Integration of Behaviour Models

Heinz Frank and Johann Eder

Universität Klagenfurt, Institut für Informatik
Universitätsstraße 65 - 67, A-9020 Klagenfurt, Austria
e-mail: {heinz, eder}@ifi.uni-klu.ac.at

Abstract

View integration is the most effective technique for developing a conceptual database model. 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 already not supported by existing view integration methods.

1 Introduction

Conceptual modeling of a universe of discourse has two dimensions: the structure of objects and their relationships are represented in a static model (or object model) and the behaviour of the objects is documented in a dynamic model. 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 view integration for dynamic models 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 view models consists of a structural (or static) model and a set of behavioral (or dynamic) models in form of state charts (for each type one).

In [EF94] we made a comparative analysis of various view integration methodologies, two for extended E-R models ([BL84] and [NEL86]) and two for object oriented models ([GLN92] and [GPNS92]). 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. [GLN92] state that behaviour describes application semantics and, therefore, should not be part of a conceptual database schema. [GPNS92] divide behaviour into *operations* and *path methods*. The later are used to access attributes via relationships of types. Also in their opinion operations should not be part of the conceptual database schema.

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 at least, for instance for comfortable access (just like views). 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 ([Che76]) were published in the past, for instance [BL84, GLN92, GPNS92, NEL86] or [She91]. Further comparative analysis of view integration methodologies were made in [BLN86] and [Sch87]. For our methodology we did not develop another strategy for integrating the structural part of types but use already published methodologies, mainly [NEL86] and [NP92].

Through the integration of the different static models of the views we get the integrated conceptual static model. Afterwards the integration of the dynamic models of the views takes place. For each integrated type its corresponding dynamic models of the views have to be integrated.

In an earlier paper ([FE97]) we presented a meta model for state charts 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 dynamic models, which consists of the following steps:

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

Second is 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 models. 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 derive the relationship of the model resulting from the integration of two models with all the other models without actually performing the integration.

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

In the next sections we concentrate on the integration of dynamic models. 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 dynamic models is shown in section 4 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 [Fra96] and [FE97].

2 The data model

Our integration methodology is based upon an object oriented data model such as OMT ([RBP$^+$91]), $\mathcal{TQL}++$ ([LM93, MT93]) or ODMG ([CBB$^+$97]). A conceptual database schema (and therefore the views too) consists of a set of types, describing the structural properties of objects (the static model or object model in OMT). Each type may have a behaviour which is described with a dynamic model (or state charts according to [Har88, RBP$^+$91, Rum93]).

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 state charts for describing the behaviour of objects. However, for the integration we had to made some extensions to dynamic models to formalize the semantics of state charts. In [FE97] we presented this language in detail. Let us discuss them briefly in the next paragraphs.

A dynamic model of a type primarily consists of states and events. We allow an event to occur several times within a dynamic model and, therefore, distinguish between the event and its event occurrences. An object which is in a state S_1 can react to an event's occurrence e resulting in a transition of the object to a new state S_2. We call S_1 the *source state* of the event occurrence e and S_2 the *target state* of the event occurrence.

To reduce the complexity and to make dynamic models more readable states can be structured to state hierarchies. That are state generalizations, to express alternatives, and state aggregations, to express parallelism according to OMT [RBP$^+$91].

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 an given object is in the state S_1, in $false$ otherwise. Dynamic models have a range too, which is defined as the disjunction of ranges of the states of the dynamic model.

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.

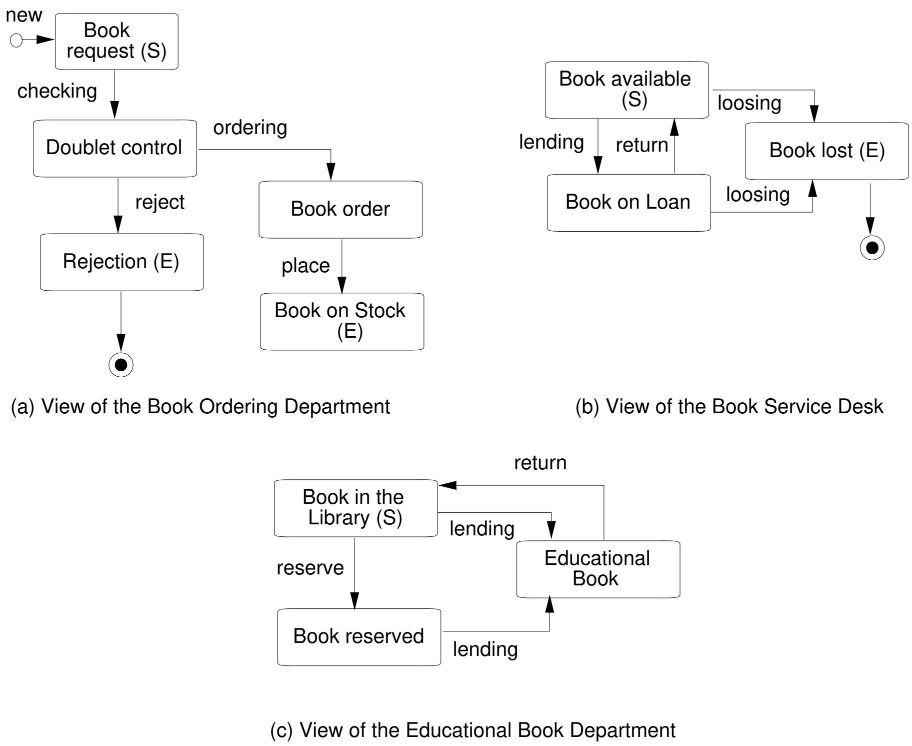
Furthermore event occurrences have pre- and postconditions, which again are logical expressions. The *preconditions* of an event occurrence are the conditions, an object must comply to enable the event occurrence causing a state transition. The precondition of an event occurrence is defined as the conjunction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

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

As specification language for these conditions we use $\mathcal{TQL}++$. For addressing these additional characteristics we use meta methods of dynamic models, states or event occurrences. With $S_1.Range()$ the conditions (the range) of the state S_1 is meant. We write $e.PreC()$ to get the preconditions of an event occurrence e. As these conditions are logical expressions we may combine them by disjunction, conjunction or negation. For instance the preconditions of an event occurrence e is computed by the conjunction of the range of its source state and its guard, that is $e.Source\_State.Range() \wedge e.Guard$.

Ranges of states as well as pre- and postconditions of event occurrences are additional characteristics of dynamic models, which are used to define the semantics of state charts. In [FE97] we defined the semantics of state charts and a complete set of schema transformations to transform a dynamic model into any other equivalent dynamic model. As an example we have transformations to decompose and to construct state hierarchies, to split and to combine states and to shift event occurrences within state hierarchies.

In the next sections we discuss the process of integrating the dynamic models 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 dynamic models to integrate. We start with a short example followed by a discussion of each integration step.



(a) View of the Book Ordering Department

(b) View of the Book Service Desk

(c) View of the Educational Book Department

Figure 1: The three dynamic models of the type Book

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 into 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
doublet control	this.Isbn is KNOWN $\wedge$ this.status = requested
rejection	this.Isbn IN book.Isbn $\wedge$ this.status = rejected
book order	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

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

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 as well as the postconditions of the event occurrences, which are shown in the tables 1, 2 and 3. The postconditions of the event occurrences are equivalent with their target states. Start and end states are marked with (S) and (E) in the views.

4 The process of integrating the behaviour

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

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

- *Integration-in-the-large:* In this integration phase an overall structure of the integrated dynamic model is developed. Based upon relationships between the dynamic models the *relationship graph* is constructed. On the basis of this graph and with the aid of an algorithm the *integration plan* is computed. The integration plan consists of a sequence of *integration operators*. An integration operator gets two dynamic models and integrates them to one dynamic model.

- *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. All integration operators except one can be applied automatically. For this integration operator a more detailed analysis of the states and events of both dynamic models to integrate is necessary.

In the next sections we discuss the steps of both integration phases. However, we omit formal details and proofs. Interested readers are refered to [Fra96].

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 dynamic models, the result is the integration plan. To develop the integration plan we have

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

4.1.1 Relationships between dynamic models

The ranges of states and the ranges of dynamic models are the basis for relationships between dynamic models. We have defined five classes of possible relationships between dynamic models, namely *parallel*, *disjoint*, *alternative*, *consecutive*

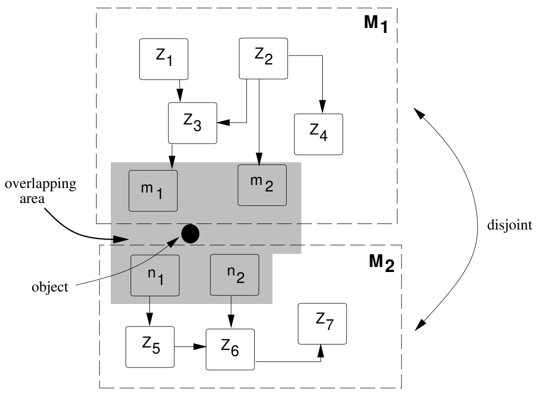


Figure 2: An example for a consecutive relationship

and *mixed*. For the formal definitions we refer to [Fra96], at this point we concentrate on the idea behind these relationships:

- parallel dynamic models: an object of the integrated type has to pass parallel both dynamic models. The ranges of the dynamic models have to be equivalent and orthogonal.

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

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

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

- mixed dynamic models: an object of the integrated type has to pass a "mixture" of both dynamic models.

Let us discuss the consecutive relationship more detailed. Consider the example in figure 2, where you can see the dynamic models 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 dynamic models 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 dynamic model M_2.

For each relationship class between dynamic models we have developed *integration operators*. An integration operator gets two dynamic models to integrate and delivers the integrated dynamic model. 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 builds the union of both dynamic models to the integrated dynamic model. $ICons$ integrates consecutive dynamic models by combining the "marginal" states of the overlapping area to single states. The same does $IAlt$ with alternative related dynamic models. However, $IMix$, which integrates mixed related dynamic models, is much more complicated. A detailed analysis of the states and events of both dynamic models is necessary, which we will discuss later on.

In our example of figure 1 we determine the following relationships between the dynamic models. The dynamic models 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 dynamic models 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 dynamic models 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 all dynamic models of an integrated type are represented with the *relationship graph*. The nodes of this graph are the dynamic models, the edges represent the relationship between the dynamic models. In the case of an alternative or consecutive relationship between dynamic models the edges are annotated with the corresponding "marginal" states of the dynamic models. 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 dynamic models. The integration plan states which dynamic models have to be integrated with which particular integration operator.

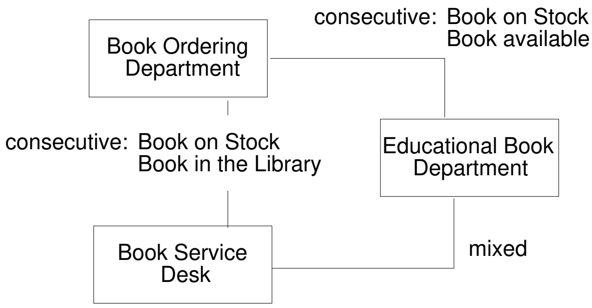


Figure 3: The relationship graph of the library example

Therefore, we have to consider the consequences of each integration step. Suppose we would like to integrate two dynamic models M_1 and M_2 with an integration operator. This results into a dynamic model M_I, the relationship graph must be changed. M_1 and M_2 have to be replaced by the model M_I. But which are the relationships of M_I to other dynamic models of the integrated type?

According to the definition of the relationships between dynamic models the answer to this question depends on the range of the integrated dynamic model and its start and end states. For each integration operator, except $IMix$, which integrates mixed related dynamic models, 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 model M_I, but not the "marginal" states of M_I. Therefore, if there is a relationship between another dynamic model M_3 and M_1 (except a disjoint relationship) M_3 and M_I will be mixed too.

It is possible to derive the consequences to the relationship graph caused by the application of an integration operator for each thinkable combination of relationships without actually recomputing them. Interested readers are refered to [Fra96], where changes of the relationship graph due to an application of an integration operator are proven.

The integration plan determines an integration sequence with minimal integration effort, which depends on the automatisation possibilities of the integration operators. Integrating mixed related dynamic models 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 in general the more states the dynamic models have the higher is the integration effort. In contrast to $IMix$ the other integration operators are automizable, their integration efforts are low. This leads to a sketch of a rule based algorithm to compute the integration plan:

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

2. Integrate other related dynamic models when the integration does not destroy "cheap" relationships.

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

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

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 dynamic models are integrated with the corresponding integration operators.

According to the integration plan of our library example we have to integrate the dynamic models 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 dynamic model M_1 is shown in figure 4.

However, integrating dynamic models whose relationship is mixed is much more complicated, a further detailed analysis is necessary. Furthermore the integration of mixed related dynamic models cannot be done automatically. We are only able to support the designer by computing integration recommendations.

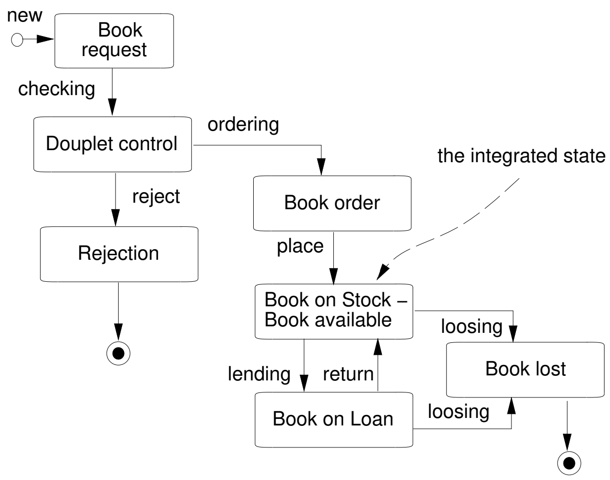


Figure 4: The integrated dynamic model M_1

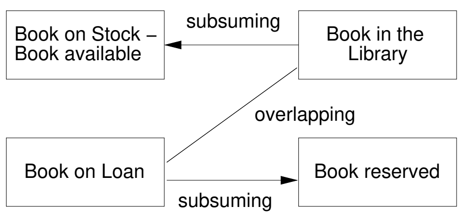


Figure 5: The state relationship graph

Just as dynamic models the states of different dynamic models have relationships. 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 dynamic models. An edge between a state of one model and a state of another model exists if there is a relationship other than a disjoint one 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 between the dynamic models to integrate are disjoint.

As we do not consider disjoint relationships in the state relationship graph, the graph must not be fully connected. For each connected subgraph the states of the dynamic models have to be integrated according to the following way:

1. If there are state hierarchies (state generalizations or state aggregations) they

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 4: State specifications after the transformation

are decomposed using appropriate schema transformations.

2. The involved states of a connected subgraph 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. An appropriate heuristic for the combination of states are for instance the event occurrences. For example, states, which are source states of event occurrences of the same event, should be combined to a single state or to a state generalization.

The first and the second step can be done automatically by using appropriate schema transformations. However, for combining states exist several integration possibilities. According to several heuristics we compute a set of integration recommendations for the designer to choose from.

To conclude our library example we transform the involved states of the state relationship graph into disjoint states, resulting in the states Book in the Library - not reserved, Book borrowed - not reserved and Book borrowed - reserved. The specification of these states are shown in table 4.

The transformed states with their event occurrences are shown in figure 6 (not involved states are dotted). Note, that the application of schema transformation leads to copying event occurrences.

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 event occurrences 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.

5 Conclusion

In this work we have presented a methodology for integrating object oriented views with the main direction to the integration of behaviour models.

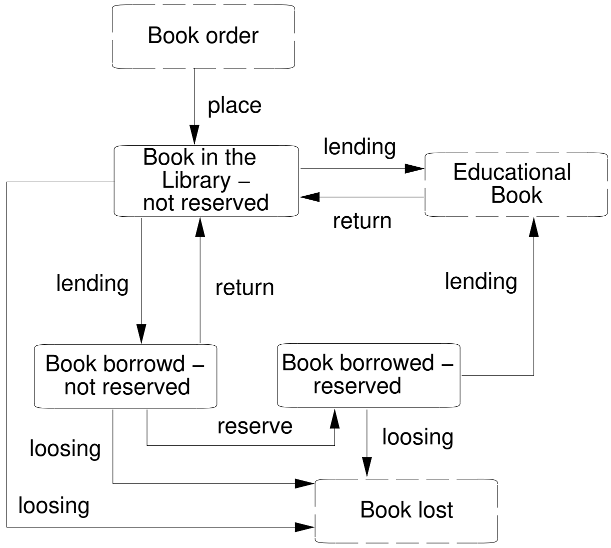


Figure 6: The integrated states

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 determine an overall structure of the integrated dynamic model. Based upon relationships between dynamic models integration operators are defined, which integrates the dynamic models. 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 dynamic models takes place. Most of the defined integration operators can be performed automatically. However, for mixed related dynamic models a further detailed analysis of states and events is necessary.

It was our aim to relieve the designer by automizing the integration as much as possible. In some situations the integration of behaviour models can be done without the aid of a designer. Even the integration of mixed related dynamic models could be computed by using additional heuristics for the combination of states. However, the designer is allowed 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 automatically carry out their consequences in the model.

References

[BL84] 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, November 1984.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323 – 364, December 1986.

[CBB+97] 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.

[Che76] P. Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transaction on Database Systems*, pages 9–36, March 1976.

[EF94] J. Eder and H. Frank. Schema Integration For Object Oriented Database Systems. In M. Tanik et al., editor, *Software Systems in Engineering*. ASME, 1994. Procceedings of the ETCE, New Orleans.

[FE97] H. Frank and J. Eder. A Meta-Model for Dynamic Models. Technical report, Institut für Informatik, Universität Klagenfurt, March 1997. http://www.ifi.uni-klu.ac.at/cgi-bin/publ_search.

[Fra96] H. Frank. *View Integration für objektorientierte Datenbanken*. PhD thesis, Institut für Informatik, Universität Klagenfurt, 1996.

[GLN92] 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, January 1992.

[GPNS92] 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.

[Har88] D. Harel. On Visual Formalisms. *Communications of the ACM*, 31(5):514 – 530, May 1988.

[LM93] 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.

[MT93] 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, September 1993.

[NEL86] S. B. Navathe, R. Elmasri, and J. Larson. Integrating User Views in Database Design. *IEEE Computers*, pages 185–197, January 1986.

[NP92] 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.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall International, Inc, 1991.

[Rum93] J. Rumbaugh. Controlling Code: How to Implement Dynamic Models. *Journal of Object-Oriented Programming*, May 1993.

[Sch87] M. Schrefl. A Comparative Analysis of View Integration Methodologies. In R. Traunmüller R Wagner and H. Mayr, editors, *Informationsbedarfsermittlung und -analyse für den Entwurf von Informationssystemen*, pages 119–136, 1987. Fachtagung EMISA.

[She91] A. P. Sheth. Issues in Schema Integration: Perspective of an Industrial Researcher. In *ARO-Workshop on Heterogeneous Databases*, September 1991.