A Meta-Model for Dynamic Models

Technical Report

(Draft)

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

State charts are a popular representation technique for the conceptual modeling of the dynamics of a universe of discourse. However, designers are not supported in their work with dynamic models as they are for working with static models. We present a meta-model and a formalization of the semantics of a state chart language. Important results are the definition of the equivalence of dynamic models and a sound and complete axiomatization of the equivalence. Based on this we define a set of basic schema transformations which do not change the semantics of a model. These schema transformations can be used to successively transform dynamic models to achieve design goals or to prepare dynamic models for implementation.

Contents

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 behavior of the objects is documented in a dynamic model (compare [Boo91, RBP+91, CAB+94]). 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 is not supported as well. Open issues are for example the formalization of the semantics of dynamic models, generalization and inheritance of dynamic models, and transformations of dynamic models (compare [KS94, Fir96, KS96]). The aim of the work reported here is to contribute to a better understanding of dynamic models and to support the modeling process.

For designing static models designers or analysts start from an initial model and successively transform this model to achieve design goals and meet quality criteria. In the end the model is in a form which is well suited to be mapped to a logical model and thus serves as a specification of the implementation. This process is supported by a well understood representation language and the provision of schema transformations which maintain the semantics of the model (compare [BCN92]). In our opinion, a similar process should be made available for the development of dynamic models.

Assumptions and scope

For this work we assume that the static part of the model is already developed. For the representation of dynamic aspects we focus on the modeling of the dynamics of a single type or class of the static model. We represent dynamic models with a popular state chart language (compare [RBP+91, Rum93, Har88]). We consider these state charts to serve several purposes. First they are a representation technique to capture the dynamics of objects in the universe of discourse. Second, dynamic models support the communication between users, analysts, designers and implementors. Finally, state charts are (partial) specifications for the implementation of an information system.

The major contributions of this paper are

- formalization of state charts for conceptual modeling
- definition of the semantics of state charts
- model-theoretic definition of the equivalence of state charts together with a sound and complete axiomatization
- a complete set of basic schema transformations for deriving equivalent dynamic models

The paper is organized as follows. In section 2 we introduce the state chart language and a meta-model for dynamic models. In section 3 we discuss the equivalence of dynamic models as equivalence of method specifications. In section 4 we present a set of basic equivalence transformations for dynamic models. The inverse transformations of the basic transformations are presented in section 5. In section 6 we discuss some properties of the schema transformation. We prove, that if two dynamic models are equivalent then they can be transformed into each other. In section 7 we draw some conclusions and discuss some applications of this work.

2 The meta-model

2.1 Overview

The dynamic model for a given type consists primarily of *model restrictions*, *states* and *events*. In figure 1 we present a meta-model for dynamic models. In the following we will present all components of dynamic models and give the necessary formalization for the succeeding sections.

We assume that the static model (types and their relationships) have already be defined. Furthermore, we assume that there is a language for defining predicates on objects. We use $\mathcal{TQL}++$ ([FM94a, FM94b, Mos95]) for this purpose, however, the following does not depend on this choice of a language.

Model restrictions:

Model restrictions are conditions that an object must comply with in order to be able to travel actually through a dynamic model.

States:

A state is a collection of values and relationships of an object, it is a subspace of the attribute and relationship space of a type. Intensionally, a state is defined by a predicate on objects of the given type. Extensionally a state is considered as the set of all objects which fulfill this predicate.

States have got a *name*, which must be unique within a dynamic model. The (redundant) attribute *kind* divides states in *atomic states*, *generalizing superstates* and *aggregating superstates*. States can be *initial* or *final* states. Each state has a *range* (represented as a meta-method) which we will define later on. Generally speaking we understand by the range on a state a condition which an object must comply with, so that it can be in the state. The condition of a state can be regarded as a predicate supplying $true$ if an object is in a state, $false$ otherwise. As specification language for conditions we use $\mathcal{TQL}++$ ([FM94a, FM94b, Mos95]). Each atomic state is provided with such a condition that is listed in the attribute *condition*. To each structured state belongs at least one further state.

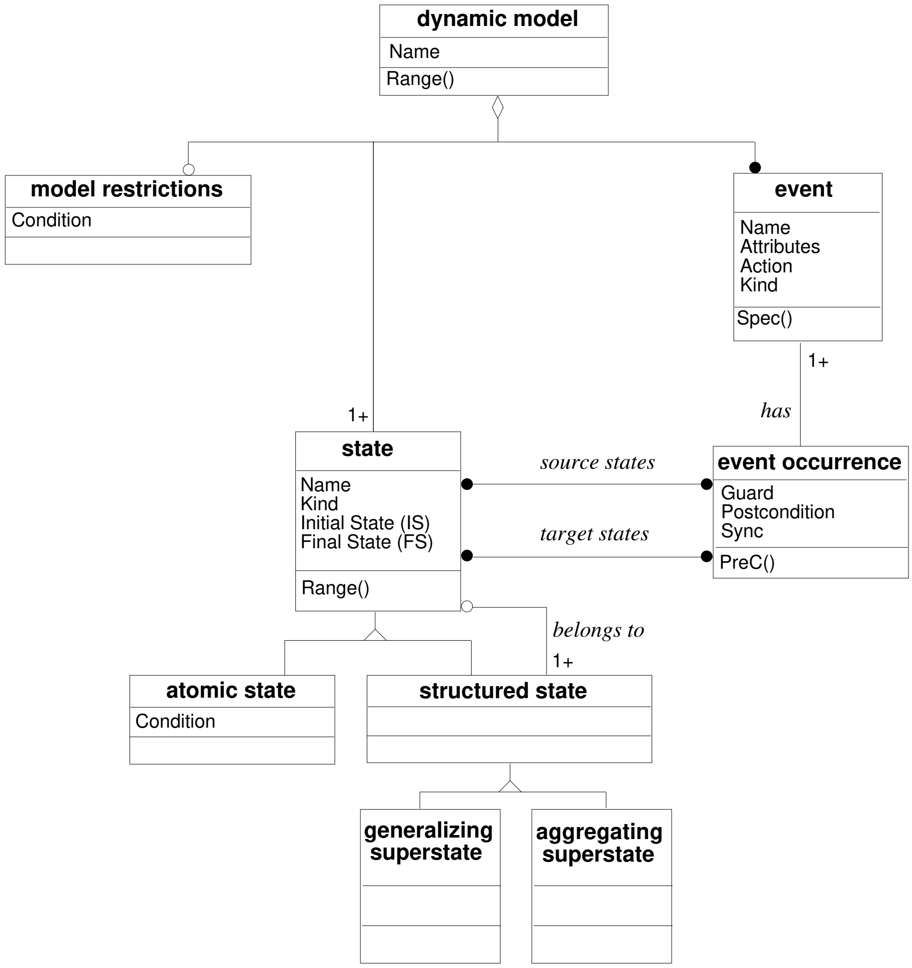

Figure 1: A meta-model for dynamic models

Events:

An event is an incident focused on an object with the aim to carry out a change of state (compare [RBP+91]). Events have got a unique *name*, *attributes* and *actions*. We distinguish between *object producing*, *object destroying* and *transforming* events, this information is stored in the attribute *kind*.

Object producing events create a new object (compare initial states in OMT of [RBP+91]). Object destroying events being carried out delete an object out of the database (compare final states in OMT). Transforming events represent state transitions as defined by OMT. Each event may occur in a dynamic model several times because an event may cause state transitions on different parts of a dynamic model. Let's take e. g. a dynamic models describing the order of an article. It is on various points of this model possible to cancel the order, the event cancel can be found in the model several times. Therefore, we say that each event has at least one event occurrence. Event occurrences have got the attributes *sync*, *guard* and *postcondition*. The attribute sync tells us, if the event occurrence is a synchronizing one in the sense of OMT. A synchronizing event occurrence leads from and/or to a state aggregation. The guard of an event occurrence are those conditions that must be true to apply an event occurrence to an certain object. In other words, the object must comply with the guard of the event occurrence. Postconditions are those conditions that an object complies with after its execution.

Event occurrences can have *source* and *target states*. We designate that states in which an object is before the execution of the event occurrence as source states, the target states are that states in which an object is after its execution. Synchronizing event occurrences, namely event occurrences that lead to or out of a state aggregation can have several source and target states. Not synchronizing event occurrences have exactly one source and exactly one target state respectively in the case of object producing or object destroying event occurrences have no source or no target state.

States, events, event occurrences and dynamic models have further characteristics which we will concentrate later on. For instance we talk about the range of a state or of a dynamic model. In our meta-model these characteristics are realized as meta-methods (resp. as “calculated” attributes). The advantage of such a description in comparison to an (non calculated) attribute is that redundancies can be avoided.

When we talk about these meta-methods or we apply them, we do not mean the meta-method itself, but the result of the carrying-out applied on a precise component of the dynamic model. We write $Z_1.Range()$ and mean the result of the meta-method $Range()$ applied on the state Z_1.

In the following sections we derive from the meta-model a $\mathcal{TQL}++$ schema for the most interesting parts of a dynamic model. Concerning the relationships we are very broad minded, we will (sometimes) derive them redundantly.

As an example we refer to figure 2 representing the static model and the dynamic behavior of a book from the view point of a library. Note that we use the notation of OMT for dynamic models which is suggested in [Rum93] and supported by OMTOOL ([OMT93]).

2.2 Model restrictions

Model restrictions are conditions that an object must comply with in order to be able to travel actually through a dynamic model. Each dynamic model can have such model restrictions, but they are not absolutely necessary. All integrity constraints which are defined for the static parts of the type of the dynamic model

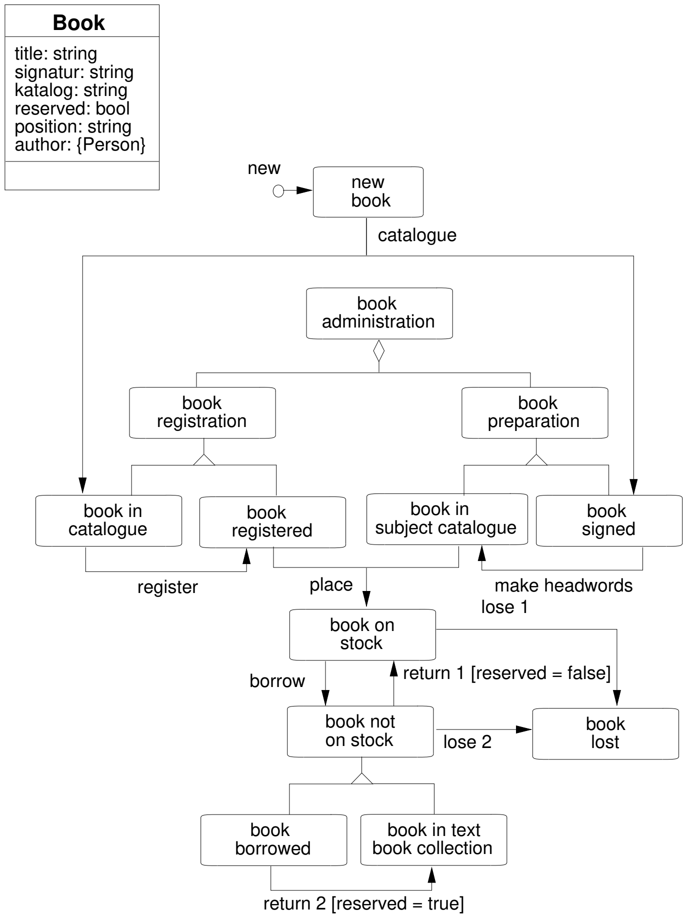

Figure 2: An example

must be added to the model restrictions. Further model restrictions, based upon the attribute and relationship space of the type, can be specified as conditions for the dynamic model. We use $\mathcal{TQL}++$ as specification language for model restrictions.

> Let's take for instance a dynamic model which only is applicable for a certain kind of books, e. g. for didactic books. An adequate model restriction for the dynamic model of the type book looks like this.catalogue = "didactic book"

Model restrictions are only a more comfortable way of naming the conditions for states. For each state of a model a certain model restrictions is valid, also for the postconditions of the event occurrences. For all following explanations we must extend the conditions of atomic states and the postconditions of the event occurrences with the model restrictions. The necessary algorithm in pseudo code:

for all $z \in M$.Atomic_States **do**
  z.Condition := z.Condition $\wedge$ M.Model_Restriction
end for
for all $ex \in M$.Event_Occurrences **do**
  ex.Postcondition := ex.Postcondition $\wedge$ M.Model_Restriction
end for

2.3 States

A state is a collection of values and relationships of an object, it is a subspace of the attribute and relationship space of a type. Intensionally, a state is defined by a predicate on objects of the given type. Extensionally a state is considered as the set of all objects which fulfill this predicate.

States have the form

```
State := [
     Name:        str,
     Kind:        (atomic, generalizing superstate,
                   aggregating superstate),
     IS:          bool,
     FS:          bool,
     Belongs_to:  Structured_State,
     Covers:      {State}
           ]

Atomic_State := ISA State
           [
     Condition: Condition_T
           ]
```

A state consists of several components which can be addressed individually. When we use e. g. $Z.Name$, the name of the state Z is meant.

First of all each state has got a *name* which must be unique within a dynamic model. We use the term *kind* in order to distinguish between generalizing superstates, aggregating superstates and atomic states. We mark the initial and the final states of dynamic models with *IS* (for initial state) and *FS* (for final states). An objects "enters" a dynamic model through a initial state and analogous to that leaves it through a final state.

Atomic states, which are states too and therefore inherit all characteristics of states, have in addition the attribute *Condition*. In this attribute the condition of an atomic state is stored. Each atomic state has exactly one condition, which can be a trough conjunction and disjunction constructed complex term. We demand the specification of the conditions for all atomic states of the dynamic model. For our approach we use $\mathcal{TQL}++$ as specification language.

By the condition of a state we understand the representation of all prerequisites an object must comply with to be in this state. In other terms it is possible to find all objects that are at the moment in this state using the state's condition. The condition of a state itself is based on the attribute and relationship space of a type.

> A state of a book in our example is book in text book collection. The condition for this state would look like this.position = ”text book collection” $\wedge$ this.reserved = false.

Structured states are generalizing or aggregating superstates. To each structured state *belongs* at least one further state, one structured state may *cover* several other states. We use structured states to represent alternatives (state generalizations) or parallelism (state aggregations).

> In our example of figure 2 the state book administration is an aggregating superstate. The states book registration, book preparation and book not on stock are generalizing superstates. All other states are atomic states. The states book borrowed and book in text book collection belong to the structured state book not on stock.

Furthermore states are divided into *elementary* and *non-elementary* states.

DEFINITION: An *elementary* state which can be an atomic state, a generalizing superstate or an aggregating superstate, doesn't belong itself to a structured state. (1)

Elementary states are in a way the “elements” of a dynamic model which can be split up further more. An elementary state doesn't belong to any other state. Elementary states are therefore the 'top-level' states of a dynamic model.

> In the library example the states new book, book administration, book on stock, book not on stock and book lost are elementary states. All other states are non-elementary states.

While the conditions of atomic states have to be stored in the meta-model, the conditions of structured states are computed. We define the *Range* of a state Z as follows:

Definition of the range of states

Definition: The range of a state Z is defined as (2)

- $Z.Condition$, if Z is an atomic state.
- the disjunction of all the states' Z_i ranges, that belongs to Z if Z is a generalizing superstate.
- the conjunction of all the states' Z_i ranges, that belongs to Z if Z is an aggregating superstate.

In the meta-model the range of a state is realized as a meta-method (calculated attribute). As result we get the condition of a state. From now on $Z.Range()$ will be used as an abbreviation for the condition of the state Z.

For instance book in text book collection.Range() results in this.position = "text book collection" ∧ this.reserved = false.

In addition we define the predicate $Z(o)$ which supplies *true* if the object o complies with the condition (the range) of the state Z, otherwise it supplies *false*.

The ranges of the states form the basis for the definition of relationships between states. Five different relationships are defined: *equivalent states*, *included states*, *overlapping states*, *orthogonal states*, and *disjoint states*. The set of all possible extensions of the type T is called $P(T)$.

Definition of relationships between states

Definition: The states Z_1 and Z_2 of the type T are called *equivalent*, if (3)

$$\forall o \in P(T) : Z_1(o) \leftrightarrow Z_2(o)$$

Definition: The state Z_2 of the type T *includes* the state Z_1 of the type T, if (4)

$$\forall o \in P(T) : Z_1(o) \rightarrow Z_2(o)$$

Definition: The states Z_1 and Z_2 of the type T are called *overlapping*, if (5)

$$\exists o \in P(T) : Z_1(o) \ \wedge \ Z_2(o)$$

Definition: The generalizing superstates Z_1 and Z_2 of the type T are called *orthogonal*, if (6)

$$Z_1 \; equivalent \; Z_2 \wedge \forall z \in Z_1.Covers, \forall z' \in Z_2.Covers \rightarrow$$
$$\exists o \in P(T) : z(o) \wedge z'(o)$$

A generalizing superstate consists of a set of states which are stored in the attribute *Covers*. For two generalizing superstates Z_1 and Z_2 being orthogonal their ranges must be equivalent and each state z of Z_1 overlaps with each state z' of Z_2 (and vice versa).

Definition: The states Z_1 and Z_2 of the type T are called *disjoint*, if (7)

$$\forall o \in P(T) : \neg \left(Z_1(o) \; \wedge \; Z_2(o)\right)$$

Based on the ranges of the states and the definitions of relationships between states we think about the correctness of the states of a dynamic model. The statements of OMT are followed, but we are able to describe them in a more formal way.

Definition: The states of a dynamic model are *correct*, if (8)

1 all elementary states are disjoint,
2 all states, belonging to the same generalizing superstate are disjoint,
3 all states, belonging to the same aggregating superstate are orthogonal.

Theorem: The orthogonal relationship *is not transitive* in dynamic models with correct states. (9)

Proof: The orthogonality is symmetrical (Z_1 orthogonal Z_2 implies Z_2 orthogonal Z_1) but not reflexive. Z_1 is not orthogonal to Z_1 as, according to the definition of the orthogonal relationship, Z_1 must be a generalizing superstate and all states belonging to Z_1 must be disjoint (compare definition 8, p. 12). It is obvious, that therefore the orthogonality of states in a dynamic model with correct states is not transitive. ■

2.4 Events and event occurrences

An event is an incident focused on an object with the aim to carry out a change of state. An event is set off explicitly (compare [RBP+91]). Events for a book could be e. g. *borrow* or *lose*.

In dynamic modeling events represent (partial) specifications of the methods for the object type. If an event is set off an object is transfered to a new state. The model defines which conditions (preconditions) an object has to fulfill in order to be able to react to an event and which conditions (postconditions) an object fulfills after the state change. These pre- and postconditions are primarily states of the dynamic models. Events are therefore usually represented as arcs between the states of the model. However, it is not always possible to find a partition of the range of a model into states, such that each pre- and postcondition of all events equal exactly one state. To overcome this situation we allow that an event appears several times in a dynamic model and we distinguish between the event and *event occurrences*.

> Consider the event lose with its two event occurrences lose 1 and lose 2 (figure 2). We numbered the event occurrences in order to differ between them. If the event lose is transmitted to a book, in subordination to the concrete state of the book one of the two event occurrences is activated with the consequence of a state change (supposing that the book is in one of the states book on stock or book not on stock).

An event looks like

```
Event := [
  Name:       str,
  Kind:       (transforming,object producing,
               object destroying),
  Attribute: {str},
  Action:     str,
  has:        {Event Occurrence}
         ]
```

The *name* of an event must be unique in a dynamic model, in *kind* events are divided into *transforming*, *object producing* and *object destroying* events. In *attributes* all attributes of an event are listed which are needed when activating the event. In *actions* we describe in an informal way what an event has to do when it is activated. Each event *has* several (at least one) event occurrences.

An event occurrence looks like

```
Event Occurrence := [
  Guard:            Condition_T,
```

```
    Postcondition:    Condition_T,
    Sync:             bool,
    Source_States:    {State},
    Target_States:    {State},
    has_Event:        Event
                    ]
```

Event occurrences could possess a *guard*. This is a (complex) condition an object must comply with so that the event occurrence can cause a state change. If there is no such condition the guard of an event occurrence is *true* (again we use $\mathcal{TQL}++$ for specifying guards).

An object complies with the condition of a *postcondition* after the event occurrence had been applied. *Sync* states whether the event occurrence is a *synchronizing* one or not. Synchronizing event occurrences always lead from and/or to a state aggregation. *Sync* is a redundant attribute (computed) which is *true*, if an event occurrence has several source or target states, *false* otherwise. In *source_states* and *target_states* the source and target states of an event occurrence are stored. In the case on a synchronizing event occurrence there can be more source and target states (that's the reason why these attributes are multi-value attributes). *Has_event* is the connection to the event of the event occurrence.

The postcondition of an event occurrence must imply the range of its target state. Object destroying event occurrences don't have target states, their postcondition is always *true*. If an event occurrence has got several target states the postcondition must imply the ranges of all these states.

Event occurrences have *preconditions*, which are the conditions that an object must fulfill so that an event occurrence can cause a state change of an object. To cause a state change the object must be in the source state of the event occurrence (in the case of synchronizing event occurrences in all source states) and the object must fulfill the condition of the guard of the event occurrence. Therefore the precondition of an event occurrence equals to the conjunction of the guard with the ranges of its source states. Object producing event occurrences don't have source states, their precondition is equivalent with its guard (or *true*, if there is no guard). In our meta-model the precondition of an event occurrence is represented as meta-method $PreC()$, which will be used to find out the precondition of an event occurrence.

According to the source and target state of event occurrences we define the correctness of event occurrences.

Definition: Event occurrences are correct, it they have source and target states according to the following conditions: (10)

1 Non-synchronizing event occurrences of transforming events have exactly one source and one target state.

2 Synchronizing event occurrences could have several source and target states. However, if there are several source states they must belong to the same state aggregation. Several target states must belong to the same state aggregation too.

3 Event occurrences of object producing events do not have source states.

4 Event occurrences of object destroying events do not have target states. Therefore, the postcondition of such event occurrences is $true$.

5 The postcondition of an event occurrence implies the ranges of all its target states.

Regarding the pre- and postconditions of event occurrences one should consider that they are independent from each other. It is not possible to conclude the postcondition of an event occurrence from its precondition. We only know that if an event occurrence should be applied to an object, the object must comply with the precondition. After the application of the event occurrence the object must fulfill the postcondition.

Furthermore events have got a specification which is represented as meta-method $Spec()$ in the meta-model. We refer to the definition 13, p. 17, where we will define the specification of an event.

Like states, we can consider each part of events and event occurrences, e. g. when we use the term $e.Name$ we mean the name of the event e. The kind of the event of the event occurrence ex is meant by the term $ex.has\_Event.Kind$.

Consider the example in figure 2. States in these example are new book, book administration, book registration etc. Events for instance are new or lose The events lose and borrow occur several times in the dynamic model, we numbered their event occurrences.

Book not on stock is a generalizing superstate covering the states book borrowed and book in text book collection. Their ranges must be disjoint.

The state book administration is an aggregating superstate to which the generalizing superstates book registration and book preparation belong to. These generalizing superstates must be orthogonal.

The event occurrence new is an object producing one with the target state new book. Its precondition is $true$, its postcondition must imply the range of new book. The precondition of the event occurrence return 1 is the conjunction its guard this.reserved = true with the range of book borrowed, its postcondition must imply the range of book in text book collection .

The event occurrence place is a synchronizing one with the source states book registered and book in subject catalogue. Its precondition results from the conjunction of the source states' ranges.

Catalogue is a synchronizing event occurrence with the target states book in catalogue and book signed. Its postcondition must imply the ranges of both target states.

Based upon the dynamic model we can derive (partial) method specifications from the events and their event occurrences. Each event of the dynamic model becomes a method (with the attributes as parameters). The event occurrences preconditions are used in order to determine the conditions in which the method can be applied to an object. The postconditions of the event occurrences specify the conditions (in subordination to the corresponding precondition) an object must comply with after the application of the method.

2.5 Correct dynamic models

A dynamic model looks like:

```
Dynamic Model = [
     Name:              str,
     States:            {State},
     Events:            {Event},
     Model_Restriction: Condition_T
                ]
```

Each component of a dynamic model can be addressed, we use $M.Name$ and mean the name of the dynamic model M. By $Z \in M.States$ is meant that Z is a state of the dynamic model M. Although they don't exist explicitly the set of event occurrences can be addressed with $M.Event\_Occurrences$ (we stored all the event occurrences of an event in the attribute *has*).

Based upon the ranges of states and the correctness of event occurrences we define a correct dynamic model. In our following considerations we assume correct dynamic models.

DEFINITION: A dynamic model is called *correct* if (11)

1 all states are correct according to definition 8, p. 12, and

2 all event occurrences are correct according to definition 10, p. 15

In analogy to states, dynamic models have also a range which is defined as:

DEFINITION: The *range* of a correct dynamic model results from the range's disjunction of all *elementary* states of the dynamic model. (12)

The range of a dynamic model is again a (complex) condition, constructed by the disjunction of the elementary states' ranges of the dynamic model (and that is under no circumstances automatically true). In our meta-model the range of a dynamic model is represented as meta-method $Range()$.

3 Equivalence of dynamic models

We wish to support designers to work with dynamic models in a similar way as they already can do with the static models. In particular, our goal is to support the transformation of dynamic schemas without changing the semantics. For this purpose we need a clear definition when dynamic models are equivalent. Our definition is based on the consideration that dynamic models are equivalent, if they provide the same partial specification for the development of methods. So the equivalence of correct dynamic models ($M_1 \equiv M_2$) bases on equivalent model ranges and equivalent events. We will first define the equivalence in a model-theoretic way and then present a sound and complete axiomatization which will then be used to prove that schema transformations preserve equivalence.

First of all we define what is understood by the specification of an event. The specification of an event is the set of conditional pairs of the form $\{(Pre_1, Post_1), \ldots, (Pre_n, Post_n)\}$. One pair $(Pre_i, Post_i)$ indicates that an object which satisfies the condition Pre_i (we say $Pre_i(o)$, if the object satisfies the condition) after the application of the event (actually of the corresponding event occurrence) satisfies the condition $Post_i$. We take the pre- and postconditions of the event occurrences of the event e in order to calculate the specification of the event:

DEFINITION: The specification $Spec$ of the event e is defined as (13)

$$e.Spec = \{(ex.PreC(), ex.Postcondition) \mid ex \in e.has\}$$

The specification of an event is computed in our meta-model by collecting all the pre- and postconditions of the corresponding event occurrences of the event (listed in the attribute *e.has*). In the meta-model the specification of an event is realized as meta-method $Spec()$.

The predicate $e.Post(o)$ is defined as the postcondition an object o statisfies after the event e occurred.

DEFINITION: The predicate $e.Post(o)$ for an event e and an object o is defined as (14)

$$e.Post(o) := \bigvee\{Post|\exists Pre : (Pre, Post) \in e.Spec() \wedge Pre(o)\}$$

whereby

$$\begin{aligned} \bigvee\emptyset &= false \\ \bigvee\{Post\} &= Post \\ \bigvee\{Post_1 \ldots Post_n\} &= Post_1 \vee \ldots \vee Post_n \end{aligned}$$

The predicate $Post(o)$ is defined as the disjunction of all postconditions of conditional pairs $(Pre, Post)$ from $e.Spec()$ for which the object o satisfy the precondition ($Pre(o)$). The predicate supplies $false$, if the object o doesn't fulfill any of the preconditions of the event specification of e.

DEFINITION: The event specifications of the two events e_1 and e_2 are *equivalent* ($e_1.Spec() \equiv e_2.Spec()$), if (15)

$$\forall o \in P(T) : e_1.Post(o) \leftrightarrow e_2.Post(o)$$

We say that two event specifications are equivalent if $Post$ applied to both specifications for all objects from $P(T)$ supplies equivalent conditions.

DEFINITION: Two events e_1 and e_2 are *equivalent* ($e_1 \equiv e_2$), if they have the same name, the same attributes and the same kind and their event specifications are equivalent. (16)

THEOREM: The equivalence of events is *reflexive*, *symmetrical* and *transitive*, consequently an equivalence relation. (17)

PROOF: The equivalence of two events e_1 and e_2 is due to the same name, the same attributes and the same kind of events and based on equivalent event specifications of e_1 and e_2 (compare definition 16). The conformity of event names, the attributes of events and the kind of events is trivially reflexive, symmetrical and transitive. We must show this for the event specifications, that means:

(1) $e_1.Spec() \equiv e_1.Spec()$ (reflexivity)

(2) $e_1.Spec() \equiv e_2.Spec() \leftrightarrow e_2.Spec() \equiv e_1.Spec()$ (symmetry)

(3) $(e_1.Spec() \equiv e_2.Spec()) \wedge (e_2.Spec() \equiv e_3.Spec()) \rightarrow$
$e_1.Spec() \equiv e_3.Spec()$ (transitivity)

ad (1) $e_1.Spec() \equiv e_1.Spec()$ follows directly from the definition of event specifications (compare definitions 13 and 14).

ad (2) $e_1.Spec() \equiv e_2.Spec() \leftrightarrow e_2.Spec() \equiv e_1.Spec()$ is valid because of the symmetry of equivalent logical terms (in our approach $\mathcal{TQL}++$ terms).

ad (3) $(e_1.Spec() \equiv e_2.Spec()) \wedge (e_2.Spec() \equiv e_3.Spec()) \rightarrow (e_1.Spec() \equiv e_3.Spec())$ follows because of the transitivity of equivalent logical terms (in our approach $\mathcal{TQL}++$ terms). ■

Now we are interested which changes of event specifications are possible in this equivalence relation since manipulations of event occurrences lead to a change of event specifications. We define the relation Ξ for event specifications. It means the left part of the relation Ξ can be changed to the right part and vice versa.

Definition: Let S, S_1, S_2 and S_3 be event specifications. Let additionally Pre, Pre_1, Pre_2, Pre_i and Pre_j as well as $Post$, $Post_1$, $Post_2$, $Post_i$ and $Post_j$ be Pre- and Postconditions ($\mathcal{TQL}++$ terms). Then: (18)

(1) $S \cup \{(Pre_1, Post), (Pre_2, Post)\} \Xi S \cup \{(Pre_1 \vee Pre_2, Post)\}$

(2) $S \cup \{(Pre, Post_1), (Pre, Post_2)\} \Xi S \cup \{(Pre, Post_1 \vee Post_2)\}$

(3) $\{(false, Post)\} \,\Xi\, \emptyset$

(4) $\{(Pre, false)\} \,\Xi\, \emptyset$

(5) $\{(Pre_i, Post_i)\} \,\Xi\, \{(Pre_j, Post_j)\}$ if $Pre_i \leftrightarrow Pre_j \wedge$
$Post_i \leftrightarrow Post_j$

(6) $(S_1 \,\Xi\, S_2) \wedge (S_2 \,\Xi\, S_3) \;\rightarrow\; S_1 \,\Xi\, S_3$

According to the definitions 18(1) and (2) we may summarize event specifications with equivalent postconditions through disjunction of their preconditions as well as event specifications with equivalent preconditions through disjunction of their postconditions. The definitions 18(3) and (4) allow us to remove event specifications whose pre- or postconditions result in $false$. The definition 18(5) states that pairs of event specifications following to the relation Ξ are equivalent if their pre- and postconditions are equivalent terms (in our approach equvalent $\mathcal{TQL}++$ terms). In definition 18(6) the transitivity of the relation Ξ is determined.

THEOREM: Let S_1, S_2 and T be event specifications. Then (19)

(1) $S_1 \Xi S_1$ and

(2) $S_1 \Xi S_2 \rightarrow (S_1 \cup T) \Xi (S_2 \cup T)$

PROOF:

(1) $S_1 \Xi S_1$ follows from the transitivity of logical terms (in our approach $\mathcal{TQL}++$ terms; compare definition 18, p. 19).

(2) If $S_1 \Xi S_2$, then there must be a sequence of event specifications $S_i \ldots S_k$ based upon operations according to the definition 18, p. 19, so that

$$S_1 \Xi S_i \Xi \ldots \Xi S_k \Xi S_2.$$

It is easy to see, that, because of the same operations:

$$(S_1 \cup T) \Xi (S_i \cup T) \Xi \ldots \Xi (S_k \cup T) \Xi (S_2 \cup T).$$

■

THEOREM: Let S_1 and S_2 be event specifications. From $S_1 \Xi S_2$ follows $S_1 \equiv S_2$. (20)

PROOF: In order to prove this theorem for the different operations of the definitions 18, p. 19 we have to show that $e.Post(o)$ for any object o returns equivalent terms before and after an operation of the relation Ξ.

(1) The definition 18(1) states, that $S \cup \{(Pre_1, Post), (Pre_2, Post)\} \Xi S \cup \{(Pre_1 \cup Pre_2, Post)\}$. Let $S' = S \cup \{(Pre_1, Post), (Pre_2, Post)\}$, with $S'' = S \cup \{(Pre_1 \vee Pre_2, Post)\}$. Let e' be the event with the event specification S', e'' the event with the event specification S'' and e the event with the event specification S.

(a) Let o be an object from $P(T)$ with $Pre_1(o)$. $P = e.Post(o)$; $P' = e'.Post(o)$ and $P'' = e''.Post(o)$. Then P' is equal to $P \vee Post$ and P'' is equal to $P \vee Post$. It is easy to see that $P' \equiv P''$.

(b) Let o be an object from $P(T)$ with $Pre_2(o)$. Analogous to (a) it is easy to see that $P' \equiv P''$.

(c) Let o be an object from $P(T)$ with $Pre_1(o) \wedge Pre_2(o)$; $P = e.Post(o)$. $P' = e'.Post(o)$ and $P'' = e''.Post(o)$. P' is equal to $P \vee Post \vee Post$, P'' is equal to $P \vee Post$. It is easy to see that $P' \equiv P''$.

(d) Let o be an object from $P(T)$ with $\neg\, Pre_1(o) \wedge \neg\, Pre_2(o)$. It follows directly that $P \equiv P' \equiv P''$.

From (a) to (d) follows, that $S \cup \{(Pre_1, Post), (Pre_2, Post)\} \equiv S \cup \{(Pre_1 \vee Pre_2, Post)\}$

(2) The definition 18(2) states, that $S \cup \{(Pre, Post_1), (Pre, Post_2)\} \;\Xi\; S \cup \{(Pre, Post_1 \vee Post_2)\}$. Let S' be $S \cup \{(Pre, Post_1), (Pre, Post_2)\}$ and S'' be $S \cup \{(Pre, Post_1 \vee Post_2)\}$. Let e' be the event with the event specification S', e'' be the event with the event specification S'' and e be the event with the event specification S.

(a) Let o be an object from $P(T)$ with $Pre(o)$. $P = e.Post(o)$, $P' = e'.Post(o)$ and $P'' = e''.Post(o)$. P' is equal to $P \vee Post_1 \vee Post_2$ and P'' equal to $P \vee (Post_1 \vee Post_2)$. Obvious $P' \equiv P''$.

(b) Let o be an object from $P(T)$ with $\neg\, Pre(o)$. It follows that $P \equiv P' \equiv P''$.

It is easy to see that $S \cup \{(Pre, Post_1), (Pre, Post_2)\} \equiv S \cup \{(Pre, Post_1 \vee Post_2)\}$

(3) The definition 18(3) states, that $\{(false, Post)\} \;\Xi\; \emptyset$. There can be no object from $P(T)$, that fulfills the precondition $false$. The predicate $Post(o)$ applied to an event specification $\{(false, Post)\}$ results in an empty set of postconditions and therefore returns $false$ (compare definition 14, p. 18). The predicate $Post(o)$ applied to an empty event specification again results in an empty set of postconditions and returns $false$ too. Obvious $\{(false, Post)\} \equiv \emptyset$

(4) The definition 18(4) states, that $\{(Pre, false)\} \;\Xi\; \emptyset$. Each object o from $P(T)$ either fulfills the Precondition Pre or does not.

(a) Let o be an object from $P(T)$ with $Pre(o)$. $Post(o)$ applied to the event specification $\{(Pre, false)\}$ returns $false$ (compare definition 14, p. 18). $Post(o)$ applied to an empty event specification returns an empty set of postconditions and, therefore, $false$.

(b) Let o be an object from $P(T)$ with $\neg\, Pre(o)$. $Post(o)$ applied to $\{(Pre, false)\}$ results in an empty set of postconditions and therefore returns $false$, as well as $Post(o)$ applied to an empty event specification.

It follows, that $\{(Pre, false)\} \equiv \emptyset$

(5) $\{(Pre_i, Post_i)\} \equiv \{(Pre_j, Post_j)\}$ if $Pre_i \leftrightarrow Pre_j \wedge Post_i \leftrightarrow Post_j$ follows from the equivalence of logical terms (in our approach of $\mathcal{TQL}++$ terms).

(6) The transitivity of event specifications follows from the transitivity of equivalent logical terms (in our approach of $\mathcal{TQL}++$ terms).

From (1) to (6) follows, that $S_1 \,\Xi\, S_2 \rightarrow S_1 \equiv S_2$. ■

THEOREM: Let S_1 and S_2 be event specifications. From $S_1 \equiv S_2$ follows $S_1 \,\Xi\, S_2$. (21)

PROOF: To prove this theorem it is sufficient to show that:

(1) $\exists\, R \mid R \equiv S_1 \wedge \forall (Pre_i, Post_i), (Pre_j, Post_j) \in R :$

  (a) $Pre_i, Pre_j, Post_i, Post_j \neq false$

  (b) $Pre_i \wedge Pre_j \neq false \rightarrow Pre_i = Pre_j$

(2) $\exists\, Q \mid Q \equiv S_2 \wedge \forall (Pre_i, Post_i), (Pre_j, Post_j) \in Q :$

  (a) $Pre_i, Pre_j, Post_i, Post_j \neq false$

  (b) $Pre_i \wedge Pre_j \neq false \rightarrow Pre_i = Pre_j$

(3) $\forall (Pre_{2i}, Post_{2i}) \in Q\, \exists\, R'$ with

  (a) $R' \cup R \,\Xi\, R$ and

  (b) $R' \,\Xi\, (Pre_{2i}, Post_{2i})$

ad (1) and (2) We transform S_1 and S_2 into event specifications where none of the pre- and postconditions results in false. Furthermore the preconditions are demanded to be disjoint. It is easy to see that this can be done by using the operations of the definitions 18(1) - (4) resulting into the event specifications R and Q which are equivalent to S_1 and S_2. From $S_1 \equiv S_2$, $S_1 \equiv R$ and $S_2 \equiv Q$ follows that $R \equiv Q$.

ad (3) Let $(Pre_{2i}, Post_{2i}) \in Q$, than $R' = \{(Pre'_j, Post_j)\} | \exists (Pre_j, Post_j) \in R$ with $Pre_j \wedge Pre_{2i} \neq false \wedge Pre'_j = Pre_j \wedge Pre_{2i}$.

It is easy to see, that $R' \neq \emptyset$ as $R \equiv Q$ and therefore there must be at least one pair $(Pre_j, Post_j) \in R$ with $Pre_j \wedge Pre_{2i} \neq false$ (compare definitions 16, p. 18, 15, p. 18 and 14, p. 18).

ad (3a) For each pair $(Pre'_j, Post'_j) \in R'$ there must be a pair $(Pre_j, Post_j) \in R$ with $Post'_j \equiv Post_j$. According to definition 18, p. 19(1) they can be combined to $(Pre'_j \vee Pre_j, Post_j)$ which results in $(Pre_j, Post_j)$ as $Pre'_j \rightarrow Pre_j$ (compare the construction of R'). Therefore $R' \cup R \,\Xi\, R$.

ad (3b) $(Pre_{2i}, Post_{2i}) \in Q$ and R' is the set $\{(Pre'_1, Post_1) \ldots (Pre'_n, Post_n)\}$. Due to the construction of R' each Pre'_i has the form $P' \wedge Pre_{2i}$. Therefore each $Pre'_i \rightarrow Pre_{2i}$.

We transform $(Pre_{2i}, Post_{2i})$ into $Q' = \{(Pre'_{2i}, Post_{2i}), (Pre''_{2i}, Post_{2i})\} \mid \exists (Pre_i, Post_i) \in R'$ with $Pre'_{2i} = Pre_{2i} \wedge Pre_i$ and $Pre''_{2i} = Pre_{2i} \wedge \neg\, Pre_i$. Q' results into the set $\{(Pre_{2i} \wedge Pre_1, Post_{2i}), (Pre_{2i} \wedge \neg\, Pre_1, Post_{2i}) \ldots (Pre_{2i} \wedge Pre_n, Post_{2i}), (Pre_{2i} \wedge \neg\, Pre_n, Post_{2i})\}$. According to definition 18(1) $(Pre_{2i}, Post_{2i}) \;\Xi\; Q'$. With definition 18(2) Q' can be transformed into the set $\{(Pre_{2i} \wedge Pre_1, Post_{2i}), \ldots, (Pre_{2i} \wedge Pre_n, Post_{2i}), (Pre_{2i} \wedge \neg\, (Pre_1 \vee \ldots \vee Pre_n), Post_{2i})\}$.

However, $Pre_{2i} \wedge \neg\, (Pre_1 \vee \ldots \vee Pre_n)$ results in $false$, as $R \equiv Q$. Otherwise there might be an object o fulfilling Pre_{2i} but not any precondition from R. According to definition 18(3) we can remove this from Q'.

Furthermore Q' can be transformed into $\{(Pre_1, Post_{2i}), \ldots (Pre_n, Post_{2i})\}$ as each $Pre_i \rightarrow Pre_{2i}$.

Now $\forall (Pre_j, Post_j) \in R' \; \exists (Pre_j, Post_{2i}) \in Q'$ and vice versa. Note, that according to the conditions for R the preconditions of R' must be disjoint. The preconditions of Q' must be disjoint too. If follows that $Post_j \equiv Post_{2i}$. Otherwise there would be contradiction as the predicate $Post(o)$ for an object fulfilling the precondition Pre_j would result in not equivalent postconditions for R and Q.

From (1) to (3) follows, that $S_1 \equiv S_2 \rightarrow S_1 \;\Xi\; S_2$.

■

We define the equivalence of dynamic models on the base of the equivalence of events.

DEFINITION: Two correct dynamic models M_1 and M_2 are equivalent ($M_1 \equiv M_2$), if (22)

(1) their ranges are equivalent and

(2) all their events are equivalent.

THEOREM: The equivalence of correct dynamic models is *reflexive*, *symmetrical* and *transitive*. (23)

PROOF: The equivalence of dynamic models is based on equivalent ranges and equivalent events. We've already proved that the equivalence of events is reflexive, symmetrical and transitive (compare theorem 17, p. 18). This is also valid for the range of dynamic models which are logical terms (in our approach $\mathcal{TQL}++$ terms). ■

Now we have the formal basis for discussing equivalence transformations of dynamic models. In the following section we introduce a set of basic schema transformations which have the important property that they deliver equivalent dynamic models.

4 Schema transformations

4.1 Overview

Schema transformations are operations on a dynamic model M_1 resulting in a different dynamic model M_2. Each schema transformation deals with a certain aspect of the dynamic model (e. g combines states or shifts event occurrences within a state generalization). In the following we present a set of 21 basic schema transformations which do not change the semantics of the dynamic model according to the definition of equivalence given above. Due to the transitivity of the equivalence of dynamic models complex transformations can be established on this basic set of transformations. In our approach the transformations are treated as meta-methods of the meta-model (e. g. as meta-methods for states).

Each schema transformation changes a correct dynamic model M_1 into a correct dynamic model M_2. We will prove, that M_1 and M_2 are equivalent as defined by definition 22, p. 23. For that it is sufficient to show that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic models according to definition 11, p. 16 and

(3) the *contributions* of the changed event occurrences to the event specifications are equivalent before and after the schema transformation. We prove this for the most general case, all other cases result trivially from that.

The *contribution* of an event occurrence ex to the event specification is a conditional pair $(ex.PreC(), ex.Postcondition)$ (compare definition 13, p. 17). The precondition $ex.PreC()$ is regarded as $(Z_1.Range() \wedge ex.Guard \wedge R)$. R represents according to this point of view the conjunction of the source states' ranges of the event occurrence ex except the state Z_1 if ex is a synchronizing event occurrence (remember, synchronizing event occurrences could have several source states; compare page 7). In the case of a non synchronizing event occurrence ex or if Z_1 is the only source state of ex R is $true$. The source state Z_1 of ex plays an important role during a schema transformation and therefore will be treated especially. In order to get a better view we use for the contribution of an event occurrence $(Z_1 \wedge G \wedge R, P)$ but we mean $(Z_1.Range() \wedge ex.Guard \wedge R, ex.Postcondition)$.

If a transformation is applied, a new dynamic model M_2 based on M_1 is constructed. Basically this means that the complete dynamic model M_1 must be copied before the transformation is applied. However, we do not consider that, we take that granted.

Combined schema transformations can be defined on the base of this transformations which are equivalence transformations too, as the equivalence of dynamic models is transitive (compare theorem 23, p. 23). We will present some combined schema transformations, further combinations are of course possible.

4.2 Shifting event occurrences within a generalization

In a correct dynamic model event occurrences can be shifted within a state generalization. They can be shifted from a state of the state generalization to the generalizing superstate or can be shifted from the generalizing superstate to the states belonging to it.

4.2.1 Shifting event occurrences to a generalizing superstate

With the aid of these schema transformation we shift an event occurrence from a state of a state generalization to the generalizing superstate. We distinguish between the transformations $UpSg$ and $UpTg$. The transformation $UpSg$ shifts an event occurrence with a source state Z to the generalizing superstate of Z. The transformation $UpTg$ shifts an event occurrence with a target state Z to the generalizing superstate of Z.

The schema transformation $UpSg(Z)$ shifts an event occurrence with a source state Z to the generalizing superstate of Z. $UpSg(Z)$ can be regarded as meta-method in our meta-model and looks like:

if Z.Belongs_to.Kind = generalizing superstate **then**
  $self$.Guard := $self$.PreC()
  replace in $self$.Source_States Z through Z.Belongs_to
end if

At this point we want to emphasize again that the components of a dynamic model can be addressed as single parts. We use $Z.Belongs\_to.Kind$ and the kind of the structured state that Z belongs to is meant. In other words we "navigate" through the dynamic model by using so called *path expressions*.

The state Z is replaced by its generalizing superstate in the source states of the shifted event occurrence. However, as the generalizing superstate has a "wider" range than the original state Z the guard of the event occurrence is replaced by its precondition to guarantee that the event occurrence could only be applied to objects that comply with the original precondition.

THEOREM: If a correct dynamic model M_1 is transformed by shifting an event occurrence using the schema transformation $UpSg(Z)$ into a dynamic model M_2 then $M_1 \equiv M_2$. (24)

PROOF: To prove this theorem it is sufficient to show that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) The ranges of M_1 and M_2 are equivalent, as states remain unchanged by the transformation $UpSg$.

ad (2) It is easy to see that the conditions of definition 11, p. 16 holds for M_2.

ad (3) The events remains unchanged if we shift event occurrences, therefore the conformity of the name, kind and attributes of events is automatically given. However, we must show that the contributions of the event occurrences to the event specifications are equivalent before and after the transformation.

Let ex be an event's e event occurrence of the dynamic model having Z_1 as source state. Let Z_1 belonging to the generalizing superstate Z_2. Let B' be the disjunction of the ranges of all states (except Z_1) belonging to the generalizing superstate Z_2. The contribution of ex to the event specification of e before and after the schema transformation $UpSg(Z_1)$:

$$\begin{aligned} before &= (Z_1 \wedge G \wedge R, P) \\ afterwards &= (Z_2 \wedge (Z_1 \wedge G) \wedge R, P), \text{comp. algorithm} \\ &= ((Z_1 \vee B') \wedge (Z_1 \wedge G) \wedge R, P), \text{comp def. 2} \\ &= (Z_1 \wedge G \wedge R, P) \end{aligned}$$

Obviously the contributions of the event occurrence ex to the event specification of e are equivalent before and after the schema transformation.

From (1) to (3) follows that the application of the schema transformation $UpSg$ to an event occurrence of M_1 results into an equivalent dynamic model M_2 ■

The transformation $UpTg$ shifts an event occurrence with a target state Z to the generalizing superstate of Z. $UpTg(Z)$ can be regarded as meta-method in our meta-model and looks like:

if Z.Belongs_to.Kind = generalizing superstate **then**
replace in $self$.Target_States Z through Z.Belongs_to
end if

This transformation only changes the target state of the event occurrence by replacing Z through the generalizing superstate of Z.

THEOREM: If a correct dynamic model M_1 is transformed by shifting an event occurrence using the schema transformation $UpTg(Z)$ into a dynamic model M_2 then $M_1 \equiv M_2$. (25)

PROOF: To prove this theorem it is sufficient to illustrate that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) The ranges of M_1 and M_2 are equivalent, as states remain unchanged by the transformation $UpTg$.

ad (2) Obvious the conditions of definition 11, p. 16 holds for M_2 too, as the transformation does not change the states of the dynamic model. The postcondition of the shifted event occurrence implies the range of its new target state (the generalizing superstate; compare definition 10, p. 15) as the range of the generalizing superstate contains the range of the original target state of the event occurrence (compare definition 2, p. 11).

ad (3) The events remain unchanged by the transformation $UpTg$, therefore the conformity of the names, kind and attributes is granted. Furthermore we have to prove that the contributions of the shifted event occurrence before and after the transformation are equivalent. Obvious the contributions must be equivalent as neither the source states nor the postconditions of the event occurrence are changed.

From (1) to (3) follows that the application of the schema transformation $UpTg$ to an event occurrence of M_1 results into an equivalent dynamic model M_2. ■

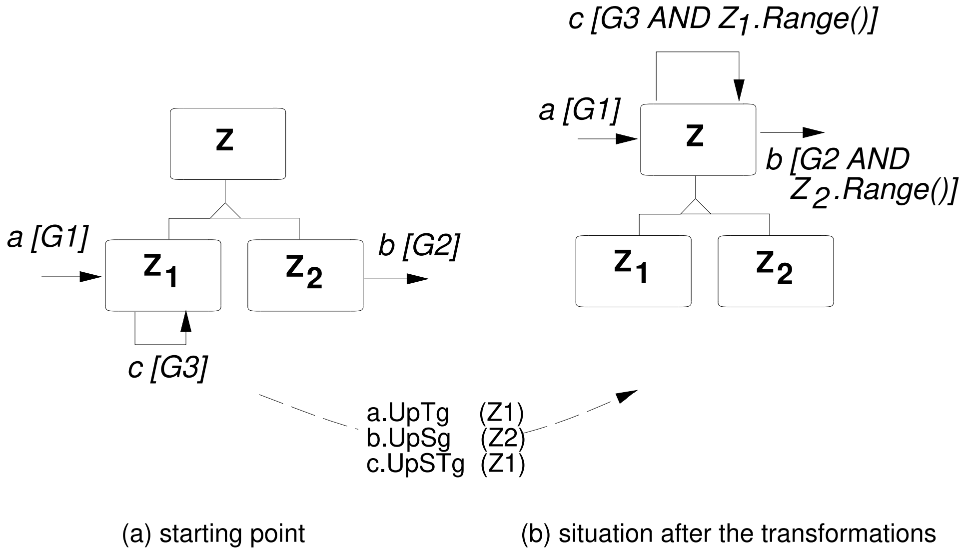

Figure 3: Shifting of event occurrences

Combined schema transformations

Based on the two transformations $UpSg$ and $UpTg$ we can determine a schema transformation $UpSTg(Z)$ shifting event occurrences having Z as source and target state to the generalizing superstate of Z. A corresponding algorithm in pseudo code:

if Z.Belongs_to.Kind = generalizing superstate **then**
  $self$.UpSg(Z)
  $self$.UpTg(Z)
end if

By applying $UpSTg$ to an event occurrence of M_1 an equivalent dynamic model M_2 is produced, as only the equivalence transformations $UpSg$ and $UpTg$ are used and the equivalence of dynamic models is transitive (compare theorem 23, p. 23).

Let's consider the example in figure 3 where we shift the event occurrences a, b and c from the states Z_1 or Z_2 to the generalizing superstate Z by applying the corresponding schema transformations.

4.2.2 Shifting event occurrences from a generalizing superstate

With the aid of these schema transformations we shift event occurrences of a generalizing superstate to those states belonging to the generalizing superstate. We distinguish between $DownSg$ and $DownTg$ (in analogy to $UpSg$ and $UpTg$).

The schema transformation $DownSg(G)$ shifts an event occurrence having the generalizing superstate G as source state to all states belonging to G. The schema transformation can be regarded as a meta-method in our meta-model. A corresponding algorithm in pseudo code:

EX := $\emptyset$
if G.Kind = generalizing superstate **then**
  for all $Z_i \in G$.Covers **do**
    ex := $self$.shallow_copy()
    EX := $EX \cup ex$
    ex.has_Event.has := ex.has_Event.has + ex
    replace in ex.Source_States G through Z_i
  end for
  $self$.has_Event.has := $self$.has_Event.has - $self$
  $self$.delete()
end if
return EX

For each state belonging to G the event occurrence must be copied and the source states are adopted. The function $shallow\_copy$ is a usual process of copying in object oriented databases. In that way a copy of the object with identical attribute values is produced. The original event occurrence is deleted. Afterwards some of the copied event occurrences may have preconditions resulting in $false$. However, according to definition 18, p. 19, these event occurrences can be deleted. Later we present the schema transformation $Clean$ for such situations. The shifted and therefore copied event occurrences are collected in the set EX which are returned as result of the transformation. We will need this set of event occurrences for another schema transformation.

THEOREM: If a correct dynamic model M_1 is transformed by shifting an event occurrence ex using the schema transformation $DownSg(G)$ into a dynamic model M_2 then $M_1 \equiv M_2$. (26)

PROOF: To prove this theorem it is sufficient to show that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) The ranges of M_1 and M_2 are equivalent as states remain unchanged by the transformation $DownSg$.

ad (2) It is easy to see that M_2 is a correct dynamic model according to definition 11, p. 16.

ad (3) The events remain unchanged by the transformation $DownSg$. Therefore the uniformity of the name, kind and attributes is automatically given. In addition we must prove that the contributions of the event occurrences to the event specifications are equivalent too.

Let ex be an event's e event occurrence without any restrictions having the generalizing superstate Z as source state. Let $Z_1 \ldots Z_i$ be those states belonging to the generalizing superstate Z. The contribution of ex to the event specification of the event e before the transformation is:

$$before \;=\; (Z \wedge G \wedge R, P)$$

The event occurrence ex is removed by the transformation and replaced by the event occurrences $ex_1 \ldots ex_i$. The contributions for these event occurrences look like:

$$\begin{aligned} afterwards &= ex_1 : (Z_1 \wedge G \wedge R, P), \text{comp. algorithm} \\ &\quad \ldots \\ &\quad ex_i : (Z_i \wedge G \wedge R, P), \text{comp. algorithm} \\ &= (((Z_1 \wedge G \wedge R) \vee \ \ldots \ \vee (Z_i \wedge G \wedge R)), P), \text{comp. def. 18(1)} \\ &= ((Z_1 \vee \ \ldots \ \vee Z_i) \wedge G \wedge R, P) \\ &= (Z \wedge G \wedge R, P), \text{comp def. 2} \end{aligned}$$

The transformation produces the event occurrences $ex_1 \ldots ex_i$. Their contributions to the event specification can be combined by the disjunction of their preconditions as their postconditions are equivalent according to the relation Ξ (compare definition 18, p. 19). We get $(Z_1 \vee \ \ldots \ \vee Z_i) \wedge G \wedge R$ by transforming the preconditions. However, this is equivalent to $Z \wedge G \wedge R$ because the disjunction of the states' ranges belonging to the generalizing superstate is equivalent to the range of the generalizing superstate (comp. definition 2, p. 11).

The contribution of ex to the event specification of the event e is equivalent to the contributions that the event occurrences $ex_1 \ldots ex_i$ supply to the event specification.

From (1) to (3) follows that the application of the transformation $DownSg$ to an event occurrence of M_1 results into an equivalent dynamic model M_2. ■

The schema transformation $DownTg(G)$ shifts an event occurrence having the generalizing superstate G as target state to all states belonging to G. The schema transformation $DownTg$ can be regarded as a meta-method of the meta-model. A corresponding algorithm looks like:

if G.Kind = generalizing superstate **then**
  for all $Z_i \in G$.Covers **do**
    ex := $self$.shallow_copy()
    ex.has_Event.has := ex.has_Event.has + ex
    ex.Postcondition := ex.Postcondition $\wedge$ Z_i.Range()
    replace in ex.Target_States G through Z_i
  end for
  $self$.has_Event.has := $self$.has_Event.has - $self$
  $self$.delete()
end if

Again for each state belonging to the generalizing superstate the shifted event occurrence must be copied and the target states are changed. Furthermore the postconditions of the copied event occurrences are adopted by the conjunction of the original postcondition with the range of their new target states. The original event occurrence is deleted. If afterwards the postcondition of an event occurrence results in $false$, it can be deleted (compare definition 18, p. 19).

THEOREM: If a correct dynamic model M_1 is transformed by shifting an event occurrence ex using the schema transformation $DownTg(G)$ into a dynamic model M_2 then $M_1 \equiv M_2$. (27)

PROOF: To prove this theorem it is sufficient to illustrate that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) The ranges of M_1 and M_2 are equivalent, as states remain unchanged by the transformation $DownTg$.

ad (2) The transformation doesn't change anything on states of the dynamic model. The conditions of definition 8, p. 12 hold automatically.

By shifting an event occurrence from the generalizing superstate to its covering states the event occurrence is copied. Target states and postconditions are changed. The new postcondition of a copied event occurrences either implies the range of its new target state or results in $false$ (those event occurrences may be removed without loosing the equivalence according to the definition 20, p. 20). Therefore the conditions of correct occurrences hold for M_2 (compare definition 10, p. 15). M_2 is a correct dynamic model.

ad (3) The events remain unchanged by the transformation $DownTg$. Therefore the uniformity of the name, kind and attributes is automatically given. In addition we must prove that the contributions to the event specifications are equivalent too.

Let ex be an event's e event occurrence without any restrictions having the generalizing superstate Z as target state. Let $Z_1 \ldots Z_i$ be those states belonging to the generalizing superstate Z. The contribution of ex to the event specification of the event e before the transformation is:

$$before = (ex.PreC(), P)$$

Let $ex_1 \ldots ex_i$ be the event occurrences produced by the transformation. The contributions of these event occurrences to the event specification of e (as the source states remain unchanged by the transformation the preconditions of the copied event occurrences are equivalent to $ex.PreC()$):

$$\begin{aligned} afterwards &= ex_1 : (ex.PreC(), P \wedge Z_1), \text{comp. algorithm} \\ & \ldots \\ & ex_i : (ex.PreC(), P \wedge Z_i), \text{comp. algorithm} \\ &= (ex.PreC(), P \wedge (Z_1 \vee \ldots \vee Z_i)), \text{comp. def. 18(2)} \\ &= (ex.PreC(), P \wedge Z), \text{comp. def. 2} \\ &= (ex.PreC(), P), \text{comp. def. 11} \end{aligned}$$

The transformation produces the event occurrences $ex_1 \ldots ex_i$. Their contributions to the event specification can be combined by the disjunction of their postconditions as their preconditions are equivalent (according to definition 18, p. 19, of the relation Ξ). We get $P \wedge (Z_1 \vee \ldots \vee Z_i)$ by transforming, which is equivalent to $P \wedge Z$ as the disjunction of the states' ranges belonging to the generalizing superstate is equivalent to the range of the generalizing superstate (compare definition 2, p. 11). As the postcondition P of the event occurrence ex from M_1 must imply the range of its target state Z, $P \wedge Z$ is equivalent to P.

The contribution of ex to the event specification of the event e is equivalent to the contributions that the event occurrences $ex_1 \ldots ex_i$ supply to the event specification.

From (1) to (3) follows that the application of the transformation $DownTg$ to an event occurrence of M_1 results into an equivalent dynamic model M_2. ■

Combined schema transformations

Based on the two transformations the schema transformation $DownSTg(G)$ is defined. The transformation shifts an event occurrence having the generalizing superstate G as source state and as target state to the states belonging to the generalizing superstate. The schema transformation $DownSTg$ can be regarded as meta-method of the meta-model. A corresponding algorithm looks like:

if G.Kind = generalizing superstate **then**
  EX := $self$.DownSg(G)
  for all $ex \in EX$ **do**
    ex.DownTg(G)
  end for
end if

First of all the event occurrence is copied for each state belonging to the generalizing superstate with the aid of the transformation $DonwSg$ and the source states adopted, resulting in a set of new event occurrences stored in the set EX. For each event occurrence in this set the transformation $DownTg$ is performed whereby the target states are adopted.

For the schema transformation $DownSTg$ only the transformations $DownSg$ and $DownTg$ are used. As both transformations are equivalence transformation and the equivalence of dynamic model is transitive (compare theorem 23, p. 23) the usage of $DownSTg$ on an event occurrence of M_1 results into an equivalent dynamic model M_2.

Let's consider the example in figure 4. In this example we want to shift the event occurrence a from the generalizing superstate Z to the states Z_1 and Z_2. First of all we apply the transformation $DownSg$. The event occurrence is copied and the source states are adopted. We receive the event occurrences a_1 and a_2 (compare figure 4(b)). Then the transformation $DownTg$ is performed to each event occurrence produced at first and change the target states. We get the event occurrences a_{11}, a_{12}, a_{21} and a_{22} (compare figure 4(c)).

4.3 Shifting event occurrences within a state aggregation

In a correct dynamic model event occurrences can be shifted within a state aggregation. Event occurrences can be shifted from a state of the state aggregation to its corresponding aggregating superstate and vice versa.

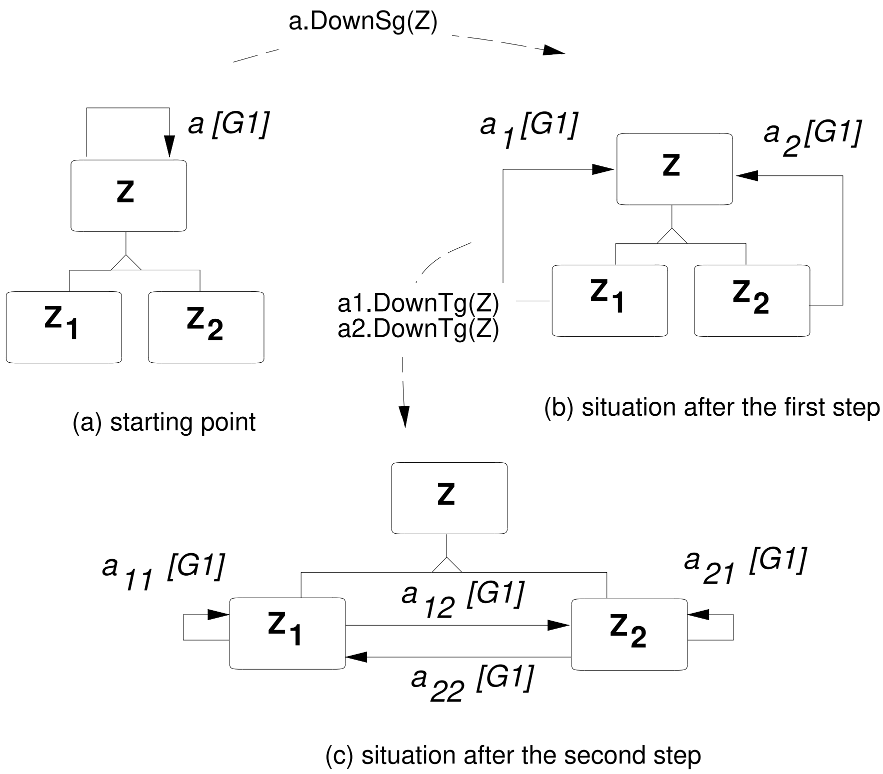

Figure 4: Shifting of event occurrences

4.3.1 Shifting event occurrences to an aggregating state

With the aid of these schema transformations we shift an event occurrence from a state of a state aggregation to its corresponding aggregating superstate. The transformation $UpSa$ shifts an event occurrence with a source state Z to the aggregating superstate of Z. The transformation $UpTa$ shifts an event occurrence with a target state Z to the aggregating superstate of Z.

The transformation $UpSa(Z)$ shifts an event occurrence with a source state Z to the aggregating superstate of Z. It can be regarded as a meta-method of the meta-model and looks like:

if Z.Belongs_to.Kind = aggregating superstate **then**
replace in $self$.Source_States Z through Z.Belongs_to
end if

The transformation replaces the source state Z of the event occurrence through the aggregating superstate of Z. According to the conditions of a correct dynamic model (compare definition 11, p. 16) the states belonging to an aggregating superstate must be generalizing superstates with equivalent ranges. Therefore, according to the definition 2, p. 11, the range of the aggregating superstate is equivalent to the ranges of its covering generalizing superstates.

Synchronizing event occurrences automatically are transformed to unsyncronizing one if afterwards the event occurrences have only one or less source and target states. Note that the attribute $Snyc$ of event occurrences is a computed one.

THEOREM: If a correct dynamic model M_1 is transformed by shifting an event occurrence using the schema transformation $UpSa(Z)$ into a dynamic model M_2 then $M_1 \equiv M_2$. (28)

PROOF: To prove this theorem it is sufficient to illustrate that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) The ranges of M_1 and M_2 are equivalent, as states remain unchanged by the transformation $UpSa$.

ad (2) Obvious the conditions of definition 11, p. 16 holds for M_2 too.

ad (3) The events remains unchanged when we shift event occurrences, therefore the conformity of the name, kind and attributes of events is automatically given. Furthermore the contributions of the changed event occurrence are equivalent in both dynamic models, as the source state is changed by an equivalent state.

Note that the state aggregation fulfills the condition of orthogonality. That means, that all states directly belonging to an aggregating superstate must be generalizing superstates with equivalent ranges (compare definition 6, p. 12). Therefore the range of the aggregating superstate is equivalent to the ranges of its covering generalizing superstates (compare definition 2, p. 11).

From (1) to (3) follows that the application of the schema transformation $UpSa$ to an event occurrence of M_1 results into an equivalent dynamic model M_2 ■

The transformation $UpTa(Z)$ shifts an event occurrence with a target state Z to the aggregating superstate of Z. It can be regarded as a meta-method of the meta-model and looks like:

if Z.Belongs_to.Kind = aggregating superstate **then**
replace in $self$.Target_States Z through Z.Belongs_to
end if

The target state Z is replaced by the aggregating superstate of Z (both states have equivalent ranges).

THEOREM: If a correct dynamic model M_1 is transformed by shifting an event occurrence using the schema transformation $UpTa(Z)$ into a dynamic model M_2 then $M_1 \equiv M_2$. (29)

PROOF: To prove this theorem it is sufficient to show that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) The ranges of M_1 and M_2 are equivalent, as states remain unchanged by the transformation $UpTa$.

ad (2) It is easy to the that M_2 is a correct dynamic model according to the conditions of definition 11, p. 16.

ad (3) Events remain unchanged by the transformation therefore the conformity of the name, kind and attributes of events is automatically given. Furthermore the contributions of the changed event occurrence are equivalent in both dynamic models, as source states or postconditions of the event occurrences are not changed.

From (1) to (3) follows that the application of the transformation $UpTa$ to an event occurrence of M_1 results into an equivalent dynamic model M_2 ■

Combined schema transformations

Based on the two transformations $UpSa$ and $UpTa$ we can determine a schema transformation $UpSTa(Z)$ shifting event occurrences having Z as source and target state to the aggregating superstate of Z. A corresponding algorithm:

if Z.Belongs_to.Kind = aggregating superstate **then**
$self$.UpSa(Z)
$self$.UpTa(Z)
end if

By applying $UpSTa$ to an event occurrence of M_1 an equivalent dynamic model M_2 is produced, as the equivalence transformations $UpSa$ and $UpTa$ are used only and the equivalence of dynamic models is transitive (compare theorem 23, p. 23).

4.3.2 Shifting event occurrences from an aggregating state

With the aid of these schema transformation we shift event occurrences of an aggregating superstate to a state belonging directly to the aggregating superstate. Again we distinguish between $DownSa$ and $DownTa$. Furthermore we have two more schema transformations $DownSas$ and $DownTas$ to change an unsynchronizing event occurrence to a synchronizing one when shifting it from the aggregating superstate.

The schema transformation $DownSa(Z)$ shifts an event occurrence from the aggregating superstate of Z to Z, if the aggregating superstate of Z is a source state of the event occurrence. It can be regarded as a meta-method of the meta-model and looks like:

if Z.Belongs_to = aggregating superstate **and**
    Z.Belongs_to $\in$ $self$.Source_States **then**
  replace Z.Belongs_to in $self$.Source_States by Z
end if

The transformation replaces in the source states of the event occurrence the aggregating superstate of Z by Z. Both states have, due to the orthogonality constraint, equivalent ranges.

THEOREM: If a correct dynamic model M_1 is transformed by shifting an event occurrence using the schema transformation $DownSa(Z)$ into a dynamic model M_2 then $M_1 \equiv M_2$. (30)

PROOF: To prove this theorem it is sufficient to show that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) None of the states are changed, the ranges of M_1 and M_2 are equivalent.

ad (2) M_2 is a correct dynamic model, in the source state of the shifted event occurrence a state is replaced by an other equivalent state.

ad (3) Events and event occurrences remain unchanged by the transformation. In event occurrences a source state is replaced by an equivalent state, which does not influence the preconditions of these event occurrences. Therefore the events of M_1 and M_2 are equivalent.

From (1) to (3) follows that the application of the transformation $DownSa(Z)$ to an event occurrence of M_1 results into an equivalent dynamic model M_2. ■

The schema transformation $DownTa(Z)$ shifts an event occurrence from the aggregating superstate of Z to Z, if the aggregating superstate of Z is a target states of the event occurrence. It can be regarded as a meta-method of the meta-model and looks like:

if Z.Belongs_to = aggregating superstate **and**
    Z.Belongs_to $\in$ $self$.Target_States **then**
  replace Z.Belongs_to in $self$.Target_States by Z
end if

The transformation replaces in the target states of the event occurrence the aggregating superstate of Z by Z. Both states have, due to the orthogonality constraint, equivalent ranges.

THEOREM: If a correct dynamic model M_1 is transformed by shifting an event occurrence using the schema transformation $DownTa(Z)$ into a dynamic model M_2 then $M_1 \equiv M_2$. (31)

PROOF: To prove this theorem it is sufficient to show that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) None of the states are changed, the ranges of M_1 and M_2 are equivalent.

ad (2) In the event occurrence a target state is replaced by an other equivalent state. The postcondition of the event occurrences implies the range of the new state too. M_2 is a correct dynamic model.

ad (3) Events and event occurrences remain unchanged by the transformation. In the event occurrence a target state is replaced by an equivalent state. Therefore the events of M_1 and M_2 are equivalent.

From (1) to (3) follows that the application of the transformation $DownTa(Z)$ to an event occurrence of M_1 results into an equivalent dynamic model M_2. ■

The schema transformation $DownSas(Z)$ adds the state Z as further source state to an event occurrence. Z must belong to an aggregating superstate A, which already is a source state of the event occurrence. $DownSas(Z)$ can be regarded as a meta-method of the meta-model and looks like:

if Z.Belongs_to = aggregating superstate **and**
    Z.Belongs_to $\in$ $self$.Source_States **then**
  $self$.Source_States := $self$.Source_States + Z
end if

Note that afterwards the event occurrence is a synchronizing one as it has several source states and the attribute $Sync$ of event occurrences is a computed one.

THEOREM: If a correct dynamic model M_1 is transformed by adding a new source state to the source states of an event occurrence using the schema transformation $DownSas(Z)$ into a dynamic model M_2 then $M_1 \equiv M_2$. (32)

PROOF: To prove this theorem it is sufficient to show that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) None of the states are changed, the ranges of M_1 and M_2 are equivalent.

ad (2) It is easy to see, that M_2 is a correct dynamic model.

ad (3) A new source state is added to the source states of the event occurrence. However, the aggregating superstate of Z is already a source state of the event occurrence. Due to the orthogonality constraint the ranges of Z and the aggregating superstate of Z are equivalent. Therefore the contributions of the shifted event occurrence to its event specification are equivalent in both dynamic models.

From (1) to (3) follows that the application of the transformation $DownSas(Z)$ to an event occurrence of M_1 results into an equivalent dynamic model M_2. ■

The schema transformation $DownTas(Z)$ adds the state Z as further target state to an event occurrence. Z must belong to an aggregating superstate A, which already is a target state of the event occurrence. $DownTas(Z)$ can be regarded as a meta-method of the meta-model and looks like:

if Z.Belongs_to = aggregating superstate **and**
 Z.Belongs_to $\in$ $self$.Target_States **then**
 $self$.Target_States := $self$.Target_States + Z
end if

Note that afterwards the event occurrence is a synchronizing one as it has several target states and the attribute $Sync$ of event occurrences is a computed one.

THEOREM: If a correct dynamic model M_1 is transformed by adding a new target state to the target states of an event occurrence using the schema transformation $DownTas(Z)$ into a dynamic model M_2 then $M_1 \equiv M_2$. (33)

PROOF: To prove this theorem it is sufficient to show that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) None of the states are changed, the ranges of M_1 and M_2 are equivalent.

ad (2) A new target state is added to the target states of the event occurrence. The event occurrence is changed to a synchronizing one. However, as the aggregating superstate of Z must already be a target state of the event occurrence and the states are equivalent, M_2 is a correct dynamic model.

ad (3) Events, pre- and the postconditions of event occurrences remain unchanged by the transformation. Therefore the events of M_1 and M_2 are equivalent.

From (1) to (3) follows that the application of the transformation $DownTas(Z)$ to an event occurrence of M_1 results into an equivalent dynamic model M_2. ■

Let's consider the example in figure 5 where a (incomplete) state aggregation is shown. Note, that the aggregating superstate Z_A and the states G_1 and G_2 (which are generalizing superstates) have equivalent ranges. In a first step the event occurrences a and b are shifted down from the aggregating superstate. Furthermore the event occurrence b is transformed to a synchronizing one. In a second step b is againg shifted from Z_A to G_1.

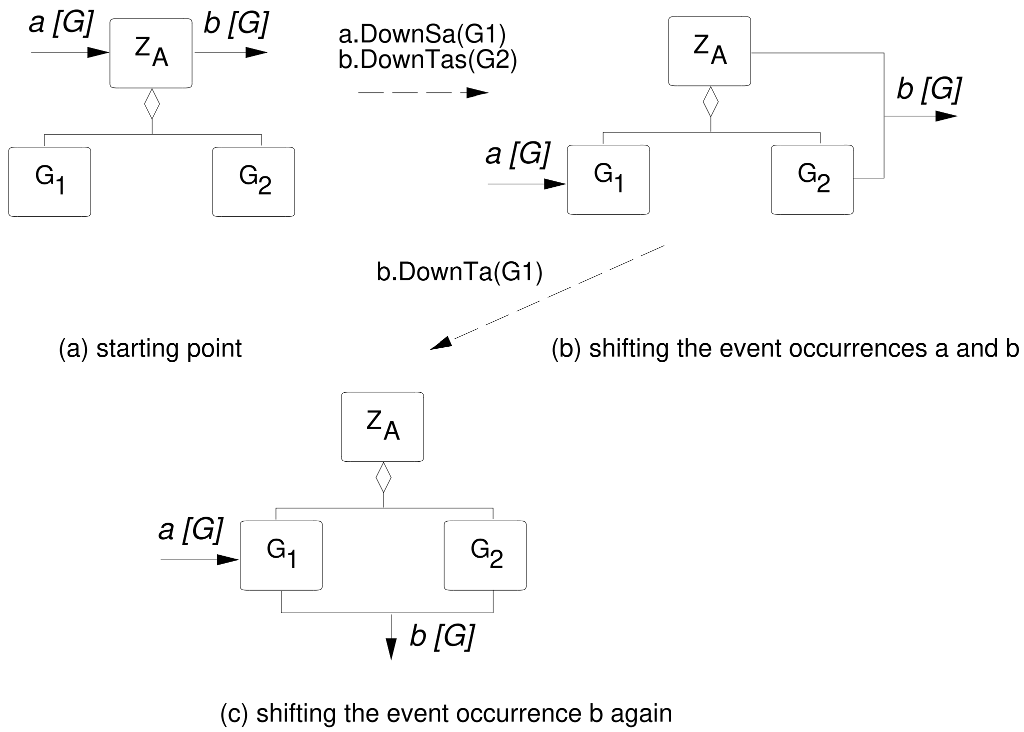

Figure 5: Shifting event occurrences within a state aggregation

4.4 The combination of event occurrences

In a correct dynamic model M two event occurrences of the same event can be combined, if they have equivalent pre- or postconditions.

The schema transformation $ComSe(e_1, e_2, e)$ combines the event occurrences e_1 and e_2 of the same event to e, if the preconditions of e_1 and e_2 are equivalent and both have the same source states and target states. The result of the transformation is the event occurrence e which is defined as:

$$e := \left\{ \begin{array}{rcl} e.Guard & := & e_1.PreC() \\ e.Postcondition & := & e_1.Postcondition \vee e_2.Postcondition \\ e.Source\_States & := & e_1.Source\_States \\ e.Target\_States & := & e_1.Target\_States \\ e.has\_Event & := & e_1.has\_Event \end{array} \right. \tag{34}$$

In $e.has\_Event.has$ e_1 and e_2 are replaced by e. Afterwards e_1 and e_2 are deleted.

THEOREM: If a correct dynamic model M_1 is transformed by the combination of two event occurrences e_1 and e_2 to an event occurrence e with the transformation $ComSe(e_1, e_2, e)$ then $M_1 \equiv M_2$. (35)

PROOF: To prove this theorem it is sufficient to show that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) The ranges of M_1 and M_2 are equivalent as states remain unchanged.

ad (2) Obviously M_2 is a correct dynamic model as M_1 was a correct one.

ad (3) We refer to the relation Ξ (definition 18(2)), which states that event occurrences with equivalent preconditions can be combinded through the disjunction of their postconditions without losing the equivalence.

From (1) to (3) follows that the application of the transformation $ComSe$ to event occurrences of M_1 results into an equivalent dynamic model M_2. ■

The schema transformation $ComTe(e_1, e_2, e)$ combines the event occurrences e_1 and e_2 of the same event to e, if the postconditions of e_1 and e_2 are equivalent and both have the same source states and target states. The result of the transformation is the event occurrence e which is defined as:

$$e := \begin{cases} e.Guard & := & e_1.PreC() \vee e_2.PreC() \\ e.Postcondition & := & e_1.Postcondition \\ e.Source\_States & := & e_1.Source\_States \\ e.Target\_States & := & e_1.Target\_States \\ e.has\_Event & := & e_1.has\_Event \end{cases} \tag{36}$$

In $e.has\_Event.has$ e_1 and e_2 are replaced by e. Afterwards e_1 and e_2 are deleted.

THEOREM: If a correct dynamic model M_1 is transformed by the combination of two event occurrences e_1 and e_2 to an event occurrence e with the transformation $ComTe(e_1, e_2, e)$ then $M_1 \equiv M_2$. (37)

PROOF: To prove this theorem it is sufficient to show that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) The ranges of M_1 and M_2 are equivalent as states remain unchanged.

ad (2) Obviously M_2 is a correct dynamic model as M_1 was a correct one.

ad (3) We refer to the relation Ξ (definition 18(1)), which states that event occurrences with equivalent postconditions can be combinded through the disjunction of their preconditions without losing the equivalence.

From (1) to (3) follows that the application of the schema transformation $ComTe$ to event occurrences of M_1 results into an equivalent dynamic model M_2. ■

4.5 The splitting of event occurrences

In a correct dynamic model M an event occurrence can be splitted into two event occurrences.

The schema transformation $SplitSe(e, P_1, P_2, e_1, e_2)$ splits the event occurrence e into the event occurrences e_1 and e_2. The parameters P_1 and P_2 are preconditions, their disjunction must be equivalent to the precondition of e. The result of the transformation are the event occurrences e_1 and e_2 which are defined as:

$$e_1 := \begin{cases} e_1.Guard & := & P_1 \\ e_1.Postcondition & := & e.Postcondition \\ e_1.Source\_States & := & e.Source\_States \\ e_1.Target\_States & := & e.Target\_States \\ e_1.has\_Event & := & e.has\_Event \end{cases} \tag{38}$$

$$e_2 := \begin{cases} e_2.Guard & := & P_2 \\ e_2.Postcondition & := & e.Postcondition \\ e_2.Source\_States & := & e.Source\_States \\ e_2.Target\_States & := & e.Target\_States \\ e_2.has\_Event & := & e.has\_Event \end{cases} \tag{39}$$

In $e.has\_Event.has$ e is replaced by e_1 and e_2. Afterwards e is deleted.

THEOREM: If a correct dynamic model M_1 is transformed by splitting an event occurrence e into two event occurrences e_1 and e_2 with the transformation $SplitSe(e, P_1, P_2, e_1, e_2)$ then $M_1 \equiv M_2$. (40)

Proof: To prove this theorem it is sufficient to show that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) The ranges of M_1 and M_2 are equivalent as states remain unchanged.

ad (2) Obviously M_2 is a correct dynamic model as M_1 was a correct one.

ad (3) We refer to the relation Ξ (definition 18(1)), which states that an event occurrence can be splitted into two event occurrences without losing the equivalence.

From (1) to (3) follows that the application of the transformation $SplitSe$ to event occurrences of M_1 results into an equivalent dynamic model M_2 ■

The schema transformation $SplitTe(e, P_1, P_2, e_1, e_2)$ splits the event occurrence e into the event occurrences e_1 and e_2. The parameters P_1 and P_2 are postconditions, their disjunction must be equivalent to the postcondition of e. The result of the transformation are the event occurrences e_1 and e_2 which are defined as:

$$e_1 := \begin{cases} e_1.Guard & := & e.Guard \\ e_1.Postcondition & := & P_1 \\ e_1.Source\_States & := & e.Source\_States \\ e_1.Target\_States & := & e.Target\_States \\ e_1.has\_Event & := & e.has\_Event \end{cases} \tag{41}$$

$$e_2 := \begin{cases} e_2.Guard & := & e.Guard \\ e_2.Postcondition & := & P_2 \\ e_2.Source\_States & := & e.Source\_States \\ e_2.Target\_States & := & e.Target\_States \\ e_2.has\_Event & := & e.has\_Event \end{cases} \tag{42}$$

In $e.has\_Event.has$ e is replaced by e_1 and e_2. Afterwards e is deleted.

Theorem: If a correct dynamic model M_1 is transformed by splitting an event occurrence e into two event occurrences e_1 and e_2 with the transformation $SplitTe(e, P_1, P_2, e_1, e_2)$ then $M_1 \equiv M_2$. (43)

Proof: To prove this theorem it is sufficient to show that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) The ranges of M_1 and M_2 are equivalent as states remain unchanged.

ad (2) Obviously M_2 is a correct dynamic model as M_1 was a correct one.

ad (3) We refer to the relation Ξ (definition 18(2)), which states that an event occurrence can be splitted into two event occurrences without losing the equivalence.

From (1) to (3) follows that the application of the transformation $SplitTe$ to event occurrences of M_1 results into an equivalent dynamic model M_2 ■

4.6 The combination of states

In a correct dynamic model M two *atomic* states (compare page 9) can be combined to one state. Both states must be either *elementary* states (compare definition 1, p. 10) or belonging to the same structured state. The transformation $Combine(Z_1, Z_2, Z)$ of two states Z_1 and Z_2 results in a state Z being defined as

$$Z := \left\{ \begin{array}{rcl} Z.Name & := & Z_1.Name + Z_2.Name \\ Z.Kind & := & atomic \\ Z.IS & := & Z_1.IS \wedge Z_2.IS \\ Z.FS & := & Z_1.FS \wedge Z_2.FS \\ Z.Belongs\_to & := & Z_1.Belongs\_to \\ Z.Condition & := & Z_1.Range() \vee Z_2.Range() \end{array} \right. \tag{44}$$

We need a name for the new state Z. Trying to be as simple as possible we choose the concatenation of the names of Z_1 and Z_2. We combine two atomic states, so Z is an atomic state too. If both combined states are initial states Z is an initial state (IS); it is a final state (FS) if both states are final states. The condition of the new states derives from the Z_1 and Z_2 ranges' disjunction. If Z_1 and Z_2 belongs to a structured state Z_S, Z_1 and Z_2 must be replaced by Z in $Z_S.Covers$.

In all event occurrences of M having Z_1 or Z_2 as source or target states, Z_1 and Z_2 must be replaced by Z. As Z_1 and Z_2 must be either elementary states or belonging to the same structured state (which must be a generalizing superstate)

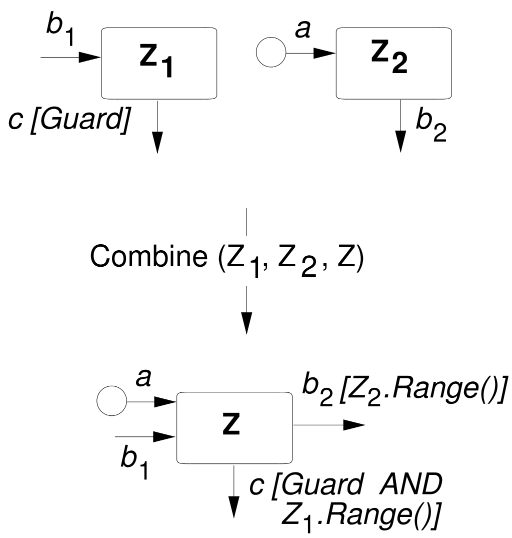

Figure 6: Combination of states

the states are disjoint. Therefore, it is not possible that both states are source or target states of an event occurrence.

However, the guard of an event occurrence must be replaced by its precondition if Z_1 or Z_2 is the source state of the event occurrence. The combination of states generates a new state with a "wider" range, nevertheless we want that event occurrences could only be applied to objects that comply with the original precondition of the event occurrences. A corresponding algorithm look like:

for all $ex \in M$.Event_Occurrences **with** $Z_1 \in ex$.Target_States or
    $Z_2 \in ex$.Target_States **do**
  replace Z_1 or Z_2 in ex.Target_States with Z
end for
for all $ex \in M$.Event_Occurrences **with** $Z_1 \in ex$.Source_States or
    $Z_2 \in ex$.Source_States **do**
  ex.Guard := ex.PreC()
  replace Z_1 or Z_2 in ex.Source_States with Z
end for
Z_1.delete()
Z_2.delete()

In figure 6 an example is illustrated. Z_1 and Z_2 are combined to Z. In event occurrences having Z_1 or Z_2 as source state we replace their guards by their precondition. The event occurrence c, e. g. gets the guard $c.PreC()$ that is, according to the definition, the conjunction of the source state's range of c and the guard of c. The source state of c was Z_1.

THEOREM: If a correct dynamic model M_1 is transformed by the combination of two states Z_1 and Z_2 to a state Z with the transformation $Combine(Z_1, Z_2, Z)$ then $M_1 \equiv M_2$. (45)

PROOF: To prove this theorem it is sufficient to show that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) The range of a dynamic model derives from the disjunction of the ranges of the model's elementary states (compare definition 12, p. 17).

If both states are *atomic* and *elementary* states then the range of $M_1 = B' \vee Z_1.Range() \vee Z_2.Range()$ whereby B' is the disjunction of all elementary states of M except Z_1 and Z_2. The dynamic model M_2 results from the combination of Z_1 and Z_2 to Z. All states of M_1 (particularly the elementary states) are also states of M_2, only Z_1 and Z_2 are replaced by Z. Therefore the range of $M_2 = B' \vee Z.Range()$. The range of Z is equivalent to the disjunction of the ranges of Z_1 and Z_2 (compare definition 44, p. 45) and therefore the ranges of M_1 and M_2 are equivalent.

If the states are not *elementary* they must belong to the same structured state Z_S which must be a generalizing superstate. It is easy to see that the ranges of Z_S in M_1 and M_2 are equivalent as we replace two states of Z_S with one equivalent state. The ranges of Z_S in both dynamic models are equivalent and therefore, the ranges of M_1 and M_2 are equivalent.

ad (2) The conditions of the definition 8, p. 12 are valid for M_2 ifZ_1 and Z_2 are *elementary* states because they are valid for M_1. It is easy to see, that M_2 is a correct dynamic model if Z_1 and Z_2 belong to the same state generalization. If both states belong indirectly to a state aggregation, the orthogonality condition holds for the combined state, as its range is equivalent to the disjunction of the original states.

The postconditions of the event occurrences must imply the ranges of their target states (according to definition 11, p. 16). It is easy to see, that each event occurrence implying the range of Z_1 or Z_2 implies the range of Z too, as, according to definition 44, p. 45, the range of Z results from the disjunction of the ranges of Z_1 and Z_2.

ad (3) Events stay unchanged by the combination of two states. Therefore the uniformity of their names, attributes and kind is valid. In addition to that we have to show that all event occurrences before and after the transformation supply equivalent contributions to the event specifications. This is trivially true for event occurrences having neither Z_1 nor Z_2 as source or target states. They are not touched by the transformation. In event occurrences which have Z_1 or Z_2 as target state, the target state is replaced by Z. However, the pre- and postcondition of those event occurrences remain unchanged. Therefore such event occurrences supply equivalent contributions to the event specification before and after the transformation. The guard of event occurrences having Z_1 or Z_2 as source state is changed. Those event occurrences must be analyzed, as the guard of an event occurrence influences its precondition.

Let ex be an event's e event occurrence of the dynamic model M_1 having Z_1 as source state. The contribution of ex to the event specification of e is:

$$\begin{aligned} before &= (Z_1 \wedge G \wedge R, P) \\ afterwards &= (Z \wedge (Z_1 \wedge G) \wedge R, P), \text{ comp. algorithm} \\ &= ((Z_1 \vee Z_2) \wedge (Z_1 \wedge G) \wedge R, P), \text{ comp. def. 44} \\ &= (Z_1 \wedge G \wedge R, P) \end{aligned}$$

The contributions of the event occurrence ex to the event specification of the event e before and after the schema transformation are equivalent. Therefore the event specifications of e in M_1 and e in M_2 are equivalent. The same holds, if Z_2 is the source state of ex.

From (1) to (3) follows, that the application of the transformation $Combine$ on states of M_1 results into an equivalent dynamic model M_2. ■

Let's take the event occurrence c of the dynamic model represented in figure 6, p. 46 and consider the third point of this proof. We check whether the contribution of the event occurrence before and after the combination of Z_1 and Z_2 are equivalent. The postcondition (in the proof the abbreviation P is used) can be ignored, it isn't changed by the schema transformation. Therefore we concentrate on the precondition of c without considering R (that is the conjunction of all source states of e except Z_1) because R remains constant. The precondition of c equals $Z_1.Range() \wedge c.Guard$ before the schema transformation and after the combination of Z_1 and Z_2 to Z the precondition of c equals $Z.Range() \wedge Z_1.Range() \wedge c.Guard$. The Preconditions of c before and after the schema transformation are equivalent because the range of Z results from the disjunction of the ranges of Z_1 and Z_2 (which must be disjoint according to definition 11, p. 16 as both states are elementary states).

4.7 The splitting of a state

In a correct dynamic model we can split an *atomic* state into two states. The schema transformation $Split(Z, B_1, B_2, Z_1, Z_2)$ needs two $\mathcal{TQL}++$ conditions as parameters, namely B_1 and B_2, their disjunction must be equivalent to the range of Z. Additionally B_1 and B_2 must be disjoint. If Z belongs to a state aggregation the splitting must not injure the orthogonality condition (compare definition 6, p. 12) . Otherwise the transformation is rejected. Furthermore if Z is not an *elementary* state, Z must be a source or target state of event occurrences. The result of $Split(Z, B_1, B_2, Z_1, Z_2)$ are two states Z_1 and Z_2 defined as:

$$Z_1 := \begin{cases} Z_1.Name & := & Z.Name + 1 \\ Z_1.Kind & := & atomic \\ Z_1.IS & := & Z.IS \\ Z_1.FS & := & Z.FS \\ Z_1.Belongs\_to & := & Z.Belongs\_to \\ Z_1.Condition & := & B_1 \end{cases} \tag{46}$$

$$Z_2 := \begin{cases} Z_2.Name & := & Z.Name + 2 \\ Z_2.Kind & := & atomic \\ Z_2.IS & := & Z.IS \\ Z_2.FS & := & Z.FS \\ Z_2.Belongs\_to & := & Z.Belongs\_to \\ Z_2.Condition & := & B_2 \end{cases} \tag{47}$$

If Z belongs to a structured state Z_S, Z must be replaced by Z_1 and Z_2 in $Z_S.Covers$.

All event occurrences having Z as source or target states must be duplicated. As Z may only be source or target state of event occurrences if Z is an atomic and elementary state Z in those event occurrences is the only source respectively target state. In the original event occurrences Z is replaced by Z_1, in the duplicated event occurrences Z is replaced by Z_2. The postcondition of an event occurrence having Z_1 as target state is replaced by $ex.Postcondition := ex.Postcondition \wedge Z_1.Range()$. Analogous $ex.Postcondition := ex.Postcondition \wedge Z_2.Range()$ is the postcondition of event occurrences having $Z2$ as target state. Afterwards Z can be deleted. A corresponding algorithm looks like:

for all $ex \in M$.Event_Occurrences **with** ex.Target_States = $\{Z\}$ **do**
  ex_1 := ex.shallow_copy()
  ex.has_Event.has := ex.has_Event.has + ex_1
  ex.Target_States := $\{Z_1\}$
  ex_1.Target_States := $\{Z_2\}$

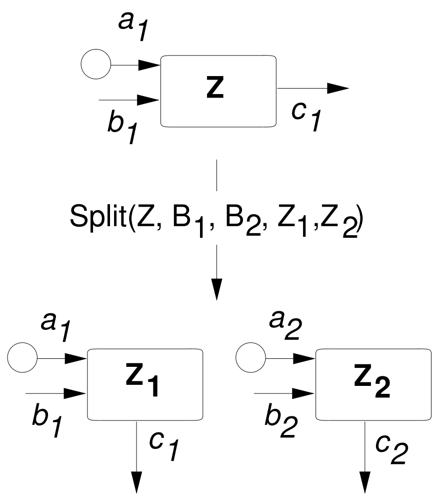

Figure 7: Splitting a state

ex.Postcondition := ex.Postcondition $\wedge$ Z_1.Range()
ex_1.Postcondition := ex_1.Postcondition $\wedge$ Z_2.Range()
end for
for all $ex \in M$.Event_Occurrences **with** ex.Source_States = $\{Z\}$ **do**
ex_1 := ex.shallow_copy()
ex.has_Event.has := ex.has_Event.has + ex_1
ex.Source_States := $\{Z_1\}$
ex_1.Source_States := $\{Z_2\}$
end for
Z.delete()

If afterwards event occurrences have pre- or postconditions resulting in $false$ the are deleted (compare definition 18, p. 19).

An example is illustrated in figure 7. The state Z is splitted in the states Z_1 and Z_2 and deleted afterwards. Note that Z is an atomic and elementary state and therefore may be source or target state of event occurrences.

THEOREM: If a correct dynamic model M_1 is transformed by splitting a state Z into two states Z_1 and Z_2 using the transformation $Split(Z, B_1, B_2, Z_1, Z_2)$ to a dynamic model M_2 then $M_1 \equiv M_2$. (48)

PROOF: To prove this theorem it is sufficient to show that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) The ranges of M_1 and M_2 are equivalent, if the range of Z is equivalent to the disjunction of the ranges of Z_1 and Z_2. This is obvious, as the ranges of Z_1 and Z_2 equal to B_1 and B_2 and their disjunction must be equivalent to the range of Z.

ad (2) The conditions of the definition 8, p. 12, are valid for M_2 as they where valid for M_1 too if Z does not belong to a state aggregation. Otherwise it must be guaranteed, that the splitting does not violate the orthogonality condition of the state aggregation. We prove this and reject the transformation in the case of a contradiction.

Event occurrences having Z as target state are copied and the target states are adapted. The postcondition of a copied event occurrence results in the conjunction of the original postcondition and the new target state's range. Obvious the postcondition of an event occurrence results in $false$ (according to definition 18, p. 19, they can be removed without loosing the equivalence) or implies the range of the new target state. Therefore, the condition of definition 10, p. 15, that the postcondition of an event occurrence must imply the range of its target states, holds too.

ad (3) Events remains unchanged by splitting up a state. Therefore the uniformity of their names, attributes and kind obvious is valid. Furthermore we must prove, that the event specifications are equivalent too.

This is obvious if Z is not an elementary state. In this case the transformation demands that Z is not a source or target state of any event occurrence and, therefore, event occurrences remain unchanged by the transformation.

If Z is an atomic and elementary state event occurrences having Z as source or target state are duplicated. Note that Z is the only source or target state of such event occurrences. We have to prove that the contributions of the copied event occurrences to the event specification is equivalent to the contribution of the origin event occurrence.

Let ex be an event's e event occurrence of the dynamic model having Z as source or target state without any restrictions. In the most general case, having ex Z as source and target state, ex is replaced by the event occurrences $ex_1 \ldots ex_4$. The contribution of ex to the event specification of e before the schema transformation:

$$before \quad = \quad (Z \wedge G \wedge R, P)$$

The contributions of the event occurrences $ex_1 \dots ex_4$ to the event specification of e after the schema transformation:

$$\begin{aligned}
afterwards &= ex_1 : (Z_1 \wedge G \wedge R, Z_1 \wedge P)\\
&\quad\; ex_2 : (Z_2 \wedge G \wedge R, Z_1 \wedge P)\\
&\quad\; ex_3 : (Z_2 \wedge G \wedge R, Z_2 \wedge P)\\
&\quad\; ex_4 : (Z_1 \wedge G \wedge R, Z_2 \wedge P)\\
&= ((Z_1 \vee Z_2) \wedge G \wedge R, Z_1 \wedge P), \text{comp. def. 18(1)}\\
&\quad\; ((Z_1 \vee Z_2) \wedge G \wedge R, Z_2 \wedge P), \text{comp. def. 18(1)}\\
&= ((Z_1 \vee Z_2) \wedge G \wedge R, (Z_1 \vee Z_2) \wedge P), \text{comp. def. 18(2)}\\
&= ((Z_1 \vee Z_2) \wedge G \wedge R, P), \text{comp. def. 11}\\
&= (Z \wedge G \wedge R, P)
\end{aligned}$$

The contributions to the event specification of e before and after the schema transformation are equivalent. All other cases follow immediately from this most general case.

From (1) to (3) follows that the application of the schema transformation $Split$ on a state of M_1 results into an equivalent dynamic model M_2. ■

The restriction, that in the case of a non elementary state, Z must not be a source or target state of event occurrences is not very extensive. In the previous sections we presented schema transformations to shift event occurrences within a state generalization or state aggregation. If we would like to split a non elementary state Z we are able to shift its event occurrences to the structured state Z belongs to and split Z afterwards.

4.8 The generalization of states

In a correct dynamic model we can generalize states. The schema transformation $Geng(Z_1 \dots Z_i, G)$ produces a state generalization with the generalizing superstate G. The states $Z_1 \dots Z_i$ must be elementary states or belonging to the same generalizing superstate. $Geng(Z_1 \dots Z_i, G)$ is defined as:

$$G := \begin{cases} G.Name &:= Z_1.Name + \dots + Z_i.Name\\ G.Kind &:= \text{generalizing superstate}\\ G.IS &:= \text{FALSE}\\ G.FS &:= \text{FALSE}\\ G.Belongs\_to &:= Z_1.Belongs\_to\\ G.Covers &:= \{Z_1, \dots, Z_i\} \end{cases} \tag{49}$$

Furthermore the generalizing superstate G has to be stored in the attribute $Belongs\_to$ of each state $Z_1 \dots Z_i$. The states $Z_1 \dots Z_i$ have to be removed in $Z_1.Belongs\_to.Covers$. A corresponding algorithm looks like:

if $\{Z_1 \dots Z_i\}$ belongs to the same generalizing superstate **or**
    are elementary states **then**
  for all $Z_k \in \{Z_1 \dots Z_i\}$ **do**
    Z_k.Belongs_to := G
  end for
  Z_1.Belongs_to.Covers := Z_1.Belongs_to.Covers - $\{Z_1 \dots Z_i\}$ + $\{G\}$
end if

THEOREM: If a correct dynamic model M_1 is transformed by generalizing states using the schema transformation $Geng(Z_1 \dots Z_i, G)$ into a dynamic model M_2 then $M_1 \equiv M_2$. (50)

PROOF: To prove this theorem it is sufficient to show that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) Let $Z_1 \dots Z_i$ be states without any restrictions which should be generalized with the aid of the transformation $Geng$. Let G be the generalizing superstate of the resulting state generalization. The range of G is defined as the disjunction of the ranges of all states belonging to G, that is $Z_1.Range() \vee \dots \vee Z_i.Range()$ (compare definition 2, p. 11). Obvious the ranges of M_1 and M_2 are equivalent.

ad (2) It is easy to see that M_2 complies with the conditions of the definitions 11, p. 16 as M_1 was a correct dynamic model.

ad (3) Events and event occurrences remain unchanged by the transformation $Geng$. Therefore the events of M_1 and M_2 are equivalent.

From (1) to (3) follows that the application of the schema transformation $Geng$ to states of M_1 results into an equivalent dynamic model M_2. ■

Let's consider the example in figure 8 where we put the states Z_1, Z_2 and Z_3 into a state generalization. We introduce the generalizing superstate G and link all states to G. All event occurrences of the states remain unchanged, they can be shifted to the new generalizing superstate with the aid of the corresponding schema transformations.

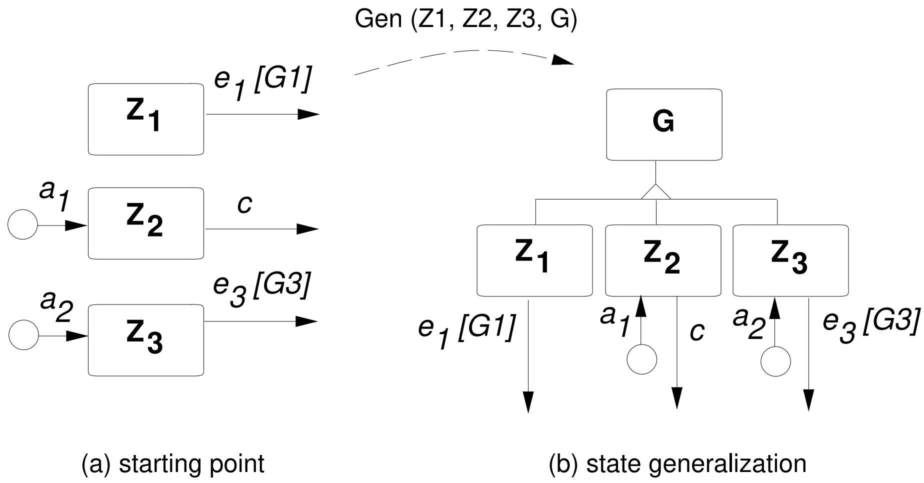

Figure 8: Building a state generalization

4.9 The decomposition of a state generalization

In a correct dynamic model we can decompose a state generalization with a generalizing superstate not belonging to an aggregating superstate and not being sorce or target state of event occurrences. $Decg(G)$ can be regarded as a meta-method of the meta-model. Applied to a generalizing superstate the state generalization is decomposed. The generalizing superstate is removed. A corresponding algorithm looks like:

if G not in source or target states of an event occurrence **and**
    G.Belongs_to $\neq$ aggregating superstate **then**
  for all $Z \in G$.Covers **do**
    Z.Belongs_to := G.Belongs_to
    G.Belongs_to.Covers := G.Belongs_to.Covers + Z
  end for
  G.Belongs_to.Covers := G.Belongs_to.Covers - G
  G.delete()
end if

THEOREM: If a correct dynamic model M_1 is transformed by decomposing an generalizing superstate using the schema transformation $Decg(G)$ into a dynamic model M_2 then $M_1 \equiv M_2$. (51)

PROOF: To prove this theorem it is sufficient to show that

(1) the ranges of M_1 and M_2 are equivalent,

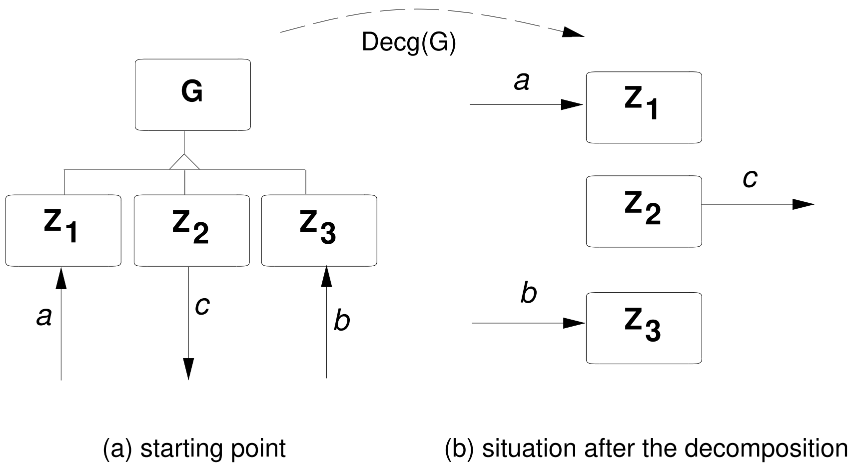

Figure 9: Decomposing a generalizing superstate

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) Let without any restrictions G be a generalizing superstate of M_1 covering the states $Z_1 \ldots Z_i$. G is deleted, afterwards $Z_1 \ldots Z_i$ either be elementary states or belongs to the same structured state. Obvious the ranges of M_1 and M_2 are equivalent.

ad (2) It is easy to see that M_2 complies with the conditions of the definitions 11, p. 16, as M_1 was a correct dynamic model.

ad (3) Events and event occurrences remain unchanged by the transformation $Decg$. Therefore the events of M_1 and M_2 are equivalent.

From (1) to (3) follows that the application of the transformation $Decg$ to a generalizing superstate of M_1 results into an equivalent dynamic model M_2. ■

Let's consider the example in figure 9 where we decompose the generalizing superstate G. Nothing happens with the events and on the event occurrences.

Combined schema transformations

At first the restriction of the transformation $Decg$ that the generalizing superstate must not occur in any source or target states of event occurrences seems to be very extensive. However, we have illustrated in section 4.2 that event occurrences within a state generalization can be shifted without losing the equivalence. Before decomposing a state generalization all event occurrences of the generalizing superstate can be shifted to the states covered by the generalizing superstate without losing the equivalence as the equivalence of dynamic models is transitive (compare theorem 23, p. 23). A corresponding algorithm looks like:

for all $ex \in M$.Event_Occurrences **with** $G \in ex$.Source_States and
        $G \in ex$.Target_States **do**
    ex.DownSTg(G)
end for
for all $ex \in M$.Event_Occurrences **with** $G \in ex$.Source_States **do**
    ex.DownSg(G)
end for
for all $ex \in M$.Event_Occurrences **with** $G \in ex$.Target_States **do**
    ex.DownTg(G)
end for
Decg(G)

4.10 The aggregation of states

In a dynamic model we can build a state aggregation based upon an *atomic* state. The transformation $Gena(Z, n, A)$ produces a state aggregation with the aggregating superstate A and n generalizing superstates G_i belonging to A, which itself covers one atomic state. A belongs to that structured state Z belongs to. The schema transformation is defined as:

$$A := \begin{cases} A.Name & := & Z.Name \\ A.Kind & := & \text{aggregating superstate} \\ A.IS & := & Z.IS \\ A.FS & := & Z.FS \\ A.Covers & := & \{G_1, \ldots, G_n\} \\ A.Belongs\_to & := & Z.Belongs\_to \end{cases} \tag{52}$$

$$G_i := \begin{cases} G_i.Name & := & Z.Name + G_i \\ G_i.Kind & := & \text{generalizing superstate} \\ G_i.IS & := & Z.IS \\ G_i.FS & := & Z.FS \\ G_i.Covers & := & \{Z_i\} \\ G_i.Belongs\_to & := & A \end{cases} \tag{53}$$

$$Z_i := \begin{cases} Z_i.Name & := & Z.Name + i \\ Z_i.Kind & := & \text{atomic state} \\ Z_i.IS & := & Z.IS \\ Z_i.FS & := & Z.FS \\ Z_i.Belongs\_to & := & G_i \\ Z_i.Condition & := & Z.Range() \end{cases} \tag{54}$$

In all event occurrences having Z as source or target state Z must be replaced by A, afterwards Z is deleted. A corresponding algorithm looks like:

for all $ex \in M$.Event_Occurrences **with** $Z \in ex$.Source_States **do**
  replace Z in ex.Source_States with A
end for
for all $ex \in M$.Event_Occurrences **with** $Z \in ex$.Target_States **do**
  replace Z in ex.Target_States with A
end for
Z.Belongs_to.Covers := Z.Belongs_to.Covers - Z + A
Z.delete()

THEOREM: If a correct dynamic model M_1 is transformed by constructing a state aggregation based on an atomic state with the aid of the schema transformation $Gena(Z, n, A)$ into a dynamic model M_2 then $M_1 \equiv M_2$. (55)

PROOF: To prove this theorem it is sufficient to show that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) Obvious the ranges of M_1 and M_2 are equivalent, Z is replaced by a state aggregation with an equivalent range.

ad (2) The constructed state aggregation fulfills the orthogonal condition, as all states are equivalent (compare definition 6, p. 12). In event occurrences with Z as target state Z is replaced by an equivalent state. Therefore the postconditions of these event occurrences imply the range of their new target state. M_2 complies with the conditions of the definition 11, p. 16, of correct dynamic models.

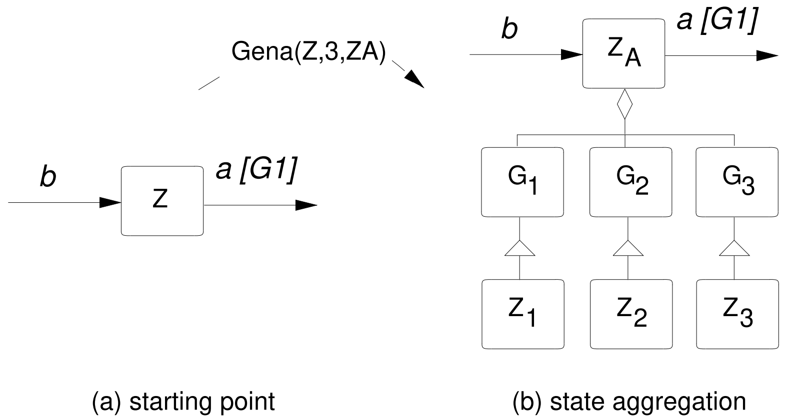

Figure 10: Building a state aggregation

ad (3) Events remain unchanged by the transformation $Gena$. In event occurrences having Z as source or target state Z is replaced by an equivalent state. Therefore the events of M_1 and M_2 are equivalent.

From (1) to (3) follows that the application of the schema transformation $Gena$ to an atomic state of M_1 results into an equivalent dynamic model M_2. ■

Let's consider the example in figure 10 where we build a state aggregation on the base of the atomic state Z. We introduce the aggregating superstate Z_A with three generalizing superstates $G_1 \dots G_3$, which belongs to Z_A. Each generalizing superstate consists of one further atomic state (the states $Z_1 \dots Z_3$).

4.11 The decomposition of a state aggregation

In a dynamic model we can decompose a state aggregation consisting of n generalizing superstate where each of these states covers only one further atomic state. Furthermore none of the states belonging to the state aggregation (except the aggregation superstate) must be source or target state of an event occurrence. The schema transformation $Deca(A, Z)$ decomposes a state aggregation with aggregating superstate A resulting in the atomic state Z which is defined as:

$$Z := \begin{cases} Z.Name & := & A.Name \\ Z.Kind & := & \text{atomic state} \\ Z.IS & := & A.IS \\ Z.FS & := & A.FS \\ Z.Condition & := & A.Range() \\ Z.Belongs\_to & := & A.Belongs\_to \end{cases} \tag{56}$$

In all event occurrences, having A as source or target state A is replaced by Z. Afterwards all states of the state aggregation are deleted. A corresponding algorithm looks like:

for all $ex \in M$.Event_Occurrences **with** $A \in ex$.Source_States **do**
  replace A in ex.Source_States with Z
end for
for all $ex \in M$.Event_Occurrences **with** $A \in ex$.Target_States **do**
  replace Z in ex.Target_States with Z
end for
A.Belongs_to.Covers := A.Belongs_to.Covers - A + Z
delete A and all states belonging to the state aggregation

THEOREM: If a correct dynamic model M_1 is transformed by decomposing a state aggregation using the schema transformation $Deca(A, Z)$ into a dynamic model M_2 then $M_1 \equiv M_2$. (57)

PROOF: To prove this theorem it is sufficient to show that

(1) the ranges of M_1 and M_2 are equivalent,

(2) M_2 is a correct dynamic model and

(3) all event occurrences before and after the transformation supply an equivalent contribution to the event specification.

ad (1) Obvious the ranges of M_1 and M_2 are equivalent, the aggregating superstate A and all other states belonging to the state aggregation are replaced by an equivalent atomic state Z.

ad (2) It is easy to see that M_2 is a correct dynamic model, an aggregating superstate is replaced by another atomic equivalent state (compare definitions 11, p. 16).

ad (3) Events remain unchanged by the transformation $Deca$. In event occurrences having A as source or target state A is replaced by an equivalent state. Therefore the events of M_1 and M_2 are equivalent.

From (1) to (3) follows that the application of the transformation $Deca$ to an aggregating superstate of M_1 results into an equivalent dynamic model M_2. ■

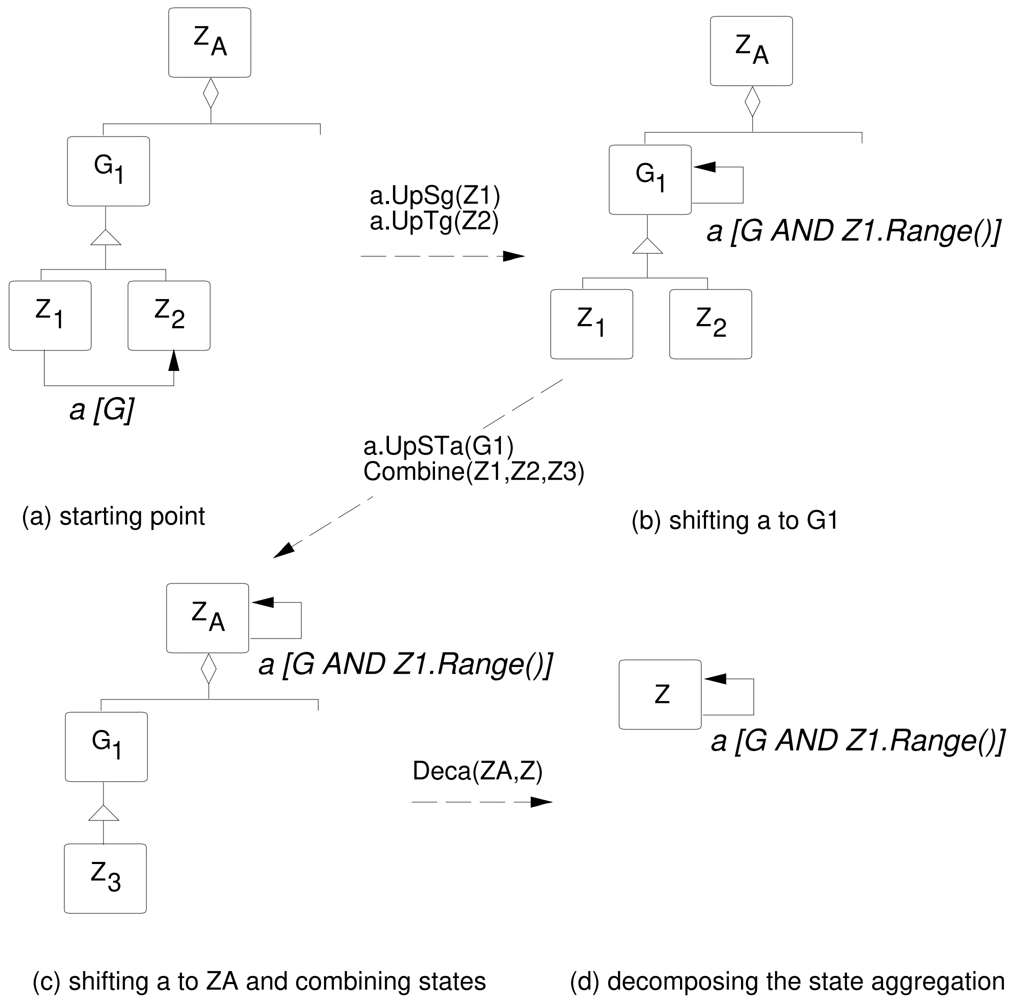

Figure 11: Combined schema transformations

Combined schema transformations

Decomposing a state aggregation is allowed only if the state aggragation consits of generalizing superstates each covers only one atomic state. Furthermore none of the states of the aggregation except the aggregating superstate is a source or target state of an event occurrence. However, these not extensive restrictions, as we are able to shift event occurrences up to generalizing and aggregating superstates, decompose state aggregations and generalisations as well as combine states.

Let's consider the example in figure 11 where we would like to decompose the state aggregation. However, there are states within the aggregation which are source and target states of an event occurrence. Furthermore the generalizing superstate G_1

covers more than one state.

First we shift the event occurrence to the generalizing superstate of Z_1 and Z_2 using the schema transformations $UpSg$ and $UpTg$ (note that the guard of the event occurrence changes when applying the transformations). Second we shift the event occurrence to the aggregating superstate using the schema transformation $UpSTa$ and combine Z_1 and Z_2 to Z_3. Afterwards we decompose the state aggregation using the transformation $Deca$ (we suppose that the other components of the state aggregation fulfills the necessary conditions).

The dynamic models shown in figure 11 are equivalent as we use equivalence transformations only, and, according to theorem 23, p. 23, the equivalence of dynamic models is transitive.

4.12 Deleting and combining event occurrences

In a correct dynamic model we would like to be able to delete and to combine event occurrences for instance when we decompose a states generalization or shift event occurrences within state hierarchies.

The deletion of event occurrences

With the aid of the schema transformation $DelEx$ we delete event occurrences whose preconditions or postconditions result in $false$. However, we don't want to remove incorrect event occurrences by this transformation but we want to delete "deceased" event occurrences after applying a schema transformation.

The transformation $DelEx$ can be regarded as meta-method of the meta-model and deletes event occurrences whose pre- or postconditions result in $false$. A corresponding algorithm looks like:

for all $ex \in self$.Event_Occurrences **with** ex.PreC() = $false$ or
  ex.Postcondition = $false$ **do**
 ex.has_Event.has := ex.has_Event.has - ex
 ex.delete()
end for

THEOREM: If a correct dynamic model M_1 is transformed with the aid of the schema transformation $DelEx$ into a dynamic model M_2 then $M_1 \equiv M_2$. (58)

PROOF: We refer to the definition 18, p. 19, which states that event occurrences can be deleted without loosing the equivalence if their pre- and/or postconditions result in $false$. States remain unchanged by the transformation. ■

The combination of event occurrences

With the aid of the schema transformation $ComEx$ we combine event occurrences of the same event having equal source states and equivalent postconditions respectively having equivalent preconditions and equal target states (compare transformations $ComSe$ and $ComTe$, page 41ff). A corresponding algorithm looks like:

repeat
  EX := $self$.Event_Occurrences
  for all $ex_1 \in self$.Event_Occurrences, $ex_2 \in self$.Event_Occurrences **with**
    $ex_1 \neq ex_2$ and ex_1.has_Event = ex_2.has_Event and
    ex_1.Target_States = ex_2.Target_States and
    ex_1.Source_States = ex_2.Source_States and
    ex_1.PreC() $\equiv$ ex_2.PreC() **do**
   $ComSe(e_1, e_2, e)$
  end for
  for all $ex_1 \in self$.Event_Occurrences, $ex_2 \in self$.Event_Occurrences **with**
    $ex_1 \neq ex_2$ and ex_1.has_Event = ex_2.has_Event and
    ex_1.Source_States = ex_2.Source_States and
    ex_1.Source_States = ex_2.Source_States and
    ex_1.Postcondition $\equiv$ ex_2.Postcondition **do**
   $ComTe(e_1, e_2, e)$
  end for
until EX = $self$.Event_Occurrences

The transformation $ComEx$ combines event occurrences as long as possible according to the relation Ξ (compare definition 18, p. 19) with the aid of the transformations $ComSe$ and $ComTe$. As both transformations are equivalence transformations $ComEx$ is an equivalence transformation too.

Based on the schema transformations $DelEx$ and $ComEx$ we define the transformation $Clean$ which deletes and combines event occurrences. The transformation can be regarded as meta-method of the meta-model. A corresponding algorithm looks like:

$self$.DelEx
$self$.ComEx

$Clean$ preserves the equivalence of the dynamic model as we only use the equivalence transformations $DelEx$ and $ComEx$ and the equivalence of dynamic models is transitive (compare theorem 23, p. 23). For example $Clean$ can be applied after states were generalized and event occurrences were shifted to the generalizing superstate if these event occurrences should be combined.

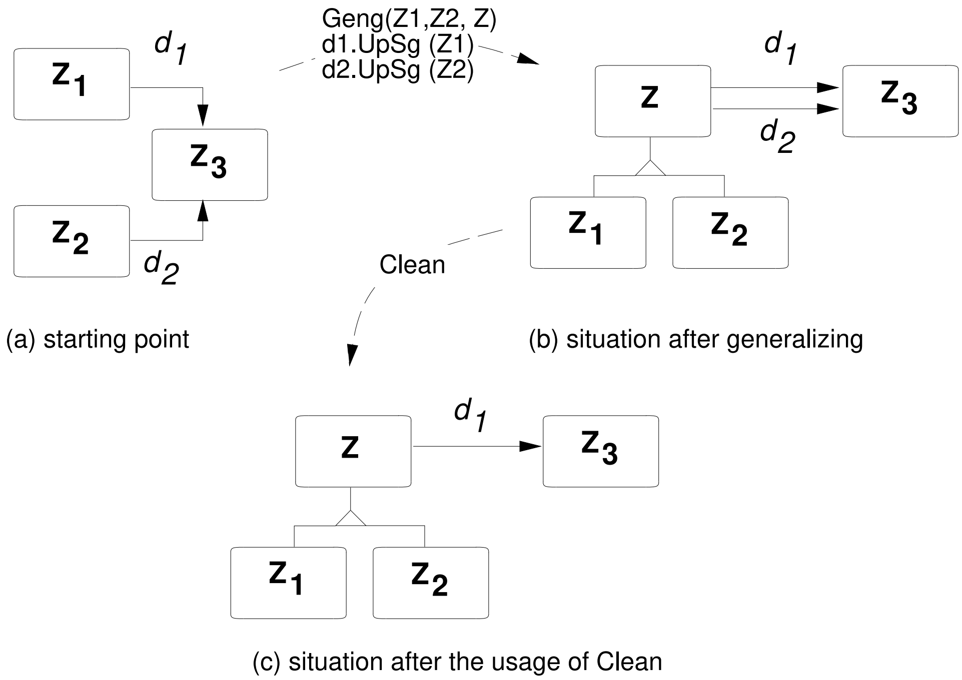

Figure 12: Usage of Clean

Such a situation is illustrated in figure 12. After generalizing Z_1 and Z_2 we shift the event occurrences d_1 and d_2 with the schema transformation $UpSg$ to the generalizing superstate Z. Supposing that the postconditions of the event occurrences are equivalent we can combine them with the transformation $Clean$ to one event occurrence.

5 Inverse schema transformations

In this section we will prove that each basic schema transformation has an inverse transformation, which however may be a complex transformation. The application of a transformation and its inverse on a dynamic model M_1 results into M_1. To prove this we have to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

For the following explanations we assume that M_1 consists only of states whose ranges are not $false$ and that non of the event occurrences has pre- or postconditions resulting to $false$.

5.1 The inverse transformation of $UpSg$

The transformation $UpSg(Z)$ shifts an event occurrence having the state Z as source state to the generalizing superstate of Z (see page 25). The inverse transformation is $DownSg(Z.Belongs\_to)$ (see page 29).

THEOREM: The inverse transformation of $ex.UpSg(Z)$ is $ex.DownSg(Z.Belongs\_to)$. (59)

PROOF: To prove that $DownSg(Z.Belongs\_to)$ is the inverse transformation of $UpSg(Z)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent as only equivalence transformations are used.
ad (2) States remain unchanged by the transformation and its inverse.
ad (3) Events remain unchanged by the transformation and its inverse.
ad (4) The transformation replaces Z by the generalizing superstate of Z in the source states of the event occurrence. Furthermore the guard of the event occurrence is replaced by its precondition. Let $Z_1 \ldots Z_n$ be the states covered by the generalizing superstate of Z. The inverse transformation produces n event occurrences (for each state covered by the generalizing superstate one event occurrenc) and changes the source states of the copied event occurrences. The event occurrence ex is deleted.

The precondition of one copied event occurrences looks like $Z_i.Range \wedge Z.Range() \wedge ex.Guard \wedge R$ where $Z_i \in Z_1 \ldots Z_n$. R is the conjunction of the ranges of all other source states of the event occurrence or $true$, if there is only one source state. However, in a correct dynamic model the ranges of $Z_1 \ldots Z_n$ must be disjoint. The preconditions of the copied event occurrences results in $false$ except $Z_i = Z$. In other words $n-1$ of the n copied event occurrences have preconditions resulting in $false$ and are deleted by the transformation $Clean$.

From (1) to (4) follows that the transformation $DownSg(Z.Belongs\_to)$ is the inverse transformation of $UpSg(Z)$. ■

5.2 The inverse transformation of $UpTg$

The transformation $UpTg(Z)$ shifts an event occurrence having the state Z as target state to the generalizing superstate of Z (see page 26). The inverse transformation is $DownTg(Z.Belongs\_to)$ (see page 30).

THEOREM: The inverse transformation of $ex.UpTg(Z)$ is $ex.DownTg(Z.Belongs\_to)$. (60)

PROOF: To prove that $DownTg(Z.Belongs\_to)$ is the inverse transformation of $UpTg(Z)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent as ony equivalence transformations are used.

ad (2) States remain unchanged by the transformation and its inverse.

ad (3) Events remain unchanged by the transformation and its inverse.

ad (4) The transformation replaces Z by the generalizing superstate of Z in the target states of the event occurrence. Let $Z_1 \ldots Z_n$ the states covered by the generalizing superstate of Z. The inverse transformation produces n event occurrences (for each state covered by the generalizing superstate one event occurrenc) and changes the target states of the copied event occurrences. Furthermore the postcondition of the copied event occurrences are replaced by the conjunction of the original postcondition and the range of its new target state. The event occurrence ex is deleted.

The postcondition of the copied event occurrences looks like $Z_i.Range \wedge ex.Postcondition$ where $Z_i \in Z_1 \ldots Z_n$. However, in a correct dynamic model the ranges of $Z_1 \ldots Z_n$ must be disjoint. Furthermore the postcondition of ex must imply the range of Z. Therefore, the postconditions of the copied event occurrences results in $false$ except $Z_i = Z$. In other words $n-1$ of the n copied event occurrences have postconditions resulting in $false$ and are deleted by the transformation $Clean$.

From (1) to (4) follows that the transformation $DownTg(Z.Belongs\_to)$ is the inverse transformation of $UpTg(Z)$. ■

5.3 The inverse transformation of $DownSg$

The transformation $DownSg(Z)$ shifts an event occurrence having the generalizing superstate Z as source state to the states belonging to Z (see page 29). For each state covered by Z the event occurrence is copied, the source states are adopted. Some of the copied event occurrences may have preconditions resulting in $false$. These event occurrences are deleted by the transformation $Clean$. If Ex is the set of the copied event occurrences which remain than the inverse transformation (lets call it $DownSg^{-1}(Z)$) is defined as:

for all $e_i \in Ex$ with $Z_j \in e_i.Source\_States$ **and** $Z_j \in Z.Covers$ **do**
  e_i.UpSg(Z_j)
end for

THEOREM: The inverse transformation of $ex.DownSg(Z)$ is $DownSg^{-1}(Z)$. (61)

PROOF: To prove that $DownSg^{-1}(Z)$ is the inverse transformation of $DownSg(Z)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent as ony equivalence transformations are used.
ad (2) States remain unchanged by the transformation and its inverse.

ad (3) Events remain unchanged by the transformation and its inverse.

ad (4) The transformation copies the event occurrence for each state that Z covers and adopts the source states. The original event occurrence is deleted afterwards. Some of the copied event occurrences may have preconditions resulting in $false$ and are deleted by the transformation $Clean$. The remaining event occurrences are collected in a set Ex. As the original precondition of the shifted event occurrence was not $false$ and the range of the generalizing superstate is defined as the disjunction of ranges of its covering state Ex is not empty.

Each copied event occurrence from Ex is shifted back to the generalizing superstate using the transformation $UpSg$. By that the guards of these event occurrences are replaced by its precondition. Afterwards these event occurrences are combined to one event occurrence using the transformation $Clean$, as they have the same event, the same source- and target states and equivalent postconditions by the disjunction of their precondions. However, as all involved transformations are equivalence transformations the precondition of the combined event occurrence must be equivalent to the precondition of the original event occurrence.

From (1) to (4) follows that the transformation $DownSg^{-1}(Z)$ is the inverse transformation of $DownSg(Z)$. ■

5.4 The inverse transformation of $DownTg$

The transformation $DownTg(Z)$ shifts an event occurrence having the generalizing superstate Z as target state to the states belonging to Z (see page 30). For each state covered by Z the event occurrence is copied, the target states and postconditions are adopted. Some of the copied event occurrences may have postconditions resulting in $false$. These event occurrences are deleted by the transformation $Clean$. If Ex is the set of the copied event occurrences which remain than the inverse transformation (lets call it $DownTg^{-1}(Z)$) is defined as:

for all $e_i \in Ex$ with $Z_j \in e_i.Target\_States$ **and** $Z_j \in Z.Covers$ **do**
  e_i.UpTg(Z_j)
end for

THEOREM: The inverse transformation of $ex.DownTg(Z)$ is $DownTg^{-1}(Z)$. (62)

PROOF: To prove that $DownTg^{-1}(Z)$ is the inverse transformation of $DownTg(Z)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent as only equivalence transformation are used.
ad (2) States remain unchanged by the transformation and its inverse.
ad (3) Events remain unchanged by the transformation and its inverse.
ad (4) The transformation copies the event occurrence for each state that Z covers and adopts the source states and postcondition. The original event occurrence is deleted afterwards. Some of the copied event occurrences may have postconditions resulting in $false$ and are deleted by the transformation $Clean$. The remaining event occurrences are collected in a set Ex. As the original postcondition of the shifted event occurrence was not $false$ but implies the range of Z and the range of the generalizing superstate is defined of the disjunction of ranges of its covering states Ex is not empty.

Each copied event occurrence from Ex is shifted back to the generalizing superstate using the transformation $UpSg$. Afterwards these event occurrences are combined to one event occurrence using the transformation $Clean$, as they have the same event, the same soure- and target states and equivalent preconditions by the disjunction of their postcondions. However, as all involved transformations are equivalence transformations the postcondition of the combined event occurrence must be equivalent to the postcondition of the original event occurrence.

From (1) to (4) follows that the transformation $DownTg^{-1}(Z)$ is the inverse transformation of $DownTg(Z)$. ■

5.5 The inverse transformation of $UpSa$

The transformation $UpSa(Z)$ shifts an event occurrence with Z as source state to the aggregating superstate of Z (see page 37). If the aggregating superstate of Z already is a source state of the event occurrence the inverse transformation is $DownSas(Z)$ (see page 39). Otherwise the inverse transformation is $DownSa(Z)$ (see page 37).

THEOREM: The inverse transformation of $ex.UpSa(Z)$ is $ex.DownSas(Z)$ if the aggregating superstate of Z already is a source state of the event occurrence ex. Otherwise the inverse transformation is $ex.DownSa(Z)$. (63)

PROOF: To prove that $DownSas(Z)$ and $DownSa(Z)$ are the inverse transformations of $UpSa(Z)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent as onle equivalence transformations are used.
ad (2) States remain unchanged by the transformation and its inverse.
ad (3) Events remain unchanged by the transformation and its inverse.
ad (4)
(a) The inverse transformation is $ex.DownSas(Z)$ if the aggregating superstate of Z already is a source state of the event occurrence ex. The aggregating superstate and Z have equivalent ranges. The transformation replaces Z in the source states by the aggregating superstate of Z. However, as the source states of an event occurrence is a set and the aggregating superstate of Z already is a member of this set, the transformation “removes” Z from the source states. The inverse adds Z as further source state of the event occurrence.

The transformation may change the event occurrence to a non-synchronizing one, if afterwards ex has only one or less source and target states (note that the attribute $Sync$ is a computed one). The inverse adds a further source state, the event occurrence is a synchronizing one again.

(b) The inverse transformation is $ex.DownSa(Z)$ if the aggregating superstate of Z is not a source state of the event occurrence ex. The aggregating superstate and Z have equivalent ranges. The transformation replaces Z by the aggregating superstate of Z in the source states of the event occurrence. The inverse changes the aggregating superstate of Z in the source states of the event occurrence back to Z.

From (1) to (4) follows that the transformations $DownSas(Z)$ and $DownSa(Z)$ are the inverse transformations of $UpSa(Z)$. ■

5.6 The inverse transformation of $UpTa$

The transformation $UpTa(Z)$ shifts an event occurrence with Z as target state to the aggregating superstate of Z (see page 38). If the aggregating superstate of Z already is a target state of the event occurrence the inverse transformation is $DownTas(Z)$ (see page 39). Otherwise the inverse transformation is $DownTa(Z)$ (see page 38).

THEOREM: The inverse transformation of $ex.UpTa(Z)$ is $ex.DownTas(Z)$ if the aggregating superstate of Z already is a target state of the event occurrence ex. Otherwise the inverse transformation is $ex.DownTa(Z)$. (64)

PROOF: To prove that $DownTas(Z)$ and $DownTa(Z)$ are the inverse transformations of $UpTa(Z)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent as only equivalence transformations are used.
ad (2) States remain unchanged by the transformation and its inverse.
ad (3) Events remain unchanged by the transformation and its inverse.
ad (4)
(a) The inverse transformation is $ex.DownTas(Z)$ if the aggregating superstate of Z already is a target state of the event occurrence ex. The aggregating superstate and Z have equivalent ranges. The transformation replaces Z in the target states by the aggregating superstate of Z. However, as the target states of an event occurrence is a set and the aggregating superstate of Z already is a member of this set, the transformation “removes” Z from the target states. The inverse adds Z as further target state of the event occurrence.

The transformation may change the event occurrence to a non-synchronizing one, if afterwards ex has only one or less source and target states (note that the attribute $Sync$ is a computed one). The inverse adds a further target state, the event occurrence is a synchronizing one again.

(b) The inverse transformation is $ex.DownTa(Z)$ if the aggregating superstate of Z is not a target state of the event occurrence ex. The aggregating superstate and Z have equivalent ranges. The transformation replaces Z by the aggregating superstate of Z in the target states of the event occurrence. The inverse changes the aggregating superstate of Z in the target states of the event occurrence back to Z.

From (1) to (4) follows that the transformations $DownTas(Z)$ and $DownTa(Z)$ are the inverse transformations of $UpTa(Z)$. ■

5.7 The inverse transformation of $DownSa$

The transformation $DownSa(Z)$ shifts an event occurrence from the aggregating superstate of Z to Z, if the aggregating superstate of Z is a source state of the event occurrence (see page 37). The inverse transformation is $UpSa(Z)$ (see page 34).

THEOREM: The inverse transformation of $ex.DownSa(Z)$ is $ex.UpSa(Z)$. (65)

PROOF: To prove that $UpSa(Z)$ is the inverse transformation of$DownSa(Z)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent as only equivalence transformations are used.
ad (2) States remain unchanged by the transformation and its inverse
ad (3) Events remain unchanged by the transformation and its inverse.
ad (4) The states Z and the aggregating superstate of Z have equivalent ranges. The transformation replaces the aggregating superstate of Z in the source state of the event occurrence ex by Z. The inverse replaces Z in the source state of the event occurrence ex by the aggregating superstate of Z.

From (1) to (4) follows that the transformation $UpSa(Z)$ is the inverse transformation of $DownSa(Z)$. ■

5.8 The inverse transformation of $DownTa$

The transformation $DownTa(Z)$ shifts an event occurrence from the aggregating superstate of Z to Z, if the aggregating superstate of Z is a target state of the event occurrence (see page 38). The inverse transformation is $UpTa(Z)$ (see page 35).

THEOREM: The inverse transformation of $ex.DownTa(Z)$ is $ex.UpTa(Z)$. (66)

PROOF: To prove that $UpTa(Z)$ is the inverse transformation of $DownTa(Z)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent as only equivalence transformations are used.
ad (2) States remain unchanged by the transformation and its inverse
ad (3) Events remain unchanged by the transformation and its inverse.
ad (4) The states Z and the aggregating superstate of Z have equivalent ranges. The transformation replaces the aggregating superstate of Z in the target state of the event occurrence ex by Z. The inverse replaces Z in the target state of the event occurrence ex by the aggregating superstate of Z.

From (1) to (4) follows that the transformation $UpTa(Z)$ is the inverse transformation of $DownTa(Z)$. ■

5.9 The inverse transformation of $DownSas$

The transformation $DownSas(Z)$ adds Z as a new source state to an event occurrence, if the aggregating superstate of Z already is a source state of the event occurrence. (see page 39). The inverse transformation is $UpSa(Z)$ (see page 34).

THEOREM: The inverse transformation of $ex.DownSas(Z)$ is $ex.UpSa(Z)$. (67)

PROOF: To prove that $UpSa(Z)$ is the inverse transformation of $DownSas(Z)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent only equivalence transformations are used.
ad (2) States remain unchanged by the transformation and its inverse
ad (3) Events remain unchanged by the transformation and its inverse.

ad (4) The aggregating superstate of Z and Z have equivalent ranges. Z is added as further source state of the event occurrence ex. The inverse replaces Z in the source states of the event occurrence ex by the aggregating superstate of Z. However, the aggregating superstate already is a source state of the event occurrence and and the source states of an event occurrence is a set this replacement removes Z from the source states of ex.

If ex is a non-synchronizing event occurrence the transformation changes it to a synchronizing one (note that the attribute $Sync$ of an event occurrence is a computed one) as afterwards ex has more than one source states. By applying the inverse ex is transformed back to a non-synchronizing event occurrence as afterwards ex has only one source state.

From (1) to (4) follows that the transformation $UpSa(Z)$ is the inverse transformation of $DownSas(Z)$. ■

5.10 The inverse transformation of $DownTas$

The transformation $DownTas(Z)$ adds Z as a new target state to an event occurrence, if the aggregating superstate of Z already is a target state of the event occurrence. (see page 39). The inverse transformation is $UpTa(Z)$ (see page 35).

THEOREM: The inverse transformation of $ex.DownTas(Z)$ is $ex.UpTa(Z)$. (68)

PROOF: To prove that $UpTa(Z)$ is the inverse transformation of $DownTas(Z)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent as only equivalence transformations are used.
ad (2) States remain unchanged by the transformation and its inverse
ad (3) Events remain unchanged by the transformation and its inverse.

ad (4) The aggregating superstate of Z and Z have equivalent ranges. Z is added as further target state of the event occurrence ex. The inverse replaces Z in the target states of the event occurrence ex by the aggregating superstate of Z. However, the aggregating superstate already is a target state of the event occurrence and as the target states of an event occurrence is a set this replacement removes Z from the target states of ex.

If ex is a non-synchronizing event occurrence the transformation changes it to a synchronizing one (note that the attribute $Sync$ of an event occurrence is a computed one) as afterwards ex has more than one target states. By applying the inverse ex is transformed back to a non-synchronizing event occurrence as afterwards ex has only one target state.

From (1) to (4) follows that the transformation $UpTa(Z)$ is the inverse transformation of $DownTas(Z)$. ■

5.11 The inverse transformation of $ComSe$

The transformation $ComSe(e_1, e_2, e)$ combines the event occurrences e_1 and e_2 with equivalent preconditions and the same source- and target states (see page 41). The inverse transformation is $SplitTe(e, P_1, P_2, e_1, e_2)$ where $P_1 = e_1.Postcondition$ and $P_2 = e_2.Postcondition$ (see page 44).

THEOREM: The inverse transformation of $ComSe(e_1, e_2, e)$ is $SplitTe(e, P_1, P_2, e_1, e_2)$ with $P_1 = e_1.Postcondition$ and $P_2 = e_2.Postcondition$ (69)

PROOF: To prove that $SplitTe(e, P_1, P_2, e_1, e_2)$ with $P_1 = e_1.Postcondition$ and $P_2 = e_2.Postcondition$ is the inverse transformation of $ComSe(e_1, e_2, e)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation.

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations,

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations and that

ad (1) The ranges are equivalent as only equivalence transformations are used.
ad (2) States remain unchanged by the transformation and its inverse
ad (3) Events remain unchanged by the transformation and its inverse.

ad (4) The transformation combines the event occurrences e_1 and e_2 to e. The disjunction of the postconditions of e_1 and e_2 is equivalent to the postcondition of e. The preconditions of e, e_1 and e_2 are equivalent. The inverse splits e again into e_1 and e_2 with its original postconditions.

From (1) to (4) follows that the transformation $SplitTe(e, P_1, P_2, e_1, e_2)$ is the inverse transformation of $ComSe(e_1, e_2, e)$. ■

5.12 The inverse transformation of $ComTe$

The transformation $ComTe(e_1, e_2, e)$ combines the event occurrences e_1 and e_2 with equivalent Postconditions and the same source- and target states (see page 42). The inverse transformation is $SplitSe(e, P_1, P_2, e_1, e_2)$ where $P_1 = e_1.PreC()$ and $P_2 = e_2.PreC()$ (see page 43).

THEOREM: The inverse transformation of $ComTe(e_1, e_2, e)$ is $SplitSe(e, P_1, P_2, e_1, e_2)$ with $P_1 = e_1.PreC()$ and $P_2 = e_2.PreC()$ (70)

PROOF: To prove that $SplitSe(e, P_1, P_2, e_1, e_2)$ with $P_1 = e_1.PreC()$ and $P_1 = e_2.PreC()$ is the inverse transformation of $ComTe(e_1, e_2, e)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent as only equivalence transformations are used.

ad (2) States remain unchanged by the transformation and its inverse

ad (3) Events remain unchanged by the transformation and its inverse.

ad (4) The transformation combines the event occurrences e_1 and e_2 to e. The disjunction of the preconditions of e_1 and e_2 is equivalent to the precondition of e. The postconditions of e, e_1 and e_2 are equivalent. The inverse splits e again into e_1 and e_2 with its original preconditions.

From (1) to (4) follows that the transformation $SplitSe(e, P_1, P_2, e_1, e_2)$ is the inverse transformation of $ComTe(e_1, e_2, e)$. ■

5.13 The inverse transformation of $SplitSe$

The transformation $SplitSe(e, P_1, P_2, e_1, e_2)$ splits the event occurrence e into two event occurrences e_1 and e_2. The parameters P_1 and P_2 are preconditions. Their disjuction must be equivalent to the precondition of e (see page 43). The inverse transformation is $ComTe(e_1, e_2, e)$ (see page 42).

THEOREM: The inverse transformation of $SplitSe(e, P_1, P_2, e_1, e_2)$ is $ComTe(e_1, e_2, e)$. (71)

PROOF: To prove that $ComTe(e_1, e_2, e)$ is the inverse schema transformation of $SplitSe(e, P_1, P_2, e_1, e_2)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent as only equivalence transformations are used.
ad (2) States remain unchanged by the transformation and its inverse
ad (3) Events remain unchanged by the transformation and its inverse.
ad (4) The transformation splits the event occurrence e into e_1 and e_2. Obvious e_1 and e_2 have the same source- and target states. The postcondtions of e, e_1 and e_2 are equivalent. The precondition of e is equivalent to the disjunction of the preconditions of e_1 and e_2. The inverse combines e_1 and e_2 as they have equivalent postconditions to e.

From (1) to (4) follows that the transformation $ComTe(e_1, e_2, e)$ is the inverse transformation of $SplitSe(e, P_1, P_2, e_1, e_2)$. ■

5.14 The inverse transformation of $SplitTe$

The transformation $SplitTe(e, P_1, P_2, e_1, e_2)$ splits the event occurrence e into two event occurrences e_1 and e_2. The parameters P_1 and P_2 are postconditions. Their disjuction must be equivalent to the postcondition of e (see page 44). The inverse transformation is $ComSe(e_1, e_2, e)$ (see page 41).

THEOREM: The inverse transformation of $SplitTe(e, P_1, P_2, e_1, e_2)$ is $ComSe(e_1, e_2, e)$. (72)

PROOF: To prove that $ComSe(e_1, e_2, e)$ is the inverse schema transformation of $SplitTe(e, P_1, P_2, e_1, e_2)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent as only equivalence transformations are used.
ad (2) States remain unchanged by the transformation and its inverse
ad (3) Events remain unchanged by the transformation and its inverse.
ad (4) The transformation splits the event occurrence e into e_1 and e_2. Obvious e_1 and e_2 have the same source- and target states. The preconditions of e, e_1 and e_2 are equivalent. The postcondition of e is equivalent to the disjunction of the postconditions of e_1 and e_2. The inverse combines e_1 and e_2 as they have equivalent preconditions to e.

From (1) to (4) follows that the transformation $ComSe(e_1, e_2, e)$ is the inverse transformation of $SplitTe(e, P_1, P_2, e_1, e_2)$. ■

5.15 The inverse transformation of $Combine$

The transformation $Combine(Z_1, Z_2, Z)$ combines two atomic states Z_1 and Z_2 to the state Z (see page 45). Z_1 and Z_2 must be elementary states or belonging to the same structured state.

For the inverse transformation we have to distinguish between two cases:

(1) if Z_1 and Z_2 are elementary states, than the inverse schema transformation is $Split(Z, Z_1.Range(), Z_2.Range(), Z_1, Z_2)$ (see page 49).

(2) if Z_1 and Z_2 are not elementary states, we need a more complex inverse transformation. The combined state will be a non elementary state. However, $Split$ demands that in the case of a non elementary state the state must not be source or target state of event occurrences. As Z_1 and Z_2 must be atomic states they must belong to a generalizing superstate if the states are not elementary states. After the transformation Z belongs to that generalizing superstate. For the inverse first we shift the event occurrences up to the generalizing superstate of Z. Second we split Z back into Z_1 and Z_2. Last

we shift the event occurrences back from the generalizing superstate. The inverse transformation $Combine^{-1}(Z)$ is defined as:

$Ex_S := \emptyset$
$Ex_T := \emptyset$
for all $e_i \in M.Event\_Occurrences$ **and** $Z \in e_i.Source\_States$ **do**
  e_i.UpSg(Z)
  $Ex_S := Ex_S + e_i$
end for
for all $e_i \in M.Event\_Occurrences$ **and** $Z \in e_i.Target\_States$ **do**
  e_i.UpTg(Z)
  $Ex_T := Ex_T + e_i$
end for
$Split(Z, Z_1.Range(), Z_2.Range(), Z_1, Z_2)$
for all $e_i \in Ex_S$ **do**
  $Ex := e_i$.DownSg($Z_1.Belongs\_to$)
  if $e_i \in Ex_T$ **then**
    replace e_i in Ex_T by Ex
  end if
end for
for all $e_i \in Ex_T$ **do**
  e_i.DownTg($Z_1.Belongs\_to$)
end for

Note that an event occurrence if shifted down from a generalizing superstate using $DownSg$ is replaced by several other event occurrences. The set of these event occurrences are returned by the transformation. If a shifted event occurrence is member of Ex_T too it must be replaced by the set Ex.

To prove that both transformations are inverse schema transformations we distinguish between this two cases.

THEOREM: The inverse transformation of $Combine(Z_1, Z_2, Z)$ is $Split(Z, Z_1.Range(), Z_2.Range(), Z_1, Z_2)$, if Z_1 and Z_2 are elementary states. (73)

PROOF: To prove that $Split(Z, Z_1.Range(), Z_2.Range(), Z_1, Z_2)$ is the inverse schema transformation of $Combine(Z_1, Z_2, Z)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent as only equivalence transformations are used.

ad (2) The transformation combines the states Z_1 and Z_2 which are atomic and elementary states to an atomic and elementary state Z. The disjunction of the ranges of Z_1 and Z_2 is equivalent to the range of Z. The inverse splits Z back to Z_1 and Z_2.

ad (3) Events remain unchanged by the transformation and its inverse.

ad (4) The transformation combines two states which are elementary and atomic states. Event occurrences, having Z_1 or Z_2 as source (target) state don't have further source (target) states. In such event occurrences Z_1 or Z_2 are replaced by Z in their source and target states. The guard of event occurrences having Z_1 or Z_2 as source state is replaced by its precondition.

The inverse splits Z back into Z_1 and Z_2 (which are disjoint). Event occurrences having Z as source and target state are duplicated, their source and target states are adopted. If Z is a target state of an event occurrence its postcondition is replaced by the conjunction of the original postcondition and its new target state (Z_1 or Z_2).

Let ex be an event occurrence having Z as source or target state. In the most general case (ex has Z as source and target state) ex is replaced by the event occurrences $ex_1 \dots ex_4$. For each ex_i follows that its pre- and postconditions result in $false$ or not. If the conditions do not result in $false$ (otherwise it is deleted by the transformation $Clean$) and as Z_1 and Z_2 are disjoint states then there must be an event occurrence e_j before applying the transformation and its inverse with equivalent pre- and postconditions as we use only equivalence transformations. Otherwise the specification of the events would not be equivalent.

From (1) to (4) follows that the transformation $Split(Z, B_1, B_2, Z_1, Z_2)$ is the inverse schema transformation of $Combine(Z_1, Z_2, Z)$ if Z_1 and Z_2 are elementary states. ■

THEOREM: The inverse transformation of $Combine(Z_1, Z_2, Z)$ is $Combine^{-1}(Z)$, if Z_1 and Z_2 are not elementary states. (74)

PROOF: To prove that $Combine^{-1}(Z)$ is the inverse schema transformation of $Combine(Z_1, Z_2, Z)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation.

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations,

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations and that

ad (1) The ranges are equivalent as only equivalence transformations are used.

ad (2) The transformation combines the atomic states Z_1 and Z_2 belonging to a state Z_S to an atomic state Z belonging to Z_S too. The disjunction of the ranges of Z_1 and Z_2 is equivalent to the range of Z. The inverse splits Z back to Z_1 and Z_2, belonging to Z_S again.

ad (3) Events remain unchanged by the transformation and its inverse.

ad (4) The transformation combines two atomic states. As both states are atomic states belonging to the same structured states not both states can be source (target) states of an event occurrence. Z_1 and Z_2 are replaced by Z in the source- and target states of event occurrences. The guard of event occurrences having Z_1 or Z_2 as source state is replaced by its precondition.

In the first steps the inverse transformation shifts the event occurrences of the combined state to its structured state (which must be a generalizing superstate) using the transformations $UpSg$ and $UpTg$. The guard of event occurrences having Z as source state is replaced by the precondition. The shifted event occurrences are collected in the set Ex_S and Ex_T.

After that Z is splitted into Z_1 and Z_2. Then each event occurrence of the sets Ex_S and Ex_T is shifted back from the generalizing superstate using the transformations $DownSg$ and $DownTg$.

The application of the transformations $UpSg$ or $UpTg$ to an event occurrence results into an event occurrence with equivalent pre- and postconditions. Using $UpSg$ replaces the guard of an event occurrence by its precondition.

Shifting down an event occurrence using the transformation $DownSg$ results in n new event occurrences $e_1 \dots e_n$, where n is the number of states covered by the generalizing superstate. The guards of the event occurrences $e_1 \dots e_n$ are equivalent to the guard of the shifted event occurrence. The generalizing superstate is replaced by a state Z_i, covered by the generalizing superstate in the source states of an e_i. However, as the states covered by a generalizing superstate must be disjoint it is easy to see that $n-1$ of the event occurrences $e_1 \dots e_n$ have preconditions resulting in $false$ and therefore are deleted by $Clean$.

Analogous happens by shifting an event occurrence down using the transformation $DownSg$. N event occurrences are produced, the target states are changed. The postcondition of an shifted event occurrence is replaced by the conjunction of its original postcondition and the range of its new target state. Again $n-1$ of the event occurrences $e_1 \dots e_n$ must have postconditions resulting in $false$ as the states covered by a generalizing superstate must be disjoint. These event occurrences are removed by the transformation $Clean$.

From (1) to (4) follows that the transformation $Combine^{-1}(Z)$ is the inverse transformation of $Combine(Z_1, Z_2, Z)$ if Z_1 and Z_2 are not elementary states. ■

5.16 The inverse transformation of $Split$

The transformation $Split(Z, B_1, B_2, Z_1, Z_2)$ splits an atomic state Z into two states Z_1 and Z_2 (see page 49). The parameters B_1 and B_2 are conditions of the states Z_1 and Z_2. B_1 and B_2 have to be disjoint and their disjunction must be equivalent to the range of Z. If Z is not an elementary state it must not be a source or target of an event occurrence. The inverse transformation is $Combine(Z_1, Z_2, Z)$ (see page 45).

THEOREM: The inverse transformation of $Split(Z, B_1, B_2, Z_1, Z_2)$ is $Combine(Z_1, Z_2, Z)$ (75)

PROOF: To prove that $Combine(Z_1, Z_2, Z)$ is the inverse schema transformation of $Split(Z, B_1, B_2, Z_1, Z_2)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent as only equivalence transformations are used.

ad (2) The transformation splits a state Z into two states Z_1 and Z_2. The disjunction of the ranges of Z_1 and Z_2 is equivalent to the range of Z. The inverse combines Z_1 and Z_2 to Z.

ad (3) Events remain unchanged by the transformation and its inverse.

ad (4)

(a) If Z are not an elementary state, it must not be source or target state of event occurrences when applying the transformation. In that case event occurrences remain unchanged by the transformation and its inverse.

(b) If Z is an elementary state event occurrences having Z as source or target state are copied. Let ex be an event occurrence having Z as source or target state. In the most general case (Z is source- and target state) ex is replaced by the event occurrences $ex_1 \dots ex_4$. The guards of these event occurrences remain unchanged. However the postcondition of the event occurrences are replaced by the conjunction of their original postcondition and the range of their new target state. If an event occurrence afterwards has pre- or postconditions resulting in $false$ it is removed by the transformation $Clean$. However, it is easy to see that at least one event occurrence remains.

The inverse transformation combines the state Z_1 and Z_2 to Z. In event occurrences having Z_1 or Z_2 as source state the guard of these event occurrences are replaced by its precondition. However, as the range of Z is equivalent to the disjunction of the ranges of Z_1 and Z_2, which are disjoint, such preconditions are equivalent to their original preconditions. The source and target states are adopted resulting in the following set of event occurrences with their pre- and postcondition

$$\begin{aligned}
ex_1 &= (Z_1.Range() \wedge ex.Guard, Z_1.Range() \wedge ex.Postcondition) \\
ex_2 &= (Z_1.Range() \wedge ex.Guard, Z_2.Range() \wedge ex.Postcondition) \\
ex_3 &= (Z_2.Range() \wedge ex.Guard, Z_1.Range() \wedge ex.Postcondition) \\
ex_4 &= (Z_2.Range() \wedge ex.Guard, Z_2.Range() \wedge ex.Postcondition)
\end{aligned}$$

Note, that some of these event occurrences might have pre- or postconditions resulting in $false$ and therefore, are deleted by $Clean$. In this case some of the following explanations can be dropped.

After applying the inverse transformation $Clean$ is used. In a first step $Clean$ combines the event occurrences ex_1 and ex_3 to one occurrence by the disjunction of their preconditions as they have the same event and equivalent postconditions. Furthermore ex_2 and ex_4 are combined. Afterwards the combined event occurrences can be combined once again as they have now equivalent preconditions, resulting in one event occurrence. As only equivalence transformations are used the combined event occurrence and ex have equivalent pre- and postconditions.

All other cases follow immediately from this most general case.

From (1) to (4) follows that the transformation $Combine(Z_1, Z_2, Z)$ is the inverse transformation of $Split(Z, B_1, B_2, Z_1, Z_2)$. ■

5.17 The inverse transformation of $Geng$

The transformation $Geng(Z_1 \dots Z_i, G)$ produces a state generalization with the generalizing superstate G covering the states $Z_1 \dots Z_i$ (see page 52). The inverse transformation is $Decg(G)$ (see page 54)) .

THEOREM: The inverse transformation of $Geng(Z_1 \ldots Z_i, G)$ is $Decg(G)$ (76)

PROOF: To prove that $Decg(G)$ is the inverse transformation of $Geng(Z_1 \ldots Z_i, G)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent as only equivalence transformations are used.

ad (2) The transformation indroduces a generalizing superstate G covering the states $Z_1 \ldots Z_i$. The inverse deletes G, the states $Z_1 \ldots Z_i$ either belongs to its original structured state or are elementary states.

ad (3) Events remain unchanged by the transformation and its inverse.

ad (4) Event occurrences remain unchanged by the transformation and its inverse.

From (1) to (4) follows that the transformation $Decg(G)$ is the inverse transformation of $Geng(Z_1 \ldots Z_i, G)$. ■

5.18 The inverse transformation of $Decg$

The transformation $Decg(G)$ decomposes the a state generalization with the generalizing superstate G, which is not source or target state of event occurrences (see page 54). The states $Z_1 \ldots Z_i$ covered by G either became elementary states or belongs to the structured state G belongs to. The inverse transformation is $Geng(Z_1 \ldots Z_i, G)$ (see page 52).

THEOREM: The inverse transformation of $Decg(G)$ is $Geng(Z_1 \ldots Z_i, G)$ where each state $Z_i \in G.Covers$. (77)

PROOF: To prove that $Decg(G)$ is the inverse transformation of $Geng(Z_1 \ldots Z_i, G)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent as only equivalence transformations are.

ad (2) The transformation deletes the generalizing superstate G. The states $Z_1 \dots Z_i$ coverd by G became elementary states or belongs to the generalizing superstate of G. The transformation demands that G does not belong to an aggregating superstate. The inverse transformation introduces G covering $Z_1 \dots Z_i$ again. G belongs is an elementary state or belongs to its original structured state.

ad (3) Events remain unchanged by the transformation and its inverse.

ad (4) Event occurrences remain unchanged by the transformation and its inverse.

From (1) to (4) follows that the transformation $Geng(Z_1 \dots Z_i, G)$ is the inverse transformation of $G.Decg$. ■

5.19 The inverse transformation of $Gena$

The transformation $Gena(Z, n, A)$ builds a state aggregation based upon the atomic state Z resulting in the aggregating superstate A. The aggregating superstate covers n generalizing superstates. Each of them covers one atomic state (see page 56). The inverse transformation is $Deca(A, Z)$ (see page 58).

THEOREM: The inverse transformation of $Gena(Z, n, A)$ is $Deca(A, Z)$. (78)

PROOF: To prove that $Deca(A, Z)$ is the inverse transformation of $Gena(Z, n, A)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent as only equivalence transformations are.

ad (2) The transformation introduces a state aggregation based upon an atomic state Z. The inverse transformation deletes this state aggregation resulting in the atomic state Z.

ad (3) Events remain unchanged by the transformation and its inverse.

ad (4) The transformation replaces Z by A in the source and target states of event occurrences. The inverse transformation replaces A by Z in the source and target states of event occurrences. Both states have equivalent ranges.

From (1) to (4) follows that the transformation $Deca(A, Z)$ is the inverse transformation of $Gena(Z, n, A)$. ■

5.20 The inverse transformation of $Deca$

The transformation $Deca(A, Z)$ decomposes a state aggregation with the aggregating superstate A resulting into the atomic state Z. The aggregating superstate must cover n generalizing superstates. Each of these generalizing superstates must cover only one atomic state. Furthermore none of the states (except A) of the state aggregation must be a source or target state of an event occurrence (see page 58). The inverse transformation is $Gena(Z, n, A)$ where n is the number of states in $A.Covers$ (see page 56).

THEOREM: The inverse transformation of $Deca(A, Z)$ is $Gena(Z, n, A)$, where n is the number of generalizing superstates of $A.Covers$. (79)

PROOF: To prove that $Gena(Z, n, A)$ is the inverse transformation of $Deca(A, Z)$ it is sufficient to show that

(1) the ranges are equivalent before and after the transformation,

(2) each state has a corresponding state with an equivalent range and the same structure after the transformations,

(3) each event has a corresponding event with an equivalent event specification after the transformations and that

(4) each event occurrence has a corresponding event occurrence with equivalent pre- and postconditions after the transformations.

ad (1) The ranges are equivalent as only equivalence transformations are used.

ad (2) The transformation decomposes the complete state aggregation resulting into the atomic state Z. The aggregating superstate must cover n generalizing superstates. Each of these states must cover exactly one atomic state. The inverse transformation based upon Z introduces exactly the same state aggregation.

ad (3) Events remain unchanged by the transformation and its inverse.

ad (4) The transformation replaces A by Z in the source and target states of event occurrences. The inverse transformation replaces Z by A in the source and target states of event occurrences. Both states have equivalent ranges. Other states of the state aggregation must not be source or target states of event occurrences.

From (1) to (4) follows that the transformation $Gena(Z, n, A)$ is the inverse transformation of $Deca(A, Z)$. ■

6 Properties of the schema transformations

The main property of the presented schema transformation is that they are equivalence transformation. The application of a schema transformation on a dynamic model results into a diffent but equivalent dynamic model.

For each discussed schema transformation there exists an inverse schema transformation, which, however, may be a complex transformations.

At last the presented set of schema transformations is complete. By that if two dynamic models are equivalent according to our definition they can be transformed into each other without changing the semantics of the dynamic models.

THEOREM: If the dynamic models M_1 and M_2 are equivalent than there exists a sequence of schema transformations to transform M_1 into M_2 and vice versa. (80)

PROOF: To prove that M_1 can be transformed to M_2 (and vice versa) if M_1 and M_2 are equivalent it is sufficient to show that

(1) M_1 can be transformed to one elementary and atomic state Z_1 by using only schema transformations,

(2) M_2 can be transformed to one elementary and atomic state Z_2 by using only schema transformations,

(3) Z_1 can be transformed such that for each event occurrence of Z_1 there exists an event occurrence of Z_2 with equivalent pre- and postconditions and vice versa and that

(4) each basic schema transformation has an inverse transformation

ad (1) and (2) A dynamic model M can be transformed to only one atomic state using the following schema transformations:

shift all event occurrences of M down to atomic states using the corresponding schema transformations

repeat
  for all generalizing superstates G of M covering only atomic states **do**
    combine the states covered by G to one atomic state using the transformation $Combine$
  end for
  for all generalizing superstates G of M covering only one atomic state **and** G.Belongs_to $\neq$ aggregating superstate **do**
    Decg(G)
  end for
  for all aggregating superstates A of M covering only generalizing superstates where each of them covers only one atomic state **do**
    shifts the event occurrences from the atomic states to A using the corresponding schema transformations
    Deca(A,Z)
  end for
until all states of M are atomic states
combine all atomic states of M using the transformation $Combine$

First we shift all event occurrences down to atomic states using the corresponding transformations $DownSg$, $DownTg$, $DownSa$ and $DownTa$. Afterwards we loop until M consists of atomic states only. These states can be combined by a sequence of $Combine$ transformations to one atomic state.

Within the loop we search for generalizing superstates covering only atomic states. These atomic states are combined to one atomic states by a sequence of $Combine$ transformations. Note, that due to the orthogonality and the fact, that at least one state must belong to a structured state in each state hierarchy there must be such a construct.

In the next step we search for generalizing superstates not belonging to an aggregating superstate which covers only one atomic state. Such generalizing superstates are decomposed by $Decg$ resulting in one atomic state. Note that the generalizing superstate is not a source or target state of event occurrences as we have shifted down event occurrences to the atomic states.

At last we search for state aggregations having the structure demanded by the transformation $Deca$, that is that the aggregating superstate only covers generalizing superstates which itself covers only one atomic state. In this situations we shift the event occurrences of these state aggregations to the aggregating superstate and decompose the state aggregating applying the transformation $Deca$, resulting in a new atomic state.

ad (3) It is easy to see that Z_1 can be transformed such that for each event occurrence of Z_1 there exist an event occurrence of Z_2 with equivalent pre- and postconditions as we have a schema transformation for each definition of the relation

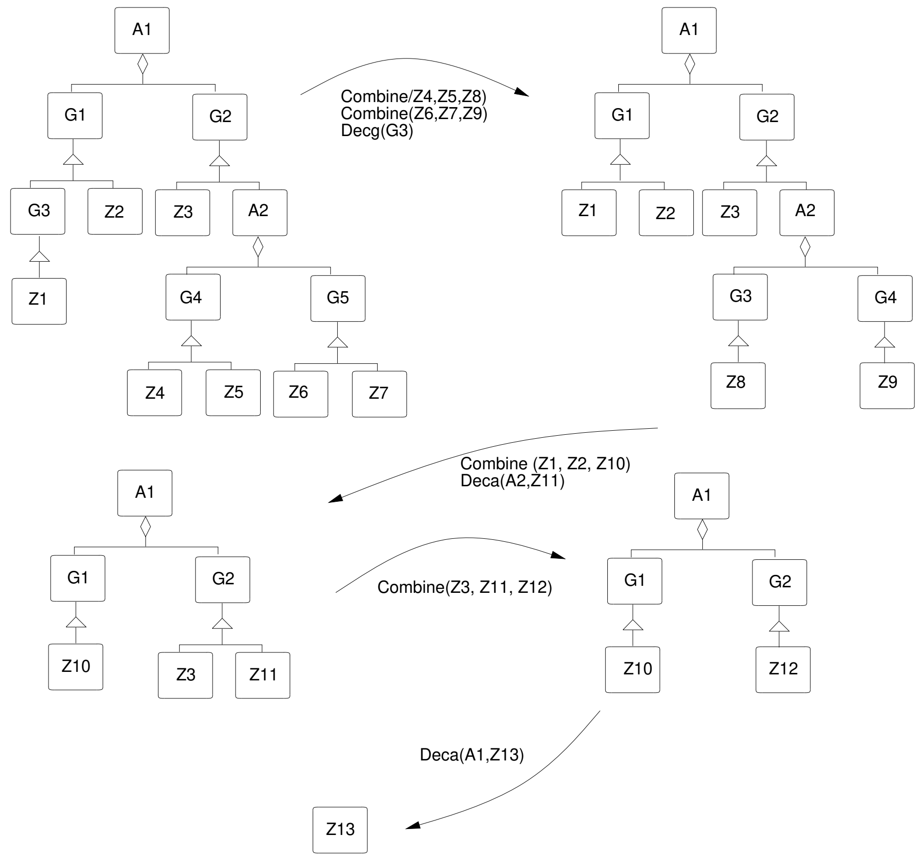

Figure 13: Transforming a state hierarchie into ony atomic state

Ξ. These transformations are $ComSe$, $ComTe$, $SplitSe$ and $SplitTe$ covering the definitions (1) to (4) of the relation Ξ (compare definition 18, p. 19). For the definition (5) and (6) of Ξ no transformations are necessary.

ad (4) We refer to section 5 where we proved that each basic schema transformation has an inverse schema transformation.

From (1) to (4) follows that if $M_1 \equiv M_2$ than there exists a sequence of schema transformation to transform M_1 into M_2 and vice versa. ■

Consider figure 13 where we show a state hierarchie which is transformed to one atomic state. Please note that we do not consider event occurrences in this example.

In the first step we combine the states of the generalizing superstates G_4 and G_5. Furthermore the generalizing superstate G_3 can be decomposed as it consits of only one atomic state.

In the second step we decompose the aggregating superstate A_2 as it fulfill the conditions demanded by the transformation $Deca$. The states Z_1 and Z_2 are combined to the state Z_{10}.

In the next iteration we combine the states Z_3 and Z_{11} to one atomic state. At last we decompose the aggregating superstate A_1 as afterwards it fulfill the conditions demandet by the transformation $Deca$, resulting in one atomic state Z_{13}.

7 Conclusion and Future Work

We presented a formalization of a model for representing the dynamic behavior of objects. We present a meta-model and define the (abstract) semantics of state charts as partial specification of methods. This allows the definition of the equivalence of dynamic models. The main contribution of this work is the development of a complete set of basic schema transformation which maintain the semantics. The presented set of transformations suffices to derive any equivalent dynamic model from a given one.

There are several applications for the presented methodology. It serves as sound basis for design tools. It enables analysts and designers to start from an initial model and improve the quality of the model step by step. We can provide automatic support to achieve certain presentation characteristics of model. A model can be transformed to inspect it from different points of view. In particular a model suitable for conceptual comprehension can be transformed to a model better suited for implementation similar to the transformation of static conceptual models to logical models.

Our main application and motivation for the development of the model was to support automatic integration of partial models. This is the extension of the view integration approach in conceptual modeling to also incorporate dynamic models ([Fra96]).

References

[BCN92] C. Batini, S. Ceri, and S. B. Navathe. *Conceptual Database Design: An Entity-Relationship Approach*. The Benjamin/Cummings Publishing Company, Inc, 1992.

[Boo91] G. Booch. *Object-Oriented Design with Applications*. Benjamin Cummings, 1991.

[CAB+94] 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.

[Fir96] D. G. Firesmith. The Inheritance of State Models. *Report on Object Analysis and Design (ROAD)*, 2(6):13 – 15, March 1996.

[FM94a] A. Formica and M. Missikoff. Constraint Satisfiability in Object-Oriented Databases. In J. Eder and L.A. Kalinichenko, editors, *East/West Database Workshop*, Workshops in Computing. Springer, 1994. Klagenfurt.

[FM94b] A. Formica and M. Missikoff. Correctness of Inheritance Hierarchies in Recursive Object-Oriented Database Schemas. Technical report, Consiglio Nazionale Delle Ricerche - Instituto Di Analisi Dei Sistemi Ed Informatica (CNR-IASI), 1994.

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

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

[KS94] G. Kappel and M. Schrefl. Inheritance of Object Behaviour - Consistent Extension of Object Life Cycles. In J. Eder and L. A. Kalinichenko, editors, *Proceedings of the Second International East/West Database Workshop*, pages 289 – 300. Springer, September 1994.

[KS96] G. Kappel and M. Schrefl. *Objektorientierte Informationssysteme*. Springer Verlag, 1996.

[Mos95] *MOSAICO 3.5 User Manual*, March 1995.

[OMT93] *OMTool: User Guide 2.0*, September 1993.

[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.

8 Appendix

In this appendix we present the defined schema transformations in alphabetical order with a short textual description.

Name	short textual description
$Clean$	Deletes event occurrences whose pre- and/or postconditions results in $false$ and combines event occurrences with equivalent pre- or postconditions by the disjunction of their post- respectively preconditions (page 62).
$Combine(Z_1, Z_2, Z)$	Combines two atomic states Z_1 and Z_2 to the state Z (page 45).
$ComSe(e_1, e_2, e)$	Combines the event occurrences e_1 and e_2 with equivalent preconditions and the same source- and target states to e (page 41).
$ComTe(e_1, e_2, e)$	Combines the event occurrences e_1 and e_2 with equivalent postconditions and the same source- and target states to e (page 42).
$ComEx$	Combines event occurrences with equivalent pre- resp. postconditions by disjunction of their post- respectively preconditions (page 62).
$Deca(A, Z)$	Decomposes a state aggregation with the aggregating superstate A resulting in an atomic state Z. Each state covered by A must cover itself only one atomic state. None of the states of the state aggregation (except A) must occur in source or target states of event occurrences (page 58).
$Decg(G)$	Decomposes a generalizing superstate G not belonging to an aggregating superstate. G must not be a source- or target state of event occurrences (page 54).
$DelEx$	Deletes event occurrences whose pre- and/or postconditions results in $false$ (page 61).
$DownSa(Z)$	Shifts an event occurrence from the aggregating superstate of Z to Z, if the aggregating superstate of Z is a source state of the event occurrence (page 37).

Name	short textual description
$DownSas(Z)$	Adds Z as a new source state to an event occurrence, if the aggregating superstate of Z is a source state of the event occurrence (page 39).
$DownSg(Z)$	Shifts an event occurrence having the generalizing superstate Z as source state to the states belonging to Z (page 29).
$DownSTg(Z)$	Shifts an event occurrence having the generalizing superstate Z as source and target state to the states belonging to Z (page 33).
$DownTa(Z)$	Shifts an event occurrence from the aggregating superstate of Z to Z, if the aggregating superstate of Z is a target state of the event occurrence (page 38).
$DownTas(Z)$	Adds Z as a new source state to an event occurrence, if the aggregating superstate of Z is a target state of the event occurrence (page 39).
$DownTg(Z)$	Shifts an event occurrence having the generalizing superstate Z as target state to the states belonging to Z (page 30).
$Gena(Z, n, A)$	Build a state aggregation based upon the atomic state Z resulting in the aggregating superstate A which covers n generalizing superstates. To each generalizing superstate belongs one atomic state (page 56).
$Geng(Z_1 \dots Z_i, G)$	Produces a state generalization with a generalizing superstate G and the states $Z_1 \dots Z_i$, which must be elementary states or belonging to the same generalizing superstate (page 52).
$Split(Z, B_1, B_2, Z_1, Z_2)$	Splits an atomic state Z into two states Z_1 and Z_2 (page 49). The parameters B_1 and B_2 are conditions of the states Z_1 and Z_2. B_1 and B_2 have to be disjoint and their disjunction must be equivalent to the range of Z.
$SplitSe(e, P_1, P_2, e_1, e_2)$	Splits an event occurrence e into two event occurrences e_1 and e_2. The parameters P_1 and P_2 are the preconditions of e_1 and e_2. Their disjunction must be equivalent to the precondition of e. (page 43).
$SplitTe(e, P_1, P_2, e_1, e_2)$	Splits an event occurrence e into two event occurrences e_1 and e_2. The parameters P_1 and P_2 are the postconditions of e_1 and e_2. Their disjunction must be equivalent to the postcondition of e. (page 44).

Name	**short textual description**
$UpSa(Z)$	Shifts an event occurrence with Z as source state to the aggregating superstate of Z. (page 34).
$UpSg(Z)$	Shifts an event occurrence having the state Z as source state to the generalizing superstate of Z (page 25).
$UpSTa(Z)$	Shifts an event occurrence with Z as source and target state to the aggregating superstate of Z (page 36).
$UpSTg(Z)$	Shifts an event occurrence having the state Z as source and target state to the generalizing superstate of Z (page 28).
$UpTa(Z)$	Shifts an event occurrence with Z as target state to the aggregating superstate of Z.(page 35).
$UpTg(Z)$	Shifts an event occurrence having the state Z as target state to the generalizing superstate of Z (page 26).