style of customizing behavior by overwriting methods. Another key principle is the *don't ask, what kind* principle, which refers to the way in which subtype polymorphism is used by client objects. Both principles support separation of concerns.

3.2.2 Identifying Analysis Classes

Object-Oriented Analysis takes a different approach to system modeling than Structured Analysis. Structured Analysis proceeds by describing the key functional processes in the problem domain and structuring them in the data flow diagram. The data model takes originally the form of a data dictionary that annotates the data flow diagram.

In Object-Oriented Analysis according to Coad and Yourdon [83] the analysis starts by identifying the concepts of the problem domain, which are conceived as classes of objects. Objects are seen as encapsulations of data in form of attributes. In a second step the interconnections of classes are elicited in form of associations and generalization hierarchies. Behavioral aspects are identified in the next stage in the form of services offered by objects.

The results of Object-Oriented Analysis vary strongly. Kamath et al. report significant advantages [125]. It has been observed in [141] that OOA is best applied to a new project and does not work well with reengineering of legacy applications.

3.2.3 The Use Case Driven Approach

Object-Oriented Analysis through a use case driven approach was popularized by Jacobson [50]. The use case driven approach begins by identifying use cases of the problem domain, together with actors which use them, in a coarse grained diagram. A use case diagram for the seminar registration system is shown in Figure 3.2. Use cases are described by scripts that have the task to storyboard scenarios of user interaction [85]. The scripts are textual, sequential descriptions of user interactions. Based on the use cases analysis classes are identified, especially boundary objects, with which the actors are conceived to communicate. The scripts can now be transformed to interaction diagrams showing the actors enacting on the boundary classes.

The use case driven approach to object-oriented software engineering has become a widely acclaimed analysis technique. From the beginning [114] [50] to state-of-the-art versions [51] of this approach the recommended human computer interface specification techniques are oriented towards the modeling of object oriented GUIs in contrast to form based interfaces. As a consequence the use case driven modeling does not have the two staged structure of interaction as it is introduced for form-orienetd analysis.

The use case driven approach may yield a description specification of an analysis-level system interface description which is not a specification in the strict sense: The system usage is described only through singular cases, which are depicted in a sequential manner, e.g as a message sequence chart. Message sequence charts are expressively weak [118]. Furthermore the system usage is

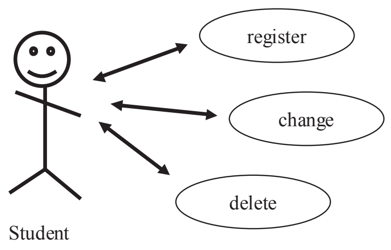

Figure 3.2: Jacobson style use case diagram of the seminar system.

depicted in a nonhierarchical manner. The interaction with form-based systems is modeled as a fine-grained sequential activity [50].

Use case driven modeling becomes even for small examples unwieldy. Use case driven modeling is intended to cover all system types from bottle receivers to warehouse management systems.

3.2.4 Object-Oriented Analysis and Form-Oriented Analysis

Comparing the artifacts of OOA and Form-Oriented Analysis shows us, that they differ in information content, but are in prinicple compatible. The form chart is a specification document, in that it specifies all possible behavior concerning the part of the model expressed in the form chart. An interaction diagram on the other hand specifies only exemplary behavior. However, if an interaction diagram is seen as useful for the undestanding of the system, interaction diagrams and form charts can be combined for the same system. Furthermore the form chart is related to a class diagram, as it can be used in OOA. Hence again, as in the case of Structured Analysis, especially form charts are compatible with Object-Oriented Analysis.

3.3 The Specification Language Z

A well-known approach to formal specification is the language Z [149]. The aim of Z is perhaps best pointed out in the phrase *specification using mathematics* [151]. In Z systems are specified as functions. For stateful systems Z uses an automaton model for the specification of system states. The model, called state machine model in [150], describes the system as an automaton with a state from a potentially infinite state space and a state transition function. The system description therefore consists of a specification of the space of possible states and the operations on the state. Z allows to specify update operations that lead to state changes, Δ operations for short, and pure queries, called Ξ operations.

There are certain connections between Z and our approach as well as notable differences. First of all Z as well as our approach uses an automaton model for the system description, but it is important to understand that the counterpart to the state in a Z specification is not the state of the form chart, but the current object net over the semantic data model in Form-Oriented Analysis. It is the form chart, where Form-Oriented Analysis goes beyond the Z specification paradigm. In Z all operations are at first hand conceived as being applicable at any time. In Form-Oriented Analysis on the other hand operations are applicable only, if they are options of the current client page. Hence Form-Oriented Analysis offers a specification place for the availability of operations. One main additional specification concept in Form-Oriented Analysis is therefore the support of conditional accessibility of operations based on a finite state machine. But beyond the finite automaton as such, Form-Oriented Analysis offers support for specifying constraints in the form of dialogue constraints. Form-Oriented Analysis also offers support for conditional system response. Such conditional response is a first class concept in Form-Oriented Analysis since in the form chart server actions can branch to different client pages. In Z such conditional system response must be coded into the single response type. Hence the return type in Z must be an EITHER-OR type, built from the different response types.

3.4 User Interface Modeling

User interface modeling has become a recognized area of research with the advent of graphical user interfaces. A major milestone in the research on user interface modeling is the Seeheim Model [58], which defines a reference architecture for User Interface Management Systems (UIMS). This reference model has served as the starting point for intense research still underway [52]. In recent times user interface modeling in the context of UML has become an area of interest.

3.4.1 Classical Approaches for User Interface Modeling

State diagrams has been used for a long time in user interface specification [53] [54], partly with the objective of user interface generation [52]. All of these approaches target user interface specification only at a fine-grained level, in our terminology concerning page interaction. Another early approach [47] targeted the modeling of push-based, form-based systems like the already disussed single-user desktop databases.

3.4.2 User Interface Modeling Approaches with UML

Within the UML Community the Discussion about dealing with the user interface is still underway [108]. In [107] a visual language for presenting user interfaces is proposed. The new artifacts are basically visualizations of page components. The method is tightly coupled with the use case driven approach. An example diagram from the article showing a search book functionality is shown in Figure 3.3. In our view, the diagrams do not reach the intuitive clarity of our proposed artifacts.

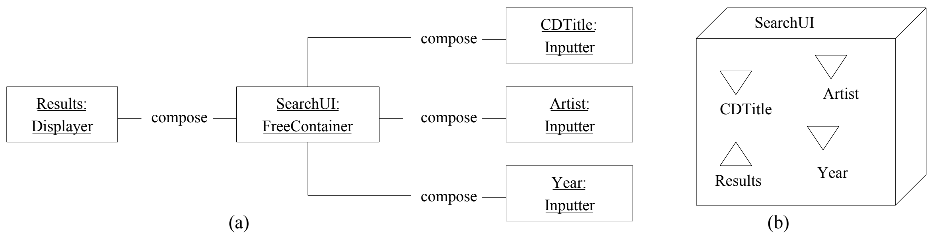

Figure 3.3: The right side (b) shows a CD search function specified in UMLi, on the left side an equivalent object diagram (a) is shown. This example follows closely the original example in [107]

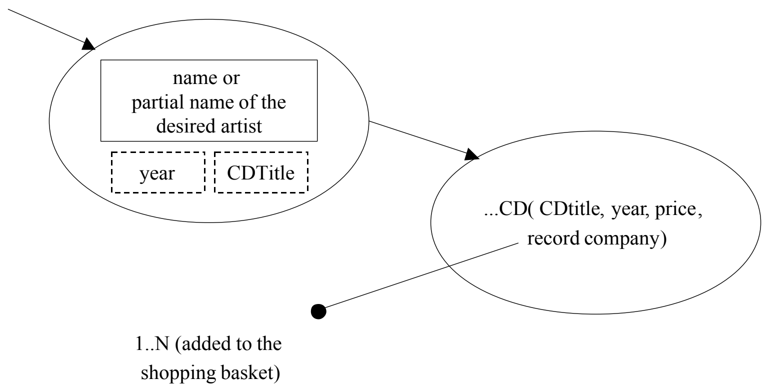

Figure 3.4: A user interface diagram (UID) according to Schwabe et al. [76]. The first step of the dialogue is roughly equivalent to the UMLi example in Figure 3.3. The second step is the selection of a CD for purchase. This UID follows closely the original example from [76].

Schwabe et al. presented a diagrammatic tool for representing web interaction [65], [76]. The diagrams are called user interaction diagrams (UID). They resemble page transition diagrams without server actions. An original diagram is shown in Figure 3.4. Very restricted and very specific annotations are placed on the transitions concerning required selections by the user.

A stereotype framework specifically for web applications is presented in [72] by Conallen. This approach allows to model the design level concepts appearing during web site development with a typical web application framework. For this purpose the Conallen approach uses a set of stereotypes. The approach targets rather design than analysis.

3.4.3 Approaches to the Conceptual Modeling of User Interfaces

The conceptual modeling of hypertext is a separate domain, which has been intensively studied since the beginning of interest in hypertext [138]. A reference model for hypertext based on formal specification with Z is [136]. A complete methodology for hypermedia design called RMM is presented in [135]. The aim to reduce the necessary navigation primitives is adressed in WebML [139], which we discuss in the next section. The conceptual modeling of navigation was also addressed in the ViewNet approach [132]. The named approaches are modeling content in logical collections of information elements. As such these approaches are related to modeling of information architectures as they are represented by content management systems.

All the named hypertext modeling approaches do not use bipartite transition descriptions. The system response in these approaches is unconditional. These approaches have therefore also no direct connection to constraint writing.

A special topic in the domain of interface modeling not considered in this thesis is the dealing with dynamically changing environments in which user interfaces are used. A broad discussion to this problem of human computer interaction in heterogeneous and dynamic environments can be found in [122].

3.4.4 Modeling Web sites with WebML

WebML is a visual language for conceptual modeling of complex web sites [139], in which all concepts are defined visually as well as in XML. WebML offers icons for page elements for composing web sites, e.g. catalogue pages and single item views. The model of the web sites involves a number of orthogonal divisions, called structural model, composition model, navigation model, presentation model and personalization model. The structural model is in principle a semantic data model described in XML. The composition model is a desription of page content based on the data model. The navigation model describes hyperlinks between the pages, with a distinction between so called contextual links, which are induced by the structure of the data model, and non-contextual links, which are freely defined. The WebML approach can therefore be seen as an advanced and customizable successor of model driven interface generators like the JANUS system [79]. The basic idea of such systems is to generate all contextual links from the model. Advanced systems like WebML offer on the other hand the possibility to chose between the contextual links as well as to use freely defineable links. Links in WebML are directed edges, which lead typically from pages to pages and form therefore a non-bipartite navigation model. WebML also offers a mechanism for accessing so called generic external operations by model elements called operation units [140]. Operation units have conditional output similar to server actions in Form-Oriented Analysis, however operation units perform asynchronous computations. They are activated by a designated link and can afterwards perform operations of arbitrary length. They can then leave content in other WebML-modeled units.