Integration of Statecharts

Heinz Frank, Johann Eder
Universität Klagenfurt, Institut für Informatik-Systeme
Universitätsstraße 65-67, A-9020 Klagenfurt
E-mail: {heinz,eder}@ifi.uni-klu.ac.at

Abstract

View integration is an effective technique for developing large conceptual database models. The universe of discourse is described from the viewpoint of different user groups or parts of the system resulting in a set of external models. In a second step these models have to be integrated into a common conceptual database schema.

In this work we present a new methodology for integrating views based upon an object oriented data model, where we concentrate on the integration of the behaviour of objects, which is not supported by existing view integration methods.

1. Introduction

Conceptual modeling of a universe of discourse using an object oriented data model has two dimensions: the structure of objects and their relationships are represented in a static model (or object model), and the behaviour of objects is documented in a dynamic model ([4], [20], [7]).

Statecharts, introduced by David Harel ([13],[14], [15]), are a popular method for designing the behaviour of objects. This concept is used in various design methodologies e. g. OMT ([20]), OOD ([4]) or UML ([1]).

While the techniques for structural modeling have a long tradition and are already quite elaborated, conceptual modeling techniques for the dynamics of a mini-world are not supported as well. In this paper we present a technique for integrating statecharts which is part of a design methodology supporting the view integration approach for all aspects of object oriented data models. For this method we assume that models have been developed from different perspectives. Each of these views consists of one structural (or static) model and several behavioral (or dynamic) models in form of statecharts of the structural model.

In [8] we made a comparative analysis of various view integration methodologies, two for extended E-R models ([2] and [18]) and two for object oriented models ([12] and [11]). We found that these integration methodologies support the integration of the static models well. However, none of them considers the integration of the dynamic models. [12] state that behaviour describes application semantics and, therefore, should not be part of a conceptual database schema. [11] divide behaviour into *operations* and *path methods*. The latter are used to access attributes via relationships of types. In their opinion operations should not be part of the conceptual database schema too.

We do not share these opinions. First, we believe that path methods are a kind of redundancy, which should be part of the conceptual database schema for special reasons, for instance, for comfortable access. Second, in our opinion important common behaviour of objects must be part of the conceptual database schema and not implemented in applications.

Our integration methodology consists of two major phases: the *integration of the static models* and the *integration of the dynamic models*.

The integration of the static models deals with the structural parts (types, attributes and their relationships to other types). The aim of this phase is to identify and solve conflicts (naming conflicts or structural conflicts) among the types of the various views. The result of this integration phase is the common conceptual static model of the universe of discourse. Several integration strategies, mainly for the entity relationship model ([6]), were published in the past, for instance [2], [12], [11], [18] or [22]. Further comparative analysis of view integration methodologies were made in [3] and [21]. We did not develop another strategy for the integration of the the structural part of types but use already published methodologies, mainly [18] and [19].

Through the integration of the different static models of the views we derive an integrated conceptual static model. Afterwards the integration of the dynamic models of the views takes place. For each integrated type the corresponding dynamic models (or statecharts) of the views have to be integrated.

In an earlier paper ([10]) we presented a formalization of statecharts together with a set of transformation which are proven to keep the semantics of the models and which are complete in the sense that they suffice to derive any equivalent model from a given one. This forms the basis of the integration of statecharts, which consists of the following steps:

First, we formalize the statecharts by formally defining the range of all states in all statecharts with constraints or conditions on the given static type. The range of a statechart is thus defined as a subspace of the object space spanned by the definition of the type (i. e. all possible object instances of a type). Then we analyze the relationship of the statecharts on the basis of their ranges and their marginal states (begin and end states). Relationship classes are disjoint, consecutive, alternative, parallel, and mixed.

The second phase is called the *integration-in-the-large*, where we develop an integration plan with the goal of minimizing the integration effort. The integration plan consists of integration operators for the different relationships of statecharts. For mixed relationships, a further analysis is necessary where all states and events of two models have to be analyzed to integrate the models. For the other relationships, integration operators only consider the marginal states. A crucial part of the development of an integration plan is to determine the relationships between an integrated statechart and all other statecharts without actually performing the integration.

Third, in the *integration-in-the-small* phase the integration is performed by executing the integration plan.

In the next sections we concentrate on the integration of statecharts. In section 2 we present an overview of the used data model. In section 3 an example from the domain of a library is presented to demonstrate our methodology. The integration process for statecharts is shown in section 4 and illustrated by the library example. However, in this paper we give an overview of our integration approach in a more descriptive way without presenting all the formal details and proofs. Interested readers are refered to [9] and [10].

2. The data model

Our integration methodology is based upon an object oriented data model such as ODMG ([5]) or $\mathcal{TQL}++$ ([16], [17]). A conceptual database schema (and therefore the views too) consists of a set of types, describing the structural properties of objects (the static model or object model). Each type may have a behaviour which is described with a dynamic model (or statecharts according to [13],[14] and [15]).

For the integration methodology we use $\mathcal{TQL}++$ for designing the static model (types with their attributes and relationships to other types) of the views and statecharts for describing the behaviour of objects. However, for the integration we made some extensions to statecharts to formalize the semantics. Here we briefly summarize the formalism. Details can be found in [10].

A statechart of a type consists primarily of *states* and *events*. Events trigger *transitions* which causes a state change of an object. An object which is in a state S_1 can react to a transition t triggered by an event e and changes its state to S_2. We call S_1 the *source state* of the transition t and S_2 the *target state* of the transition.

To reduce the complexity and to make dynamic models more readable, states can be structured to state hierarchies: *state generalizations* to express alternatives, and *state aggregations* to express parallelism.

States are provided with conditions which an object must fulfill to be in the state. These conditions, we call them the *range* of a state, are based upon the attributes and relationships of the type. The range of a state S_1 can be regarded as a logical expression resulting in $true$ if a given object is in the state S_1, in $false$ otherwise. Statecharts have a range too, which is defined as the disjunction of the ranges of all its states.

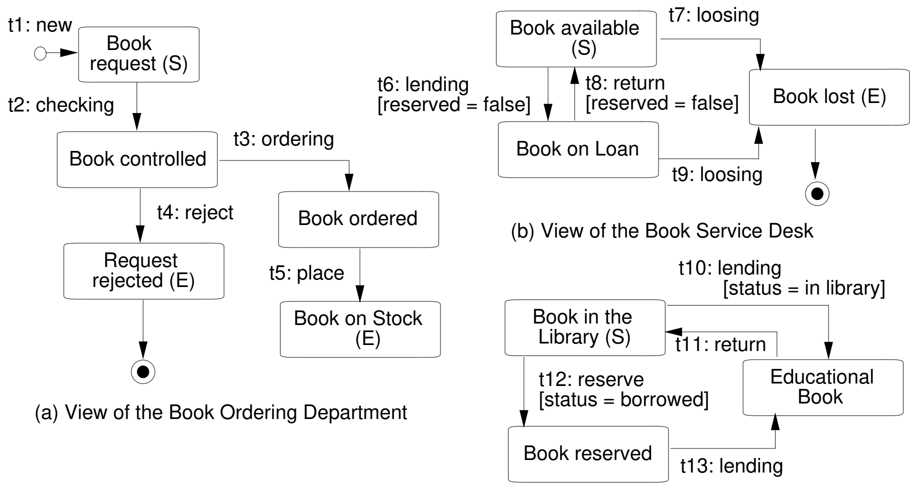
States may be marked as *start* and *end* states ("marginal" states). For instance, the initial and final states are start and end states. However, the designer is allowed to mark any further state too.

Transitions have pre- and postconditions, which again are logical expressions. The *preconditions* of a transition are the conditions an object must comply to enable the transition causing a state change. The precondition of a transition is defined as the conjunction of the range of its source states and its guard.

The *postcondition* of a transition are those conditions an object must fulfill after the application of the transition. Therefore, the postcondition of a transition must imply the ranges of its target states.

As specification language for these conditions (the range of states, pre- and postconditions and guards of transitions) we use $\mathcal{TQL}++$. For addressing these additional characteristics, we use meta methods of statecharts, states or transitions. With $S_1.Range()$ the conditions (the range) of the state S_1 is meant. We write $t.PreC()$ to get the preconditions of a transition t. As these conditions are logical expressions we may combine them by disjunction, conjunction or negation. For instance the preconditions of a transition t is computed by the conjunction of the range of its source state and its guard, that is $t.Source\_State.Range() \wedge t.Guard$.

Ranges of states as well as pre- and postconditions of transitions are additional characteristics of statecharts, which are used to define the semantics of statecharts. In [10] we defined the semantics of statecharts and a complete set of schema transformations to transform a statechart into any other equivalent statechart. As an example we have transformations to decompose and to construct state hierarchies,



(a) View of the Book Ordering Department

(b) View of the Book Service Desk

(c) View of the Educational Book Department

Figure 1. The statecharts of the type Book.

to split and to combine states, and to shift transitions within state hierarchies.

In the next sections we discuss the process of integrating the statecharts of an integrated type. We assume that the integration of the static models has already been finished. Therefore, we have to deal with one integrated type and several statecharts describing the behaviour of objects of the type to integrate. We start with a short example followed by a discussion of each integration step.

3. The example

Let us introduce a short example from the domain of a library showing the behaviour of books from the viewpoint of three different departments. Assume that the integration of the static models results in an integrated type Book having the following $\mathcal{TQL}++$ syntax:

Book = [
  Isbn: str,
  Title: str,
  Authors: {Author},
  reserved: bool,
  status: (requested, ordered, lost, in library, borrowed,
    in textbook collection)]

The behaviour of a book from the viewpoint of the Book Ordering Department is shown in figure 1(a). The department treats incoming book requests, checks for doublets, and orders books. Delivered books are registered and placed into the library.

book request	this.Isbn is UNKNOWN $\wedge$ this.status = requested
book controlled	this.Isbn is KNOWN $\wedge$ this.status = requested
request rejected	this.Isbn IN book.Isbn $\wedge$ this.status = rejected
book ordered	this.Isbn # book.Isbn $\wedge$ this.status = ordered
book on stock	this.status = in library $\wedge$ this.reserved = false

Table 1. State specification of the Book Ordering Department.

Books can be borrowed at the Book Service Desk if they are available. Books may get lost. The dynamic model of a book from the viewpoint of the Book Service Desk is shown in figure 1(b).

At least the Educational Book Department is responsible for the administration of educational books, which are necessary for a lecture during a certain period. Such books may not be borrowed by anyone until the end of the lecture. If the book in question is out of stock it can be reserved by the department. Reserved books may not be borrowed by anyone except the educational book department. The behaviour of a book from the viewpoint of this department is shown in figure 1(c).

For the integration of dynamic models we demand the specification of states which are shown in tables 1, 2 and 3. The postconditions of the transitions are either equivalent with the range of their target states or shown in table 4. Start and end states are marked with (S) and (E) in the views.

book available	this.status = in library $\wedge$ this.reserved = false
book on loan	this.status = borrowed
book lost	this.status = lost

Table 2. State specification of the Book Service Desk.

book in the library	(this.status = in library $\vee$ this.status = borrowed) $\wedge$ this.reserved = false
book reserved	this.status = borrowed $\wedge$ this.reserved = true
educational book	this.status = in textbook collection

Table 3. State specification of the Educational Book Department.

4. The process of integrating statecharts

The integration of the static models results in an integrated conceptual static model, a common agreement about types, their internal structure (attributes) and their relationships between them. Afterwards the integration of the statecharts takes place. The input parameter are an integrated type and its various statecharts from the different views. The aim of this integration phase is to obtain a common behaviour described within one statechart of this type. To integrate the statecharts of a type we propose two phases:

- *Integration-in-the-large:* In this integration phase an overall structure of the integrated statechart is developed. Based upon relationships between the statecharts, the *relationship graph* is constructed. On the basis of this graph the *integration plan* is computed. The integration plan consists of a tree of *integration operators*. An integration operator has two statecharts as input and the integrated statechart as output.

- *Integration-in-the-small:* In this phase the integration takes place. According to the integration plan the integration operators are carried out step by step. The integration operators can be applied automatically. Only the integration operator for mixed related statecharts (see later) requires a more detailed analysis of the states and events of both statecharts and probably designer decisions.

In the next sections we discuss the steps of both integration phases. However, we omit formal details and proofs. Interested readers are refered to [9].

t6: lending	this.status = borrowed $\wedge$ this.reserved = false
t11: return	this.status = in library $\wedge$ this.reserved = false

Table 4. Postconditions of the transitions.

4.1. Integration-in-the-large

The aim of the integration-in-the-large is mainly to prepare for the integration which actually is done in the integration-in-the-small phase. Starting point is an integrated type and its statecharts, the result is the integration plan. To develop the integration plan we have

- to determine the relationships between the involved statecharts
- to represent the relationships in the relationship graph
- to compute the integration plan

4.1.1. Relationships between statecharts

The ranges of states and the ranges of statecharts are the basis for relationships between statecharts. We have defined five classes of possible relationships between statecharts. For the formal definitions we refer to [9], at this point we concentrate on the idea behind these relationships:

- *parallel statecharts*: an object of the integrated type has to pass both statecharts in parallel. The ranges of the statecharts have to be equivalent and orthogonal. Orthogonal means for any two state Z_1 of the statechart M_1 and Z_2 of the statechart M_2 the ranges of Z_1 and Z_2 must not be disjoint.

- *disjoint statecharts*: an object of the integrated type has to pass either the first or the second statecharts. The ranges of the dynamic models must be disjoint.

- *consecutive statecharts*: an object of the integrated type has to pass first the one and second the other statechart. Beyond some start and end states the ranges of both statecharts must be disjoint.

- *alternative statecharts*: an object of the integrated type has to pass the statecharts alternatively. Beyond some start and end states the ranges of both dynamic models have to be disjoint.

- *mixed statecharts*: an object of the integrated type has to pass a “mixture” of both statecharts.

Let us discuss the consecutive relationship more detailed. Consider the example in figure 2, where you can see

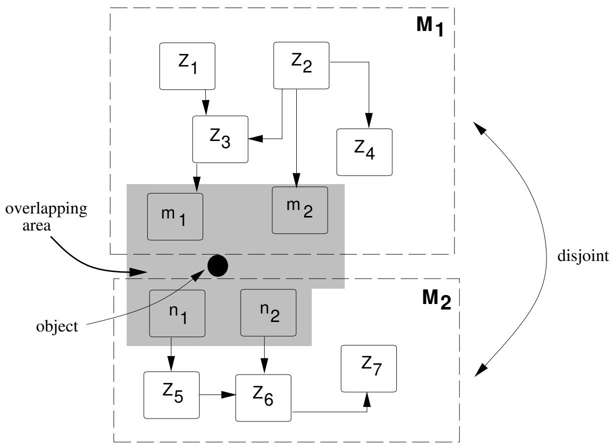


Figure 2. An example for a consecutive relationship.

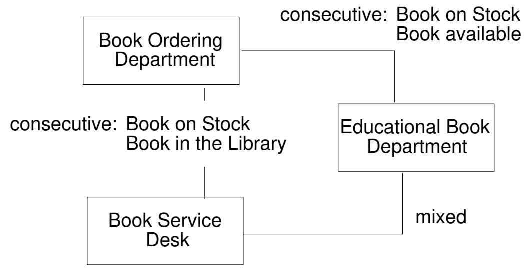


Figure 3. The relationship graph of the library example.

the statecharts M_1 and M_2. Except an overlapping area between "marginal" states (end states of M_1 and start states of M_2), the ranges of the statecharts have to be disjoint. The ranges of the "marginal" states must either be equivalent. or the range of one state has to imply the range of the other state. Figuratively spoken, an object travels first through M_1, reaching for example the end state m_1 of M_1. As the range of m_1 is equivalent (or implies) the range of the start state n_1 of M_2 the object gets into the statechart M_2.

For each relationship class between statecharts we have developed an *integration operator*. An integration operator gets two statecharts to integrate and results in the integrated statechart. The integration operator $IPar$ integrates two parallel dynamic models into a state aggregation. $IDis$, which integrates disjoint dynamic models, is very simple as there is nothing to integrate. It simply computes the union of both statecharts. $ICons$ integrates consecutive related statecharts by merging the "marginal" states of the overlapping area to single states. $IAlt$ does the same with alternative related statechart. However, $IMix$, which integrates mixed related statecharts, is much more complicated. A detailed analysis of the states and events of both statecharts is necessary and will be discussed later on.

In our example of figure 1, we determine the following relationships between the statecharts. The statecharts of the Book Ordering Department and Book Service Desk are *consecutive*. The end state book on stock from the Book Ordering Department and the start state book available from the Book Service Desk are equivalent, the remaining states are disjoint (compare the state specifications in the tables 1 and 2). The statecharts of the Book Ordering Department and Educational Book Department are *consecutive* too, because the range of the state book on stock implies the range of the state book in the library (compare tables 1 and 3). The statecharts of the Book Service Desk and the Educational Book Department are *mixed* related (compare tables 2 and 3).

4.1.2. The relationship graph

The relationships between the statecharts of an integrated type are represented with the *relationship graph*. The nodes of this graph are the statecharts, the edges represent the relationship between them. In the case of an alternative or consecutive relationship between statecharts, the edges are annotated with the corresponding "marginal" states of the statecharts. The relationship graph of our library example is shown in figure 3.

4.1.3. The integration plan

The aim of an integration plan is to determine an integration sequence with minimal integration effort without actually integrating the statecharts. The integration plan states which statecharts have to be integrated with which particular integration operator. Therefore, we have to consider the consequences of each integration step. Suppose we would like to integrate two statecharts M_1 and M_2 with an integration operator. This results in a statechart M_I, the relationship graph must be changed. M_1 and M_2 have to be replaced with M_I. But which are the relationships of M_I to other statecharts of the relationship graph?

According to the definition of the relationships between statecharts, the answer to this question depends on the range of the integrated statecharts and its start and end states. For each integration operator, except $IMix$, which integrates mixed related statecharts, we know the range as well as the start and end states of the integrated dynamic model.

Consider a small example. Suppose we integrate M_1 and M_2, which are disjoint, to M_I. Suppose further that M_3 is consecutive to M_1 but disjoint to M_2. Obviously, M_I and M_3 will be consecutive too. Another example, suppose we integrate M_1 and M_2 with a mixed relationship. This cannot be done automatically. We may conclude only the range of the integrated statechart M_I, but not the "marginal" states of M_I. Therefore, if there is a relationship between another statechart M_3 and M_1 (except a disjoint relationship) M_3 and M_I will be mixed too.

It is possible to compute the consequences of the integration operators for the relationship graph. Interested readers are refered to [9], where changes of the relationship graph due to the application of an integration operator are proven.

The integration plan determines an integration sequence with minimal integration effort, which depends on the automation possibilities of the integration operators. Integrating mixed related statecharts with the integration operator $IMix$ requires an additional analysis and cannot be done fully automatically. Therefore, the integration effort for the application of $IMix$ is high. Furthermore, the more states the statecharts have, the higher is the integration effort. In contrast to $IMix$ the other integration operators can be performed automatically, their integration efforts are low. This leads to a sketch of a rule based algorithm to compute the integration plan:

1. Integrate mixed related statecharts as soon as possible, as long as the integration does not destroy "cheap" relationships.

2. Integrate statecharts when the integration does not destroy "cheap" relationships.

3. Integrate statecharts whose integration preserves as much "cheap" relationships as possible.

Back to our example from figure 1. If we would integrate the statecharts of the Educational Book Department and Book Service Desk, which are mixed related, the integrated statechart would have a mixed relationship to the statechart of the Book Ordering Department. We would need another usage of the "expensive" integration operator $IMix$. However, if we first integrate the statecharts of the Book Ordering Department and the Book Service Desk with the integration operator $ICons$ to M_1 and afterwards M_1 model with the statechart of Educational Book Department to M_I we would need $IMix$ only once. Our integration plan looks like as follows:

ICons (Book Ordering Department, Book Service Desk, M_1)
IMix (M_1, Educational Book Department, M_I)

4.2. Integration-in-the-small

We start the integration-in-the-small after the development of the integration plan. The integration plan is executed step by step, the statecharts are integrated with the corresponding integration operators.

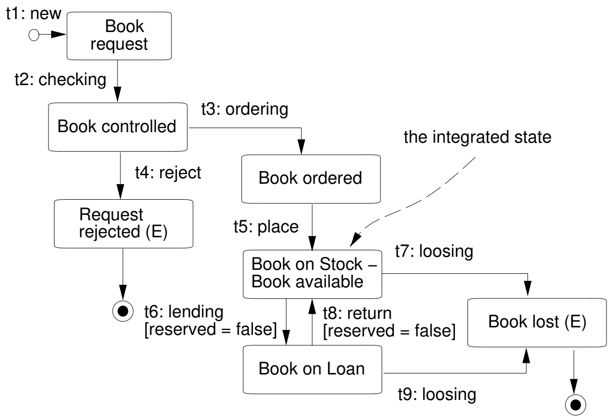


Figure 4. The integrated statechart M_1.

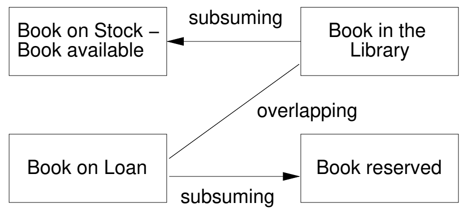


Figure 5. The state relationship graph.

According to the integration plan of our library example, we have to integrate the statecharts of the Book Ordering Department and the Book Service Desk using the integration operator $ICons$. The operator $ICons$ simply combines the annotated states on the edge of the relationship graph, that are Book on Stock and Book available, to one state. The integrated statechart M_1 is shown in figure 4.

In the next step we have to integrate the statechart M_1 and the statechart of the Educational Book Department. However, integrating statecharts whose relationship is mixed is much more complicated, a further detailed analysis is necessary. Furthermore, this integration step cannot be done automatically. We are only able to support the designer by computing integration recommendations.

Just as statecharts the states of different statecharts have relationships too. For instance, the ranges of states may be equivalent or disjoint, or the range of a state may imply the range of another state (we say a state *subsumes* another state). Ranges of states may be *overlapping*. Once more a graph, the *state relationship graph*, is used to represent the relationships between states. The nodes of this graph are the states of the statechart. An edge between a state of one model and a state of another model exists if there is a relationship other than disjoint between them.

A part of the state relationship graph which is computed by the usage of $IMix$ in our library example is shown in figure 5. The ranges of the other states of the statecharts to integrate are disjoint.

As we do not consider disjoint relationships in the state relationship graph, the graph must not be fully connected. The states of a connected subgraph have to be integrated. For the integration we use the schema transformations defined in [10].

1. State hierarchies are decomposed using appropriate schema transformation.
2. The states of a fully connected subgraph of the state relationship graph are transformed by schema transformations so that they are disjoint.
3. The transformed states are combined according to several heuristics to single states or state hierarchies (schema restructuring). An appropriate heuristic for the combination of states are, for instance, the transitions. For example, states which are source states of transitions triggered by the same event may be combined to a single state.

The first and the second step can be done automatically by using appropriate schema transformations. However, for combining states there are several integration possibilities. According to some heuristics a set of integration recommendations could be computed, but the designer has to choose one.

The transformation of states of a subgraph of the relationship graph depends on the involved states and the relationships between them. In our example we have several states of both statecharts and the relationship classes *subsuming* and *overlapping*. In such a case we construct disjoint states according to the following algorithm:

1. For each state Z of the first (second) statechart, which is not subsumed by a state of the second (first) statechart, we construct a state $\overline{Z}$. The range of this state results in the conjunction of the range of the state Z and the negation of the disjunction of the ranges of all states of the second (first) statechart. All transitions, having Z as source or target state, are copied and Z is replaced by $\overline{Z}$ in the source or target states of the copied transition. The postcondition of transitions having $\overline{Z}$ as target state is replaced with the conjunction of the original postcondition and the range of $\overline{Z}$.
2. For each state Z_1 of the first statechart and state Z_2 of the second statechart, which are not disjoint related, we construct a state $\hat{Z}$ with the range $Z_1.Range()$ $\wedge$ $Z_2.Range()$. All transitions having Z_1 or Z_2 as source or target states are copied. Z_1 and Z_2 are replaced by $\hat{Z}$ in the source or target states of the copied transition. The postcondition of transitions having $\hat{Z}$ as target state is replaced with the conjunction of the original postcondition and the range of $\hat{Z}$.
3. All states of the subgraph and all transitions having states of the subgraph as source or target state are deleted.

For the construction of disjoint states we use schema transformations, which preserve the equivalence of the statecharts according to [10]. It is possible that some of the constructed disjoint states have ranges resulting in false. Such states and all transitions having such states as source or target states are deleted. Furthermore, the preconditions or postconditions of some copied transitions may result in false and can be deleted without changing the semantics of the statecharts as shown in [10].

To conclude our library example, we transform the involved states of the state relationship graph into disjoint states. According to the above approach, we have to construct five states, two states because of the first rule and three state because of the second rule.

The first state to construct (according to the first rule) has the condition Book in the Library.Range() ∧ ¬ (Book on Stock - Book available.Range() ∨ Book on Loan.Range()). However, this condition results in false (compare tables 2 and 3). The second state to construct has the condition Book on Loan.Range() ∧ ¬ (Book in the Library.Range() ∨ Book reserved.Range()), which is false too (compare tables 2 and 3).

The next three states must be constructed according to the second rule, these are the states:

- Book in the Library - not reserved with the condition Book on Stock - Book available.Range() ∧ Book in the Library.Range().
- Book borrowed - reserved with the condition Book reserved.Range() ∧ Book on Loan.Range().
- Book borrowed - not reserved with the condition Book in the Library.Range() ∧ Book on Loan.Range().

The specification of these states are shown in table 5. All transitions of the involved original states have to be copied, and their source and target states as well as their postconditions are adapted. Lets take a closer look on the first constructed state Book in the Library - not reserved, which is shown in figure 6.

The state is constructed as the “intersection” of the states Book on Stock - Book available and Book in the Library. The transitions of both states are copied and become transitions of the constructed state. However, the precondition of the transition t12(1) triggered by the event reserve, which is computed as the conjunction of the guard of the transition

book in the library - not reserved	this.status = in library $\wedge$ reserved = false
book borrowed - not reserved	this.status = borrowed $\wedge$ this.reserved = false
book borrowed - reserved	this.status = borrowed $\wedge$ this.reserved = true

Table 5. State specifications after the transformation.

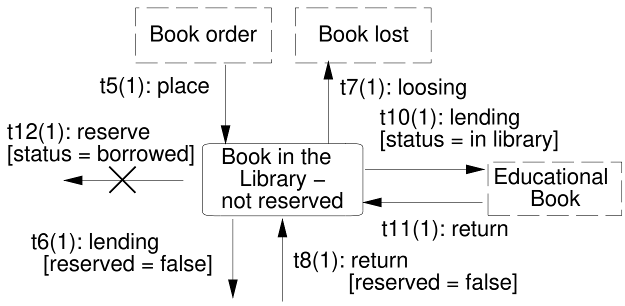


Figure 6. The state Book in the Library - not reserved.

and the range of its new source state, results in false (compare table 5). Therefore, this transition can be deleted.

All other states are constructed similarly. The constructed states are shown in figure 7 (not involved states are dotted). The original states of the subgraph of the state relationship graph with their transitions are deleted.

In the second step we may consider combining some of the states or creating state hierarchies. For instance, we may build a state generalization based upon the states Book borrowed - not reserved and Book borrowed - reserved as they have some transitions triggered by the same event in common, e. g. loosing.

As all states of the state relationship graph of our example are integrated, we finish the integration with the integration operator $IMix$ and return to the integration plan. The integrated statechart can be obtained just by merging the statecharts from the figures 4 and 7.

5. Conclusion

In this work we have presented a methodology for integrating object oriented views with the main contribution to the integration of behaviour models. The behaviour of objects is represented with statecharts according to [14].

Our methodology of integrating behaviour models consists of the phases: integration-in-the-large and integration-in-the-small. The aim of the first integration phase is to

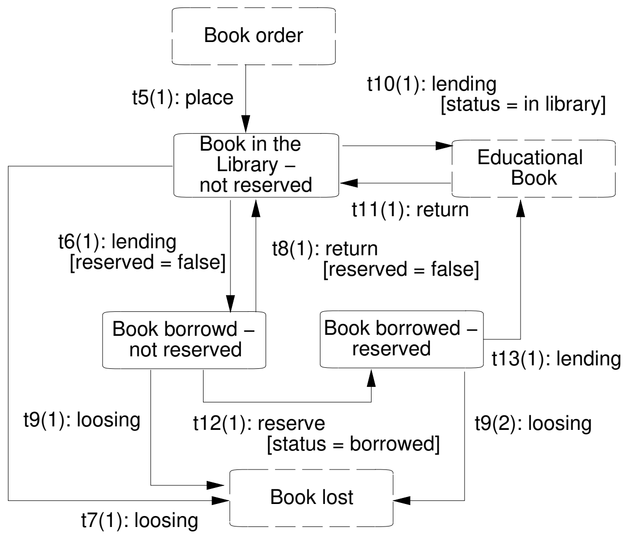


Figure 7. The integrated states.

determine an overall structure of the integrated dynamic model. Based upon relationships between statecharts, integration operators, which integrate the statecharts, are defined. The result of the integration-in-the-large is the integration plan stating an integration order with minimal effort.

Within the integration-in-the-small phase the integration of the statecharts takes place. Most of the defined integration operators can be performed automatically. However, for mixed related statecharts a further detailed analysis of states and events is necessary.

It was our aim to relieve the designer by automating the integration as much as possible. In some situations the integration of statecharts can be done without the aid of a designer. Even the integration of mixed related statecharts could be computed by using additional heuristics for the combination of states. However, the designer is invited to make changes in order to improve quality.

We see the main advantages of our approach in the formal treatment of the integration process, which allows a highly automatic integration, while giving the designer the possibility to make decisions and carry out their consequences in the model automatically.

References

[1] Rational Software et.al. Unified modeling language (uml) version 1.1. http://www.rational.com/uml, Sept. 1997.

[2] C. Batini and M. Lenzerini. A methodology for data schema integration in the entity relationship model. *IEEE Transactions on Software Engineering*, 10(6):650–664, Nov. 1984.

[3] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323 – 364, Dec. 1986.

[4] G. Booch. *Object-Oriented Design with Applications*. Benjamin Cummings, 1991.

[5] R. Cattell, D. Barry, D. Bartels, M. Berler, J. Eastman, S. Gamerman, D. Jordan, A. Springer, H. Stickland, and D. Wade. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann Publishers, Inc, 1997.

[6] P. Chen. The entity-relationship model - toward a unified view of data. *ACM Transaction on Database Systems*, pages 9–36, Mar. 1976.

[7] D. Coleman, P. Arnold, S. Bodoff, C.Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes. *Object-Oriented Development: The Fusion Method*. Prentice Hall Object-Oriented Series. Prentice-Hall, Inc, 1994.

[8] J. Eder and H. Frank. Schema integration for object oriented database systems. In M. T. et al., editor, *Software Systems in Engineering*. ASME, 1994. Procceedings of the ETCE, New Orleans.

[9] H. Frank. *View Integration für objektorientierte Datenbanken*. PhD thesis, Institut für Informatik, Universität Klagenfurt, 1996.

[10] H. Frank and J. Eder. A meta-model for dynamic models. Technical report, Institut für Informatik, Universität Klagenfurt, Mar. 1997. http://www.ifi.uni-klu.ac.at/cgi-bin/show_an_abst?1997-05-FrEd.

[11] J. Geller, Y. Perl, E. Neuhold, and A. Sheth. Structural schema integration with full and partial correspondence using the dual model. *Information Systems*, 17(6):443–464, 1992.

[12] W. Gotthard, P. C. Lockemann, and A. Neufeld. System guided view integration for object-oriented databases. *IEEE Transaction on Knowledge and Data Engineering*, 4(1):1–22, Jan. 1992.

[13] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231 – 274, 1987.

[14] D. Harel. On visual formalisms. *Communications of the ACM*, 31(5):514 – 530, May 1988.

[15] D. Harel and A. Naamad. The statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293 – 333, Oct. 1996.

[16] H. Lam and M. Missikoff. On semantic verification of object-oriented database schemas. In *Proceedings of Int. Workshop on New Generation Information Technology and Systems - NGITS*, pages 22 – 29, June 1993.

[17] M. Missikoff and M. Toaiti. Mosaico: an environment for specification and rapid prototyping of object-oriented database applications. EDBT Summer School on Object-Oriented Database Applications, Sept. 1993.

[18] S. B. Navathe, R. Elmasri, and J. Larson. Integrating user views in database design. *IEEE Computers*, pages 185–197, Jan. 1986.

[19] S. B. Navathe and G. Pernul. *Advances in Computers*, volume 35, chapter Conceptual and Logical Design of Relational Databases, pages 1 – 80. Academic Press, 1992.

[20] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall International, Inc, 1991.

[21] M. Schrefl. A comparative analysis of view integration methodologies. In R. T. R Wagner and H. Mayr, editors, *Informationsbedarfsermittlung und -analyse für den Entwurf von Informationssystemen*, pages 119–136, 1987. Fachtagung EMISA.

[22] A. P. Sheth. Issues in schema integration: Perspective of an industrial researcher. In *ARO-Workshop on Heterogeneous Databases*, Sept. 1991.