Christoph Schütz

Multilevel Business Modeling

From Hetero-Homogeneous Data Warehouses to Multilevel Business Processes

in collaboration with:

Lois M. L. Delcambre
Michael Schrefl
Bernd Neumayr

Johannes Kepler University
Portland State University

Contents

1

Introduction

Previous research on information systems design has revealed short-comings of existing modeling techniques with respect to multilevel abstraction hierarchies [40], in particular with respect to heterogeneities in conceptual data warehouse models. Consequently, Neumayr et al. [38] have introduced *multilevel objects* (m-objects) and *multilevel relationships* (m-relationships) to tackle common issues in multilevel conceptual modeling. The application of m-objects and m-relationships has proven particularly promising in the field of data warehousing [39, 57, 60, 56]. In this report, we further investigate the hetero-homogeneous data warehouse modeling approach.

Moreover, we apply the proven concepts of m-objects and m-relationships to business process modeling. A business process involves tasks on many organizational levels. Thus, a process model may be viewed on different levels of abstraction, with varying levels of detail. Current modeling approaches support modular process models, but lack a compact representation and an explicit notation for abstraction levels. M-objects and m-relationships, however, might be extended in order to accommodate for the representation of business process models at multiple levels of abstraction. This modeling approach might be implemented in existing business process engines or further extended to support business process intelligence.

For example, the production process of a manufacturing company has multiple levels of abstraction. First, on the *corporation* level, a process model must account for the decision process of top management. Second, on the *department* level, a process model must account for the operational tasks of middle management which operates research, marketing, and distribution of the company. Finally, each individual *factory* has a manufacturing process. Entities may associate process models with these abstraction levels, and instantiate a single level.

The process models at different levels of abstraction are interconnected. Entities at more abstract levels may define meta processes for entities at more detailed levels of abstraction. Likewise, each entity defines common process models for entities at more detailed levels of abstraction. These common pro-

cess models may be refined through specialization, thereby extending m-object concretization to business process models.

This report presents the results of a research stay of Christoph Schütz at Portland State University from March 1st to August 31st, 2012. Host professor at Portland State University was Lois M. L. Delcambre of the Computer Science Department. Supervisor at the home institution, Johannes Kepler University in Linz, is Michael Schrefl of the Department of Business Informatics – Data & Knowledge Engineering.

In Chapter 2, we investigate the integration of heterogeneous information in data warehouses using the hetero-homogeneous approach together with a standardized metamodel. This chapter is a revised version of a paper [58] that was presented at the Americas Conference on Information Systems in August 2012.

In Chapter 3, we motivate the need for multilevel business process modeling, sketch the basic approach, identify the major design issues, and give examples of possible fields of application. This chapter is an extended version of a dissertation proposal [59] presented at the Workshop for Ph.D. Students in Information and Knowledge Management in November 2012.

In Chapter 4, we introduce the multilevel business artifact (MBA) as an extension of the multilevel object (m-object) [39] for data-centric business process modeling. This chapter is a revised version of a paper [58] presented at the Workshop on Data- and Artifact-centric Business Process Management in September 2012.

2

Integration and Reuse of Heterogeneous Information

The corporate data warehouse integrates data from various operational data stores of a company. These operational data stores may be heterogeneous with respect to the represented information. The hetero-homogeneous data warehouse modeling approach overcomes issues associated with the integration of heterogeneous information from the operational data stores by featuring a generally homogeneous schema which may be interspersed with heterogeneities in well-defined portions of the data. In order to leverage the capabilities of existing business intelligence (BI) tools for the analysis of hetero-homogeneous information, the schema must comply with the metamodel of the particular BI tool. The Common Warehouse Metamodel (CWM) is a standard for data warehouse metadata which facilitates the reuse of data across multiple BI tools. In this chapter, we present guidelines for modeling hetero-homogeneous data warehouses in the CWM. We demonstrate feasibility with a proof-of-concept prototype for the model-driven implementation of hetero-homogeneous data warehouses.

2.1 Introduction

The corporate data warehouse provides decision-makers with the information that is required for well-founded decisions, integrating data from various operational data stores of the different departments and local branches of a company. These data stores may be heterogeneous with respect to the represented information. For example, the production department has different requirements for their data than the accounting department. The data model of the corporate data warehouse reconciles the diverse data models of the operational data stores in a company.

In general, the integration of heterogeneous data models either results in the elimination of all heterogeneities or yields an overly complex integrated data model which preserves heterogeneities but is unfit for analysis purposes. Neither solution is particularly appealing. The elimination of heterogeneities

leads to the loss of valuable information that could otherwise improve decision quality. Complex data models, on the other hand, preserve heterogeneities at the expense of clarity. These data models exceed the capabilities of the analyst who is overburdened with the sheer volume and complexity of the presented information. Existing modeling solutions do not adequately solve this integration issue.

The hetero-homogeneous approach as presented by Neumayr et al. [39] overcomes the limitations of other data warehouse modeling techniques with respect to the representation of heterogeneities. Hetero-homogeneous data warehouse models are homogeneous with respect to a globally agreed schema. This schema may be specialized for portions of the data provided the specialization does not violate the global schema. Depending on which portion of the data is analyzed, the analyst can either rely on the globally agreed, homogeneous schema or leverage an increased amount of information that is available specifically for the analyzed portion. This principle recursively applies to the portions of the portions of the data. For example, a data warehouse records monthly revenues of product sales by city. For U.S. sales, a refined schema is employed, recording revenues by store rather than city and, apart from revenues, recording the sold quantity. Whenever the analysis concerns only U.S. data, the analyst can rely on an increased amount of information.

In order to leverage the capabilities of existing business intelligence (BI) tools for the analysis of hetero-homogeneous information, the schema must comply with the specific metamodel that is employed by the particular BI tool. The Common Warehouse Metamodel (CWM) is an open industry standard for representing and managing warehouse metadata which facilitates the reuse of multidimensional data across multiple BI tools. The CWM as a standard for representing multidimensional schemas plays a major role in component reuse, which is a key issue in modern software engineering [11]. Real-world use cases confirm the suitability of the CWM to serve as a foundation for data warehouse metadata standardization [37]. The CWM easily maps to the proprietary metadata models of software vendors. In addition, many BI tools natively support the CWM either for exchange (IBM InfoSphere Data Architect, SAS Data Integration, CA ERwin Data Modeler) or as their primary representation model (Pentaho Metadata). Consequently, many data warehouse modeling approaches base their concepts on the CWM [51] or rely directly on the CWM for vendor-neutral specifications of multidimensional data models [46].

In this chapter, we present guidelines for modeling hetero-homogeneous data warehouses in the CWM. We begin with a short introduction about data warehousing in general and the CWM in particular, followed by a review of the relevant literature. We then illustrate our modeling approach on a basic example. We demonstrate feasibility of our approach by providing a proof-of-concept prototype for the model-driven implementation of hetero-homogeneous data warehouses. This prototype builds on the existing management system for hetero-homogeneous data warehouses as introduced by

Schütz [57], extending this system with functionality for the export of CWM metadata in order to improve interoperability with other BI tools.

2.2 Background

2.2.1 Data Warehousing, OLAP, and the CWM

Data warehouses organize strategic data for decision support using dedicated data models. Unlike operational databases, which support day-to-day business operations, data warehouses provide decision makers with information at an adequate level of detail [8, p. 977]. This information is commonly represented by multidimensional schemas. A multidimensional schema consists of (hyper-) cubes and dimensions. Dimensions consist of hierarchically ordered aggregation levels; cubes consist of cells. The cells of a cube represent business events of interest which are quantified by measures. For example, a cube may visualize product sales over time in various local markets (Figure 2.1). Each cell of such a sales cube might store the monthly revenues of a product model within a city or store.

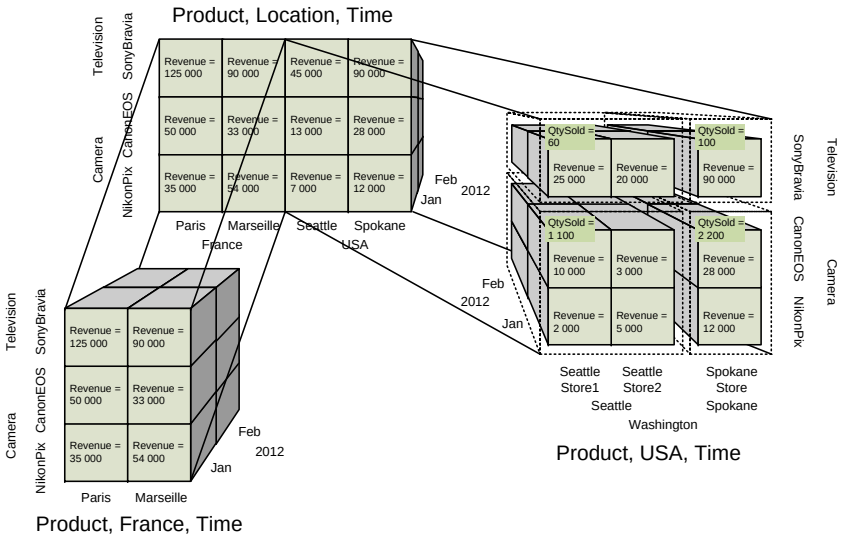


Fig. 2.1. A three-dimensional sales cube with heterogeneities in the *USA* region

Online Analytical Processing (OLAP) refers to the analyses that are performed on the data of a data warehouse. Among the most common OLAP operations are slice and dice, roll up and drill down. Slice and dice refer to the selection of cells from a base cube in order to obtain a sub-cube. This sub-cube may contain only the information that is needed for a particular analysis.

Roll up and drill down refer to a change in granularity of the viewed data. Hierarchically organized dimensions allow for the analysis of data at various levels of granularity. For example, a cube may store the monthly revenues of products in French and U.S. cities (Figure 2.1). The French area manager may be interested only in the France portion of the cube, thus performing a dice operation on the cube to obtain only sales in France. Through summarization of revenues by year and product category, the analyst may roll up the data to obtain an aggregated view with a coarser level of granularity. Likewise, along with the dice operation to obtain a USA sub-cube, an analyst may drill down for an in-depth analysis to view the revenues by store rather than city.

In real-world applications, not all portions of a cube of data are uniform. Some portions of a cube may contain more information than others. Such heterogeneous cubes present a problem for modelers and analysts alike. Modelers are torn between a faithful representation of information and the definition of an understandable schema. Analysts cannot rely on the results of the analysis as availability of the necessary information cannot be guaranteed for any portion of the cube. For example, the sales cube in Figure 2.1 might contain additional data for the USA sub-cube. For U.S. sales, revenues are available per store rather than city. Furthermore, U.S. sales can be aggregated by states, a political entity that is non-existent in other countries. Apart from revenues, the USA sub-cube records an additional measure, namely the sold quantity (*QtySold*). The sold quantity, however, is recorded by product category, year, and city. The USA sub-cube is thus multi-granular, containing measures at multiple levels of granularity. Throughout the remainder of this chapter, we use the cube in Figure 2.1 as hypothetical example for illustration purposes. Real-world applications have shown, though, that similar kinds of heterogeneities actually occur in reality. Consider, for example, the data model of the German marketing research institute GfK [4], which features, among others, heterogeneous product hierarchies.

The analysis of data requires detailed knowledge about nature and structure of the analyzed data. Therefore, in order to effectively and efficiently analyze the data, BI tools require the schema of the analyzed data. This schema is metadata - data about data. Metadata, much like "ordinary" data, again follow a specific schema which is referred to as the metamodel.

Many BI tools employ their own proprietary metamodel. The Object Management Group (OMG), in an effort to harmonize the various proprietary models for warehouse metadata, promotes the Common Warehouse Metamodel (CWM) as an open industry standard. The CWM builds on the Unified Modeling Language (UML) and complies with the Meta Object Facility (MOF). UML and MOF are widely accepted standards for model-driven software development. The Extensible Markup Language (XML) and the XML Metadata Interchange (XMI) language serve as a universally understood exchange format for CWM metadata. A more detailed introduction to the CWM standard is given by Poole et al. [50].

2.2.2 Related Work

Conceptual data warehouse models abstract from a concrete implementation and present a business-oriented view on the data. The Dimensional Fact Model (DFM) as proposed by Golfarelli et al. [10] is arguably the most popular conceptual modeling approach for data warehouses. The DFM emphasizes on facts and dimensions. Facts are business events of interest which are quantified by measures. The dimensions are hierarchically organized. Consequently, measures may be aggregated along the dimension hierarchies, thereby allowing for the analysis of facts on multiple levels of granularity. Most modeling approaches rely on similar modeling concepts. Pedersen et al. [47] evaluate several data warehouse modeling approaches and identify key requirements for modern data warehouse modeling. Popular conceptual modeling approaches, however, fall short of these requirements.

Most notably, the accurate representation of heterogeneous information presents a tough challenge for modelers. Heterogeneities add considerably to the complexity of data warehouse models. Furthermore, heterogeneous models are prone to summarizability issues [35]. Summarizability is an important property which ensures the correctness of aggregation operations [29]. Multidimensional normal forms for data warehouse dimensions have been proposed in order to avoid summarizability issues [28, 27]. Similarly, integrity constraints allow to reason about summarizability within heterogeneous dimensions [19, 18]. Apart from dimensions, heterogeneities may also occur in facts, yielding multi-granular data [20].

More recently, several conceptual data warehouse modeling approaches based on the UML or the Entity-Relationship (ER) model have been proposed which provide an increased support for complex modeling issues. Abelló et al. [1] propose yet another multidimensional model (YAM^2) based on the UML for modeling and querying data warehouses. Luján-Mora et al. [31] develop a UML profile for conceptual data warehouse modeling. Malinowski and Zimányi [32, 33] introduce the MultiDimER model which builds on the concepts of the ER model. The MultiDimER model incorporates an inheritance mechanism for representing heterogeneous dimensions. Pinet and Schneider [48] with their UML-based approach follow a similar principle for allowing heterogeneities in dimensions. Still, the UML/ER-based modeling approaches do not satisfactorily solve summarizability issues in heterogeneous dimensions and generally yield overly complex data warehouse models. Moreover, the UML-based approaches commonly focus on heterogeneous dimensions, thereby neglecting heterogeneous, multi-granular facts.

The hetero-homogeneous modeling approach of Neumayr et al. [39] features a generally homogeneous data warehouse model which may be interspersed with heterogeneities in well-defined portions of the data. Subdimensions and sub-cubes may introduce additional information with respect to the entire dimension or cube, respectively, without violating the common global model. This feature proves advantageous especially when integrating

data models of different data warehouses [60]. The hetero-homogeneous modeling approach allows for the preservation of heterogeneities during the process of model integration while reducing the complexity of heterogeneous data models.

Their support for model-driven implementation is a major advantage of UML-based conceptual modeling approaches. Prat et al. [51] provide a full-fledged data warehouse design method based on UML. This design method covers the transformation of the conceptual model into a logical model and a physical implementation. A similar design approach builds on the UML and the CWM [36, 46]. Kurze and Gluchowski [26] give an overview of current approaches and hint future research directions for model-driven data warehouse development. Rather than extending the UML, Neumayr et al. [39] employ multilevel objects (m-objects) and multilevel relationships (m-relationships) for modeling hetero-homogeneous data warehouses. M-objects and m-relationships overcome the strict separation of schema and instance. This schema/instance duality allows for the introduction of a compact inheritance mechanism for dimension and fact schemas. Schütz [57] presents a proof-of-concept prototype implementation for managing hetero-homogeneous data warehouses in a standard object-relational database. In this prototype implementation, the relational database automatically derives from the conceptual model with its m-objects and m-relationships. Despite providing basic export and analysis functionality, the system lacks a powerful exchange mechanism which preserves hetero-homogeneous information for other analysis tools.

2.3 Logical Hetero-Homogeneous CWM Modeling

In the CWM, the logical model describes the multidimensional structure of the data warehouse. The logical model consists of dimensions, hierarchies, and cubes as well as mappings between individual model elements. The logical model abstracts from a particular implementation, leaving the choice of the appropriate data structures to the physical model. In this section, we present guidelines for modeling hetero-homogeneous dimensions and cubes in the CWM. We base our approach on Neumayr et al. [39] and refer to Poole et al. [50] as an authoritative guide for the development of CWM applications. Some details of CWM modeling, for example, certain types of mappings, which are not specific to hetero-homogeneous data warehouse modeling are mentioned only briefly or left out due to space considerations.

2.3.1 Dimensions

The dimensions set the context for the analysis. A dimension consists of a multitude of members which belong to the same real-world concept. These members are used to denote business events of interest. Attributes further describe the members of a dimension. In each dimension, a designated attribute

identifies the members of the dimension. This attribute is referred to as the key of the dimension. For the key attribute, each member of the dimension must provide a unique value which no other member of that same dimension shares. The key attribute may be used to obtain a list of members of the dimension.

A dimension collects its members into aggregation levels. Some attributes are relevant for members at specific aggregation levels only. For that reason, attributes are attached to an aggregation level. Think of an aggregation level as a class. The class defines a set attributes; the instances assign values to these attributes. Similarly, the aggregation level defines a set of attributes; the members of the level assign values to these attributes. The members of a level, however, are not obliged to provide a value for each and every attribute that is defined for the level. For example, dimension Location contains data about geographic entities (Figure 2.2). Geographic entities exist at various levels of aggregation, namely *Country*, *Region*, *State*, *City*, and *Store*. Each geographic entity has a unique name which is the key of the dimension. For geographic entities at level *City*, the dimension contains data about the mayor and the elevation.

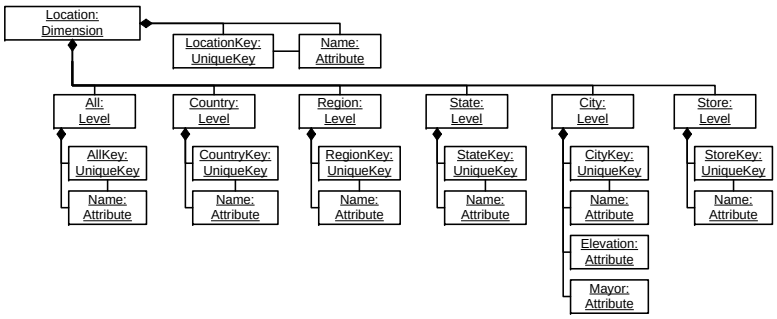


Fig. 2.2. Dimension *Location* with aggregation levels and attributes

The levels of a dimension are arranged in aggregation hierarchies. Within an aggregation hierarchy, the levels of a dimension are ordered from coarsest to most specific. The order of the levels determines the roll-up relationships between the levels. Each member of a level rolls up to a member of the precedent level. For example, given the primary hierarchy of dimension Location (Figure 2.3), level All is coarser than level *Country* and level *Country* is coarser than level *City*.

A dimension may have multiple aggregation hierarchies. Modelers commonly make use of this possibility in order to model alternative aggregation hierarchies for a dimension. For example, every city is located in a country and, in addition, every city is part of a region. Consequently, dimension Location - besides its primary hierarchy with levels *City*, *Country*, and *Top* (Figure 2.3)

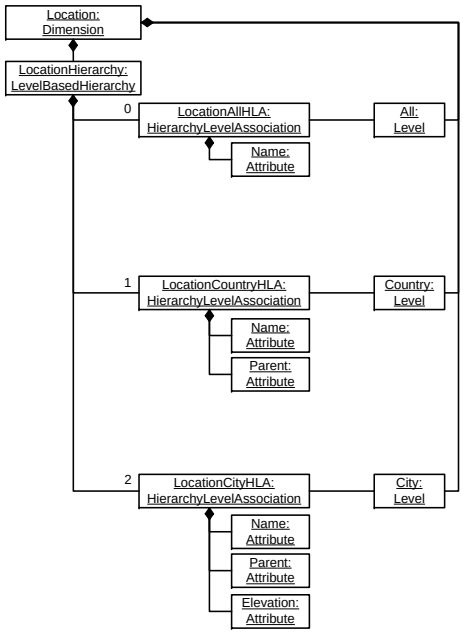


Fig. 2.3. Primary hierarchy of dimension *Location*

- has an alternative aggregation hierarchy with levels *City*, *Region*, and *Top* which is not shown in the examples.

Each hierarchy is homogeneous with respect to the levels, their relationships, and the attributes. While the dimension is inherently heterogeneous, defining for each level only optional attributes, the hierarchy imposes attributes and roll-up relationships to its members. That is, each member of an aggregation hierarchy must provide a value for all attributes associated with its level and roll up to a member of the precedent level. Note that this behavior is not standard CWM. Rather, it is an important guideline for hetero-homogeneous modeling.

The possibility of defining multiple aggregation hierarchies for the same dimension also allows for the representation of hetero-homogeneous information. In case a branch of a dimension introduces heterogeneities to the model, the branch has its own aggregation hierarchy defined. The branch is homogeneous with respect to the information that is contained in its hierarchy; it may be heterogeneous with respect to other branches, though. For example, the *USA* branch of dimension Location has additional aggregation levels. All U.S. cities roll up to a state, a political entity that does not exist in other countries. Also, the *USA* branch provides an additional level of detail underneath the *City* level: Level *Store* is the most specific aggregation level that is available for the whole *USA* branch. Furthermore, for all U.S. cities, data about the mayor is available in addition to the elevation. The USA branch

thus introduces heterogeneities. Consequently, the *USA* branch has its own aggregation hierarchy defined (Figure 2.4).

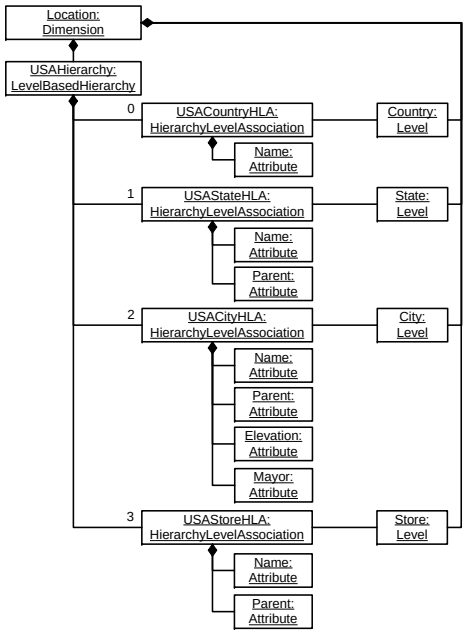


Fig. 2.4. Hierarchy of the *USA* branch of dimension *Location*

This *USA* hierarchy concretizes the primary hierarchy of dimension Location. It does not contradict the information given in the primary hierarchy. After all, level *City* still has attribute Elevation and rolls up to level Country. The *USA* hierarchy, however, includes more information which could not be included in the primary hierarchy without rendering the primary hierarchy heterogeneous. The *USA* hierarchy, nevertheless, is homogeneous even after the inclusion of this additional information. The information may not be available for the whole dimension; it is for the entire *USA* branch, though.

2.3.2 Cubes

A cube is of arbitrary dimensionality and contains an arbitrary number of measures. Cubes are the most important modeling primitive as they organize the main data of interest - measures. For example, Figure 2.5 illustrates cube Sales of dimensions *Product*, *Time*, and *Location*. The cube has measures *Revenue* and *QtySold*.

In hetero-homogeneous CWM modeling, the cube itself does not assert any level of granularity for any measure nor does it guarantee that a measure

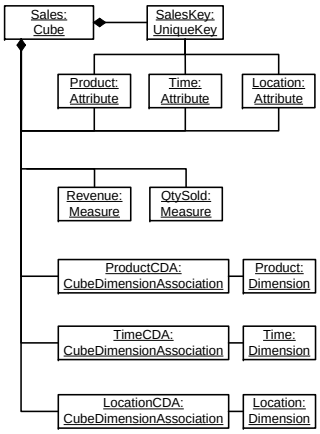


Fig. 2.5. Cube *Sales*

is available for all portions of the cube. Rather, a cube defines what is to be expected at the most from the data of a cube. In order to represent hetero-homogeneous cubes, the logical model borrows the concept of cube regions from the deployment model. In hetero-homogeneous modeling, cube regions represent portions of a cube with a homogeneous schema. Each cube region is single-granular, homogeneous with respect to the measures, and its schema applies only to a portion of the cube. Note that this is only a guideline for hetero-homogeneous modeling and not standard CWM behavior.

In case a sub-cube introduces heterogeneities with respect to the global schema, the sub-cube has its own cube region de-fined. The same principle as for hetero-homogeneous dimensions applies. Each cube region is homogeneous and guarantees certain properties for the data of a sub-cube. The cube region, however, may be heterogeneous with respect to a global schema in that it concretizes this schema. The cube region may contain an arbitrary subset of the cube's set of measures.

For example, cube *Sales* has three cube regions (Figure 2.6), each having a distinct set of measures and a single level of granularity. The primary cube region *ProductTimeLocation* defines the homogeneous schema for the entire cube. This cube region's level of granularity is ⟨ *Model, Month, City* ⟩, meaning that the measures of this cube region are recorded by product model, month, and city. The *USA* sub-cube records an additional measure and tracks revenues by store rather than city. For the *USA* sub-cube, two separate cube regions exist. The first cube region, *ProductTimeUSA1*, contains the measures that are recorded at the ⟨ *Category, Year, City* ⟩level of granularity. The other cube region, *ProductTimeUSA2*, contains the measures that are recorded at the ⟨ *Model, Month, Store* ⟩level of granularity. Notice that the definition that these cube regions are limited to the *USA* sub-cube cannot be done in the logical model but will rather be accomplished in the physical model.

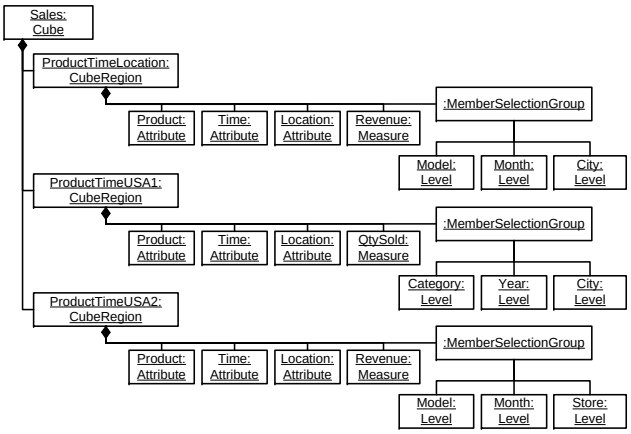


Fig. 2.6. Regions of cube *Sales*

2.4 Physical Hetero-Homogeneous CWM Modeling

The physical model describes the representation of the data in a database. The physical model consists of tables as well as mappings between these tables and the logical model. While the logical model serves as a schema for the formulation of analytical queries, the data for answering these queries come from the tables that are defined in the physical model. In this section, we propose a relational implementation for hetero-homogeneous data warehouses which considers the particularities of the logical model in the CWM. We rely on a variant of the widely-accepted star schema, consisting of dimension and fact tables. We further provide guidelines for mapping the logical hetero-homogeneous model to the physical representation of the data.

2.4.1 Dimension Tables

A dimension stores its data in a single dimension table. Each row of the dimension table represents a member of the dimension. The columns of the table represent the attributes of the dimension's members. Two additional columns are introduced as auxiliaries for the implementation, namely *Level* and *ID*. These columns have no connection to the logical model and are not used for query formulation. Column *Level* holds the aggregation level of the dimension's member that is represented by a row of the dimension table. Column *ID* serves as a surrogate key. For example, the Location dimension table has columns *All*, *Country*, *Region*, *State*, *City* and *Store* for the key attribute of each level. The table further has columns *Elevation* and *Mayor* for the attributes of level *City*, and the table has columns *Level* and *ID*. The country France would be represented as a row with value 'Country' in column *Level*, given the fact that France is a country. The value in column *ID* must be

an arbitrary yet unique value, possibly provided by a sequence. Furthermore, the *France* row would have value 'Location' in column *All*, 'France' in *Country*, and NULL values in *State*, *City*, *Elevation*, *Mayor*, and *Store*. The NULL values reflect the fact that some attributes are not applicable to all members of a dimension - the dimension is heterogeneous.

A surrogate key allows for the representation of dimension members at various aggregation levels within a single table. Without a surrogate key, the number of columns that identify a row would vary depending on the level of the represented dimension member. This surrogate key does not necessarily bear meaningful semantics; values for the surrogate key might well come from a sequence. Again, the surrogate key is not used for query formulation, but is only an internal auxiliary for the implementation.

The dimension tables are not normalized. In operational databases, the normalization of tables ensures non-redundant data sets. Large tables with many columns are split into smaller, redundancy-free tables in order to prevent the occurrence of up-date and delete anomalies. Anomalies, however, are not an issue in a data warehouse environment since updates and deletes of existing rows are rather uncommon. The smaller number of tables in a non-normalized schema, on the other hand, reduces the number of table joins and consequently improves query performance. For example, given the Location dimension table, the French city of Paris would be represented as a row with value 'City' in column *Level* and again some unique value in column *ID*. Furthermore, the Paris row would have value 'Paris' in column *City*. The values in other columns would match the values in the France row. The table therefore contains redundancies. Splitting the dimension table into several smaller tables, for example, a separate table for level *Country* and level *City*, would prevent redundancies. This split, however, would also increase the number of joins required in analytical queries.

The analyst formulates queries over the logical model. The BI tool answers these queries using the physical model. Thus, the BI tool must be able to relate the logical model to the physical model. Explicit mappings formalize the relationships between the dimensions in the logical model and the tables in the physical model. For example, a set of feature maps relates the attributes of level City of dimension Location to the corresponding columns of the dimension table (Figure 2.7).

Each hierarchy of a dimension presents a homogeneous view on the dimension table. The view of a hierarchy only contains columns for levels and attributes that are associated with that very same hierarchy. The view also lacks the Level column and does not have a surrogate key. Rather, the view of a hierarchy contains only members at a single aggregation level, namely the most specific level of the hierarchy. Likewise, the homogeneous view of a hierarchy does not contain any NULL values. The existence of NULL values would indicate the existence of heterogeneities within the hierarchy. The existence of heterogeneities in a supposedly homogeneous branch would violate the hetero-homogeneous approach. The heterogeneities should thus be

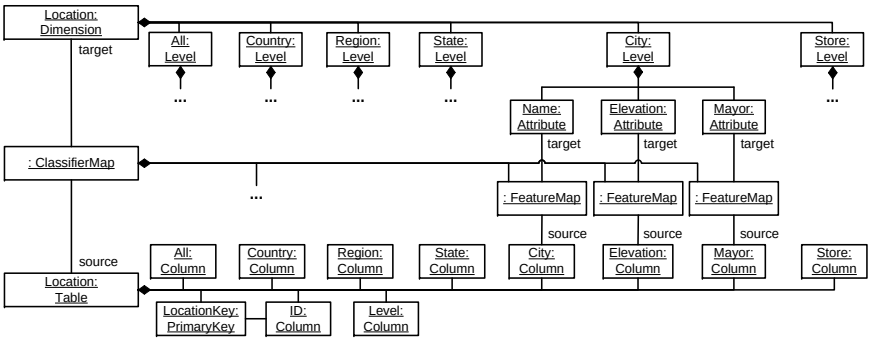


Fig. 2.7. Mapping dimension *Location* to its dimension table (level *City* only)

removed from the hierarchy and their introduction be pushed further down the aggregation hierarchy.

For example, the view for the primary hierarchy of dimension Location includes only columns *All*, *Country*, and *City* for the levels of the hierarchy. The view also includes column *Elevation* for the only attribute that is shared by all members that are part of this hierarchy's *City* level. Notice that including column *Mayor* in the view would yield NULL values since information about a city's mayor is not available for every city; the view would not be homogeneous anymore. Furthermore, the view contains only the *City*-level rows of the dimension table. The following query characterizes the homogeneous view of the primary hierarchy of dimension Location:

```
01 SELECT l.All, l.Country, l.City, l.Elevation
02 FROM   location l
03 WHERE  l.Level = 'City';
```

Similarly, the view of the USA hierarchy of dimension Location includes columns *Country*, *State*, *City*, and *Store* for the levels of the hierarchy. Besides column *Elevation*, the view also includes *Mayor* since information about the mayor is available for every city in the USA. The view contains only the *Store*-level rows of the dimension table. The following query characterizes the homogeneous view of the *USA* hierarchy of dimension Location:

```
01 SELECT l.Country, l.State, l.City, l.Elevation,
02        l.Mayor, l.Store
03 FROM   location l
04 WHERE  l.Level   = 'Store' AND
05        l.Country = 'USA';
```

The aggregation hierarchies map to the homogeneous views instead of the heterogeneous dimension tables. Two types of mappings are especially noteworthy. First, a *listOfValues* mapping allows to retrieving for each attribute the set of values that are actually assigned by the dimension's members to the

respective attribute. For the key attribute of a level, the *listOfValues* mapping retrieves the list of dimension members at this level. Second, an *immediateParent* mapping defines for each level the column that holds the names of the parent member of each member of the respective level. Figure 2.8 illustrates mappings for the *USA* branch of dimension Location. A set of feature maps relates the attributes of the *City* level to the corresponding columns of the *USA* view. Notice that the feature maps are part of different classifier maps which in turn are linked to different structure maps. The feature map that relates attribute Parent of the hierarchy's *City* level to column State of the *USA* view is part of the classifier map that is linked to the *immediateParent* mapping. The other mappings are *listOfValues*.

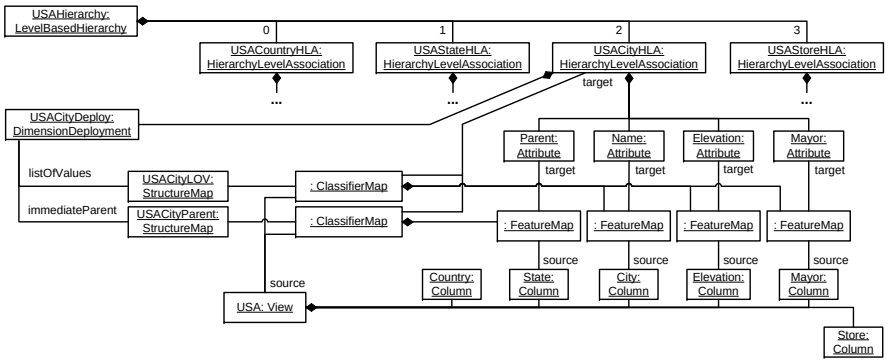


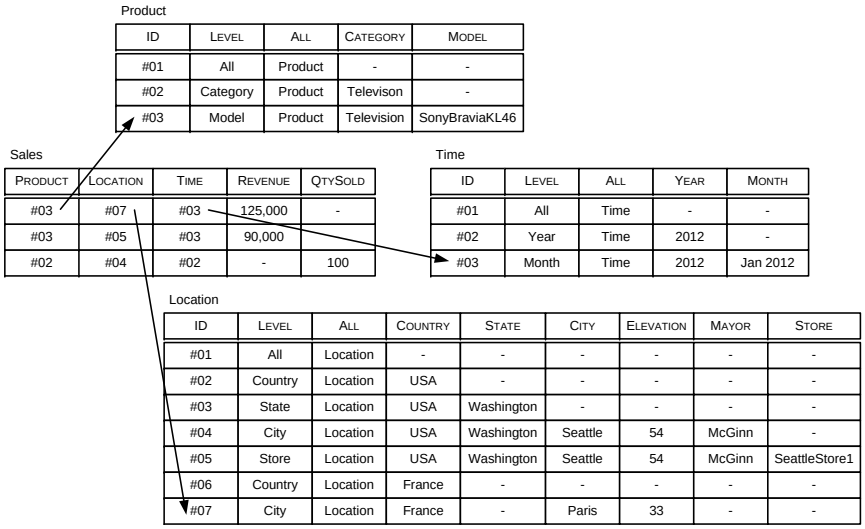
Fig. 2.8. Mapping the *USA* branch of dimension Location to its database view (level *City* only)

2.4.2 Fact Tables

A cube stores its data in a single fact table. This fact table has a column for each of the cube's dimensions and for each of the cube's measures. The dimension columns reference the table of the respective dimension. The measure columns record the values of the measures. Each row of the fact table represents a business event of interest at some level of granularity. The level of granularity is determined by the dimension columns. A fact table can be multi-granular and heterogeneous.

Figure 2.9 illustrates the star schema organization of cube Sales. The Sales fact table links to the surrogate key columns of the dimension tables of Product, Time, and Location. The first row in the Sales fact table records the revenues of the product model SonyBraviaKL46 in Paris in January 2012. It contains a NULL value in *QtySold*. The second row records the revenues of *SonyBraviaKL46* in the SeattleStore1 in January 2012 and contains a NULL value in *QtySold*. Finally, the third row records the sold quantity of products

in the Television category in the city of Seattle in the year 2012. Thus, the Sales fact table is heterogeneous and multi-granular.



Product

ID	Level	All	Category	Model
#01	All	Product	-	-
#02	Category	Product	Televison	-
#03	Model	Product	Television	SonyBraviaKL46

Sales

Product	Location	Time	Revenue	QtySold
#03	#07	#03	125,000	-
#03	#05	#03	90,000	
#02	#04	#02	-	100

Time

ID	Level	All	Year	Month
#01	All	Time	-	-
#02	Year	Time	2012	-
#03	Month	Time	2012	Jan 2012

Location

ID	Level	All	Country	State	City	Elevation	Mayor	Store
#01	All	Location	-	-	-	-	-	-
#02	Country	Location	USA	-	-	-	-	-
#03	State	Location	USA	Washington	-	-	-	-
#04	City	Location	USA	Washington	Seattle	54	McGinn	-
#05	Store	Location	USA	Washington	Seattle	54	McGinn	SeattleStore1
#06	Country	Location	France	-	-	-	-	-
#07	City	Location	France	-	Paris	33	-	-

Fig. 2.9. A sample star schema: Fact table *Sales* and its dimension tables *Product*, *Time*, and *Location*

Each cube region presents a homogeneous, single-granular view on the heterogeneous, multi-granular fact table. Columns of the fact table that represent measures from other cube regions are disregarded. Furthermore, measures available at more specific granularities are rolled up to the required level of granularity, assuming that such a roll-up is possible without summarizability problems, which may not always be the case (see [29]).

For example, the primary cube region of cube Sales contains only measure Revenue. Measure *QtySold*, which is available only for U.S. sales, is disregarded. The level of granularity is ⟨ *Model, Month, City* ⟩, meaning that monthly revenues of product models are recorded by city. Since U.S. sales are tracked at a finer level of granularity, namely by store rather than city, the revenues must be aggregated in order for the view to remain single-granular. The following query characterizes the homogeneous, single-granular view of the primary cube region of cube Sales:

```
01 SELECT   p.Model AS Product, t.Month AS Time,
02          l.City AS Location, SUM(s.Revenue) AS Revenue
03 FROM     Sales s JOIN Product  p ON s.Product  = p.ID
04                  JOIN Time     t ON s.Time     = t.ID
05                  JOIN Location l ON s.Location = l.ID
06 GROUP BY p.Model, t.Month, l.City;
```

Measure *QtySold* is available only for U.S. sales and is recorded at the ⟨ *Category, Year, City* ⟩level of granularity. The view for this cube region selects only the column for measure *QtySold*, disregarding measure Revenue, as it is tracked at a different level of granularity. The view further restricts the number of rows that are considered, selecting only U.S. sales. The following query characterizes the homogeneous, single-granular view of the USA cube region at the ⟨ *Category, Year, City* ⟩granularity level:

```
01 SELECT   p.Category AS Product, t.Year AS Time,
02          l.City AS Location, SUM(s.QtySold) AS QtySold
03 FROM     Sales s JOIN Product  p ON s.Product  = p.ID
04                  JOIN Time     t ON s.Time     = t.ID
05                  JOIN Location l ON s.Location = l.ID
06 WHERE    l.Country = 'USA'
07 GROUP BY p.Category, t.Year, l.City;
```

Measure Revenue is available at a finer granularity for U.S. sales only. While this heterogeneity was eliminated in the primary cube region, a separate view preserves this additional information in the *USA* cube region:

```
01 SELECT   p.Model AS Product, t.Month AS Time,
02          s.Store AS Location, SUM(s.Revenue) AS Revenue
03 FROM     Sales s JOIN Product  p ON s.Product  = p.ID
04                  JOIN Time     t ON s.Time     = t.ID
05                  JOIN Location l ON s.Location = l.ID
06 WHERE    l.Country = 'USA'
07 GROUP BY p.Model, t.Month, l.Store;
```

Depending on which part of the cube is required for the analysis, a different view serves as the source for answering the query. The choice of what view will be used to answer a query depends on the context of the analysis. With the views being homogeneous, the analyst does not have to worry about heterogeneities when formulating the query.

The cube regions map to the homogeneous views instead of the heterogeneous, multi-granular fact table. Figure 2.10 illustrates mappings from the *USA* cube region at granularity level ⟨ *Category, Year, City* ⟩to the corresponding view. The mappings are straight-forward; each attribute or measure from the cube region maps to a column of the same name.

2.5 Implementation

The presented guidelines for modeling logical and physical hetero-homogeneous data warehouse models can be automated. Without machine support, the definition of a logical and physical model in the CWM would be a tedious task for any modeler to complete. Rather, a modeler should rely on the conceptual modeling approach as presented by Neumayr et al. [39] which offers a more

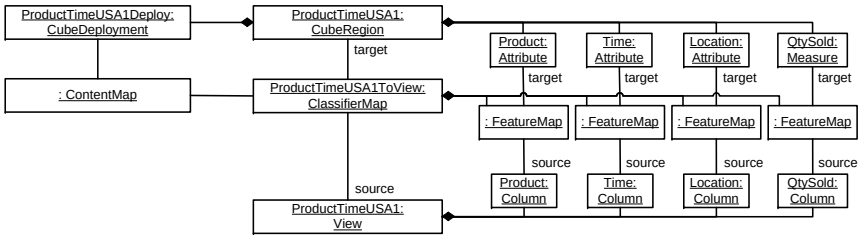


Fig. 2.10. Mapping the *USA* cube region at granularity level ⟨ Category, Year, City ⟩to its database view

intuitive approach towards hetero-homogeneous data warehouse modeling, using m-objects and m-relationships for the representation of dimensions and cubes. From this conceptual model, the CWM logical and physical models can be derived automatically.

Schütz [57] presents a first approach towards the automatic derivation of a logical and physical data model from a conceptual hetero-homogeneous data warehouse model according to Neumayr et al. [39]. The management system of Schütz [57] derives dimension and fact tables from the conceptual model. Furthermore, the system also holds the conceptual model itself for management and analysis purposes in object-relational tables. These tables may be accessed from the outside in order to gain insights about the conceptual model. These insights, in turn, may serve as the basis for the derivation of the logical and physical model in the CWM.

We provide export functionality for the management system of Schütz [57] which consists of transformation routines in Java that extract from the database tables the hetero-homogeneous conceptual model and derive from this model the CWM logical and physical models . For the representation of CWM metadata in Java, our export functionality relies on the Pentaho Metadata libraries. These libraries implement the CWM specification in Java. Furthermore, the Pentaho Metadata libraries allow for the representation of CWM metadata using the XMI language. XMI data is well understood by various BI tools and therefore well-suited for the exchange of models.

2.6 Summary and Future Work

In this chapter, we present an approach for modeling hetero-homogeneous data warehouses in the CWM. Figure 2.11 summarizes the proposed modeling approach. The conceptual model abstracts from the actual implementation and provides a high-level perspective on the modeling domain. M-objects and m-relationships are well-suited for the conceptual representation of hetero-homogeneous dimensions and cubes [39]. In order to use this conceptual model with various BI tools, the conceptual model must be translated into a more

widely understood standard data model. This standard data model is the CWM, with the XMI language as its primary representation format.

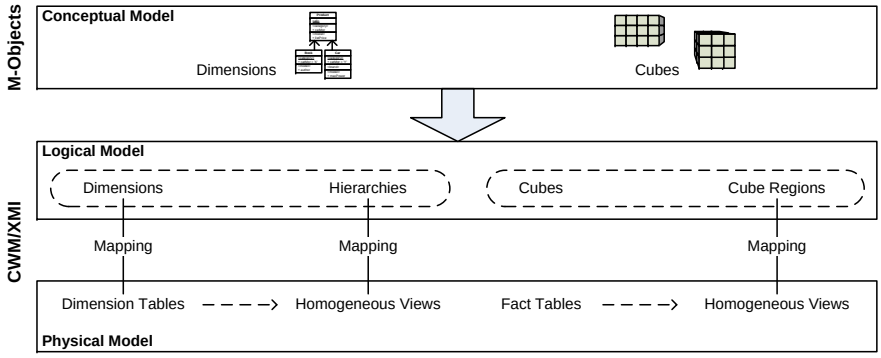


Fig. 2.11. Hetero-homogeneous data warehouse modeling in the CWM in a nutshell

The logical hetero-homogeneous CWM model consists of dimensions and their hierarchies as well as cubes and their cube regions. Hierarchies provide homogeneous schemas for particular branches of a heterogeneous dimension. Likewise, cube regions provide homogeneous schemas for particular portions of a multi-granular, heterogeneous cube. The physical hetero-homogeneous CWM model consists of multiple tables. Dimension and fact tables contain the actual data. The heterogeneous dimension tables in the physical CWM model map to corresponding dimensions of the logical model. The hierarchies of these dimensions, in turn, map to homogeneous views on the heterogeneous dimension tables. Likewise, the cube regions map to homogeneous views on the heterogeneous fact tables. Notice that the cubes in the logical model do not map directly to the fact tables. The CWM provides cube regions for mapping the cubes of the logical model to the tables of the physical model; cube regions are therefore not truly part of the logical model. In order to represent hetero-homogeneous cubes, however, cube regions are employed in the logical model.

Our approach to hetero-homogeneous data warehouse modeling in the CWM facilitates the integration of heterogeneous data sources and allows for the reuse of hetero-homogeneous data models in various BI tools. We stress that our approach to hetero-homogeneous modeling is complementary to the modeling approach of Neumayr et al. [39] and the corresponding prototype presented by Schütz [57]. Rather than providing an alternative modeling approach and implementation, we extend the existing prototype system with a powerful export mechanism. Existing infrastructure may thus be reused without losing the benefits of hetero-homogeneous data models. Still, future work will have to address the following issues:

- Examine the usability of hetero-homogeneous CWM models in various BI tools.
- Define a transformation process compliant with the OMG's model-driven architecture from the hetero-homogeneous conceptual model to the logical and physical CWM models.
- Provide a more concise graphical notation for hetero-homogeneous models in the CWM.

3

Multilevel Business Process Modeling

Conceptual models organize real-world information from a business domain using a human-readable, yet formal representation language. Besides the static data model, many modeling approaches also consider the processes working with these data. In conceptual modeling, abstraction is a common design pattern to more accurately represent a business domain. For static data models, various modeling approaches have been proposed to represent complex multilevel abstraction hierarchies. For process models, though, current modeling approaches do not provide a flexible and powerful formalism for representing complex multilevel abstraction hierarchies. In this chapter, we motivate the need for multilevel business process models and their applications. We present an initial approach towards multilevel business process modeling and sketch future research in this area which can mature into a dissertation.

3.1 Motivation

Conceptual models organize real-world information from a business domain using a human-readable, yet formal representation language. Among the most popular conceptual modeling languages are the entity-relationship (ER) model [6] for database modeling and the much broader Unified Modeling Language (UML) for model-driven software engineering.

Early approaches towards the conceptual modeling of business domains focused merely on the static structure of data while completely neglecting the dynamic aspects of the represented information. Besides the data model, the processes dealing with these data are equally important for running a business. Traditional data modeling approaches, however, could not represent the activities performed on the data. This inability led to the emergence of data-centric business process modeling approaches, for example, object/behavior diagrams [24]. Similarly, the UML was extended with state machine and activity diagrams. More recently, business artifacts [41, 15] have been successfully introduced in both theory and practice. The underlying principle of such

data-centric approaches to business process modeling remains unchanged: Encapsulate in a single object both data and process information.

In conceptual modeling, aggregation is a common design pattern to more accurately represent a business domain. Lower-level, more concrete data objects are collected into higher-level, more abstract aggregates. These aggregates may have a distinct set of attributes and provide metadata for the lower-level data objects. For example, an insurance claim has an amount which is constrained by the coverage indicated in the contract that the claim belongs to. These aggregates may again be collected into aggregates. For example, an insurance contract belongs to a particular insurance category.

Generalization is another common principle to more elegantly represent business domains. The common features of different types of data objects are encapsulated in a common supertype, the generalization. The subtypes are specializations of this supertype. For example, claims under a health insurance contract are filed for a particular medical treatment which is covered by the contract. This information is not applicable for insurance categories other than the health insurance category. The amount, however, is a feature of all claims regardless the category.

Some modeling approaches go beyond ordinary generalization and aggregation relationships to allow for the representation of complex multilevel abstraction hierarchies. Among the most popular approaches with support for multilevel (meta-)modeling are powertypes [45, 12], deep instantiation [2], and materialization [49, 7]. The instances of a powertype are subtypes of another object type, thereby providing metamodeling capabilities [45, p.28]. Gonzalez-Perez and Henderson-Sellers [12] propose a powertype-based approach which abandons traditional two-level instantiation with class and object, using the notion of the "clabject" instead. This class/object duality – a term used by Atkinson and Kühne [2] – is also an important feature of deep instantiation. Deep instantiation allows arbitrary-depth instantiation hierarchies where data objects can instantiate (certain aspects of) other data objects which instantiate other data objects, and so on. Materialization, on the other hand, blurs the boundaries between the aggregation and instantiation relationships.

The multilevel object (m-object), as originally introduced by Neumayr et al. [38], is an expressive and flexible approach to modeling multilevel abstraction hierarchies [40]. The m-object approach combines elements from powertypes, deep instantiation, and materialization. An m-object has several abstraction levels ordered from the most abstract to the most concrete. With each of these levels, an m-object associates a class. These classes are connected by aggregation relationships, constituting an aggregation hierarchy. The introduction of the concretization relationship for m-objects allows for the specialization of entire sub-hierarchies.

In business process modeling, hierarchical modeling commonly refers to the description of the same process at different levels of granularity in order to hide unnecessary details from the user. For example, when handling an insurance claim, the claim is assessed before a verdict is returned. The as-

sessment phase can also be viewed, if desired, in more detail, revealing that this phase consists of an investigation and a decision phase. Many languages allow for the representation of such hierarchical models. UML state machines support sub-states and sub-machines [43]. More recently, Smirnov et al. [61], for example, describe a mechanism for describing the same process at different levels of detail. More prominently, the guard-stage-milestone modeling language introduces such hierarchies for business artifacts [17, 16].

Previous work has identified flexibility [53, 52] and variability [67] as important issues in business process modeling. For example, when handling a health insurance claim, a medical examination takes place during the assessment phase, which is not done for insurance categories other than the health insurances. Case handling [65] provides the actors carrying out the process tasks with a great deal of flexibility. Due to the need for flexibility in business process modeling, rules that govern specialization and change have also been proposed. Behavior-consistent specialization draws from object-oriented data modeling principles for defining precise rules for extending business processes [55, 3]. Closely related to the matter of flexibility is the demand for dynamic changes of models during the execution of a process [54, 66].

The representation of complex abstraction hierarchies in business process modeling might lead to a better alignment of the processes at different hierarchical levels within a company. Existing approaches towards conceptual business process modeling lack a powerful and flexible construct combining aggregation and generalization of processes for the representation of complex abstraction hierarchies. M-objects elegantly combine the advantages of advanced multilevel modeling techniques. M-objects, however, focus on the static aspects of information while neglecting the dynamic aspects. We propose the application of multilevel abstraction principles from data modeling to business process modeling. By extending m-objects with life cycle models, we intend to create a multilevel business process modeling approach.

The remainder of this chapter is organized as follows. In Section 3.2, we present the fundamentals of the proposed approach to multilevel business process modeling. In Section 3.3, we discuss the most important design issues to be solved in realizing our approach to business process modeling. In Section 3.4, we investigate possible applications for multilevel business process models. We conclude with a summary.

3.2 Approach

We propose the multilevel business artifact (MBA) as an extension of the m-object for business process modeling. An MBA, as introduced by Schütz et al. [58], is an m-object which associates with each level of abstraction a single life cycle model. Each of these life cycle models defines the legal execution order of the methods of the class associated with the respective level. Thus, in the spirit of business artifacts and m-objects, an MBA encapsulates

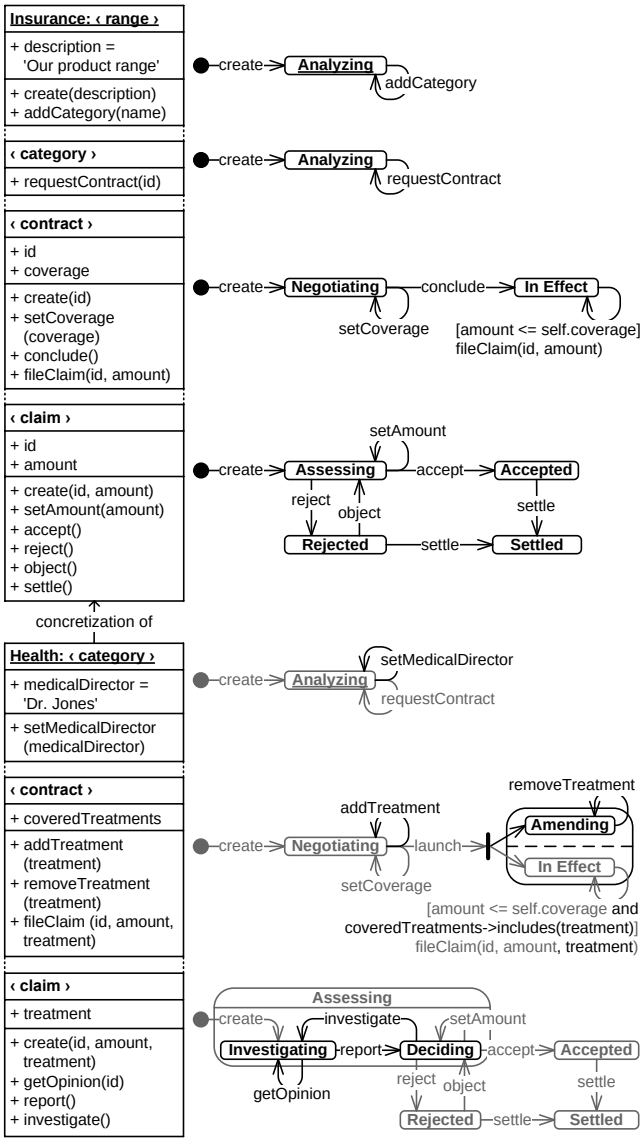


Fig. 3.1. An MBA and one of its concretizations for the management of insurance data

in a single object information about the static and dynamic aspects of multiple levels of abstraction.

Figure 4.1 shows MBAs *Insurance* and *Health* for the management of insurance data. MBA *Insurance*, for example, has levels *range*, *category*, *contract*, and *claim*. Level *range* is the most abstract, level *claim* is the most concrete

level. Each box on the left-hand side represents a class associated with one of these levels. The compartments contain attribute and method definitions, respectively. At level *range*, for example, MBA *Insurance* defines attribute *description* as well as methods *setDescription* and *create*.

We use UML state machines for modeling life cycles. A UML state machine basically consists of states and transitions between these states. With these transitions, methods are associated. The invocation of a method triggers the corresponding transition, causing a change in the current state of the object whose method has been invoked. The method may only be invoked if its object is in the right state. UML state machines are widely-known, are standardized, allow parallel transition paths and sub-states, and provide advanced modeling primitives, for example, pre- and post-conditions for further restricting the triggering of transitions. In principle, though, any other life cycle modeling approach could be used. In particular, Petri nets [64] are a suitable alternative to UML state machines due to their simplicity and expressiveness. Furthermore, the guard-stage-milestone approach might also be suitable for MBAs.

In Figure 4.1, on the right-hand side, for each level, a state machine diagram represents the life cycle model for the corresponding class. MBA *Insurance* at level *contract*, for example, has two states: *Negotiating* and *In Effect*. After creation, MBA *Insurance* is in the *Negotiating* state. In this state, only methods *setCoverage* and *conclude* may be invoked. The invocation of method *conclude* triggers a change to state *In Effect*. In this state, only method *fileClaim* may be invoked. A pre-condition specifies that method *fileClaim* may not be invoked if the value of parameter *amount* is greater than the value of the object's *coverage* attribute. Pre- and post-conditions are specified using the Object Constraint Language (OCL).

An important feature of MBAs (and also m-objects) is their class/object duality. An MBA is an instance of its top-level class and contains only schema information for the more concrete levels. Likewise, an MBA is in a particular state of its top-level life cycle model. For example, in Figure 4.1, MBA *Insurance* is an instance of the class associated with level *range*, assigning a value to attribute *description* and being in the *Analyzing* state.

Multilevel concretization allows for the specialization of the data and process models of entire sub-hierarchies. The concretizing MBA inherits from its parent MBA every level underneath the parent's top level. With each of these levels, the concretizing MBA associates a class which is a specialization of the class associated with the same level by the parent MBA. For example, in Figure 4.1, MBA *Health* is a concretization of *Insurance*. The attributes and methods defined by *Health* are additions to those defined by *Insurance*. Besides the data model, the concretizing MBA may also specialize the state machines inherited from the parent. In Figure 4.1, gray color marks the states and transitions which MBA *Health* inherits from *Insurance*. In addition, at the *contract* level, MBA *Health* has a state *Amending*, parallel to state *In Effect*, and adds a transition to the *Negotiating* state as well as modifying

the pre-condition of an inherited transition. At the *claim* level, MBA *Health* refines the *Assessing* state. We refer to Stumptner and Schrefl for a formal definition and more detailed presentation of behavior-consistent specialization in UML [62].

An MBA represents the common schema of an entire hierarchy of data objects. Using multilevel concretization, an MBA may represent the common schema of an entire branch of the hierarchy represented by some other MBA. The concretizing MBA specializes the data and process models defined by the parent MBA, but only for a particular branch. For this branch, additional information is available. Depending on which part of the hierarchy is relevant for completing a task, the additional information is either considered or disregarded.

3.3 Design Issues

Several issues must be solved when designing an approach for multilevel business process modeling. First, the processes at multiple levels of abstraction interact with each other and must be coordinated. Second, processes should be able to dynamically adapt to different modeling situations, providing flexibility when needed but allowing modelers to intentionally restrict this flexibility when unwanted. Third, information about the actors carrying out the tasks should be incorporated into the model. Fourth, a formalization and implementation should be provided in order to support modelers with their modeling tasks.

3.3.1 Process Interaction and Coordination

The different levels of a single MBA do not exist in isolation but are interdependent, interact with each other, and need coordination. For example, in Figure 4.1, the *range* level of MBA *Insurance* determines the creation of objects at the *category* level. After the execution of the *addCategory* method, an additional category with the given name exists. Furthermore, objects at more abstract levels define invariants for the lower-level objects, thereby constraining the values of certain attributes at these levels. For example, in Figure 4.1, attribute *coverage* at level *contract* of MBA *Insurance* constrains the values of attribute *amount* at the *claim* level. Schütz et al. [58] propose the use of pre- and post-conditions in UML state machines to model interaction between levels as well as their coordination. In this case, in Figure 4.1, the transition at the *Analyzing* state of MBA *Insurance* at the *range* level associated with method *addCategory* would have a post-condition added in order to model interaction between the *range* and the *category* levels. Similarly, the transition at the *Assessing* state at the *claim* level associated with method *setAmount* would have an additional pre-condition added in order to model coordination between the *contract* and the *claim* levels.

In process modeling, m-relationships between MBAs would represent the passing of messages for coordination purposes. The considerations of Schütz et al. [58] concerning process interaction and coordination are limited to levels of the same MBA and MBAs that are in a concretization relationship with each other. MBAs of different hierarchies that are not in a concretization relationship with each other might also be required to interact with each other. Just as multilevel relationships (m-relationships) [38] have been proposed for linking m-objects of different hierarchies, m-relationships could be adapted and extended in order to model process interaction. The implications on behavior-consistent specialization will have to be addressed. Future work could build on the findings of Yongchareon et al. [68] about specialization of synchronization dependencies in artifact-centric process models.

3.3.2 Flexibility and Change

Multilevel concretization is not an atemporal operation but is itself a meta-process. A modeler creating a concretization of some other MBA specializes the schema of the concretization by adding new attributes and methods as well as abstraction levels. From a more technical perspective, the concretization process is a sequence of invocations of reflective methods on MBAs. These reflective methods allow changes in the schema during runtime. For example, an MBA could have a reflective function *addAttribute* with parameters *name*, *dataType* and *level* which adds to the MBA a new attribute at the specified level with a specified name and data type. Similarly, reflective methods for altering the life cycle models must be provided. This approach towards flexibility is similar to the idea of the two-tier artifact-centric framework proposed by Liu et al. [30] with process design entities and business entities, but on multiple levels of abstraction and incorporating behavior-consistent specialization.

The concretization mechanism for MBAs provides modelers with flexibility when capturing the different information needs and tasks of the various departments and branches within a company. Künzle et al. [25], for example, identify flexibility as one fundamental requirement for business process modeling approaches. In some cases, however, it might be desirable to limit the freedom of a modeler. For example, legal regulations might impose that a certain process must not to be altered. Likewise, company policy might require certain things to remain unchanged. By referencing the reflective methods in the life cycle models of MBAs, the flexibility of modelers provided by the standard semantics of multilevel concretization might be restricted. The specialization of the schema could be restricted to certain phases in the life cycle of an object For example, states and transitions might only be added to a life cycle model in a *Developing* state. Pre- and post-conditions could be used to restrict the use of reflective functions. For example, the specialization of the schema could be restricted to specific levels. This approach has been explained by Schütz et al. [58] in more detail.

The addition of new levels during the concretization of MBAs remains an open question. The m-object approach allows adding new levels in a concretizing m-object provided that the relative order of the levels remains unchanged. Previous work [58] on MBAs has not addressed this issue. In a dynamic context, when introducing a new level in a concretizing MBA, issues arise when pre- and post-conditions reference immediate parents of children.

3.3.3 Actors

A business process model identifies tasks on business objects which are carried out by actors. In general, MBAs represent business objects and contain information about the tasks that use and modify these objects. In order to represent information about actors in MBA-based business process models, two approaches seem suitable. First, from a database perspective, actors are also data objects with a distinct set of attributes. These data objects may be represented by MBAs and could be linked to other MBAs using an adapted form of m-relationships. Second, MBAs might be integrated into existing modeling approaches, thereby replacing or extending the native constructs for representing business objects.

Even though MBAs, in general, represent business objects, modeling actors as MBAs is not at all a paradox. Most actors in a business domain, for example, employees and, at a more abstract level, departments, have a distinct set of attributes. Thus, from a database perspective, these actors are data objects just as any other. Indeed, m-objects have been used to model both objects and actors, for example, products and the producing companies [38].

MBAs could be extended with activity diagrams in order to represent actors at multiple levels of abstraction. Each activity diagram would then correspond to a specific process as seen from the actor's point of view. These activity diagrams would reference methods of the processed MBAs used to complete the tasks.

Using MBAs for modeling actors requires these actors to be of a multilevel nature, which is, in practice, not always the case. An alternative to modeling multilevel actors would be the integration of MBAs in existing modeling approaches. For example, the business object modeling primitive of the Business Process Model and Notation (BPMN) could probably be replaced by the MBA. Similarly, subject-oriented business process modeling (S-BPM) [9] could be complemented by MBAs. An advantage of S-BPM is the tool support for creating models and transforming these models into executable code.

Neither of the proposed approaches has been realized yet. In practice, though, a combination of both approaches could prove most useful.

3.3.4 Metamodel and Implementation

We propose the use of the O-Telos modeling language and its implementation, ConceptBase [23], for the specification of the MBA metamodel and the

implementation of a development tool. O-Telos has a simple and thus very flexible data model: Everything in O-Telos is an object and objects may be related with each other. An object may specialize and/or instantiate any object, including itself. Constraints and queries are formulated using first-order logic expressions. The implementation of O-Telos, ConceptBase, has already been successfully employed for business process models [22].

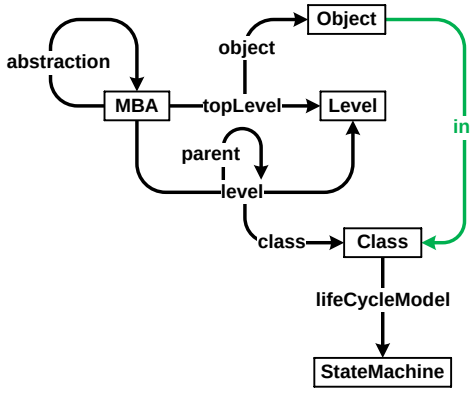


Fig. 3.2. The MBA metamodel in O-Telos

Figure 4.4 illustrates the MBA metamodel using a common graphical notation for O-Telos. The boxes represent objects. The arrows between these boxes represent relationships between objects. An arrow labeled *in* represents an instantiation relationship. Note that each relationship is also an object and may therefore be related to other objects.

Each MBA is an instance of object *MBA*. When instantiating *MBA*, each relationship from *MBA* to another object is also instantiated. The source object of such a relationship must be an instance of *MBA*; the target object must be an instance of the object that is targeted by the instantiated relationship.

Each level is an instance of object *Level*. Levels have an identity independent from MBAs. Therefore, different MBAs may reference the same level objects. MBAs establish the hierarchical order of the levels and associate class definitions with these levels. Therefore, rather than attaching the parent level to an instance of *Level* directly, the parent level is associated with the relationship between this level and the MBA. Similarly, the class definition for a level is associated with this relationship.

Each MBA has a single, designated top level. The relationship *topLevel* between objects *MBA* and *Level* may only be instantiated once per MBA. Furthermore, relationship *topLevel* has a relationship with *Object*. An instance of a relationship *topLevel* has a relationship to an instance of *Object*. Every class referenced by an MBA is a specialization of *Object*.

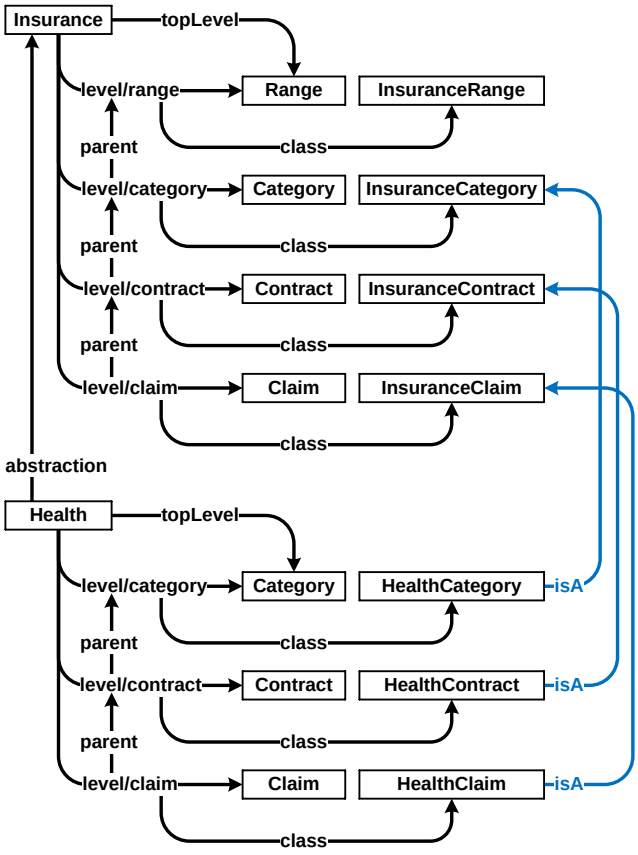


Fig. 3.3. MBAs *Insurance* and *Health* in O-Telos

Figure 3.3 illustrates MBAs *Insurance* and *Health* from the previous examples (Figure 4.1) using the MBA metamodel in O-Telos. The instantiation relationships with the objects defined in the metamodel are not shown. Objects *Insurance* and *Health* are instances of *MBA*. Objects *Range*, *Category*, *Contract*, and *Claim* are instances of *Level*. The other objects are instances of *Class* and specializations of *Object*. Arrows with label *isA* represent specialization relationships. Notice that the relationship *level* from the MBA metamodel (Figure 4.4) is instantiated multiple times.

Due to the class/object duality of MBAs, the association of the top level with an object is not shown in Figure 3.3. The representation of the class/object duality of MBAs is a particular challenge and worth a closer look. In Figure 3.4, object *Insurance* is an instance of *MBA* and *InsuranceRange*. Object *InsuranceRange* is the class associated with level *Range*. Level *Range*, in turn, is the top level of *InsuranceRange*. The object associated with this top level is the MBA itself. This approach to the class/object duality has one

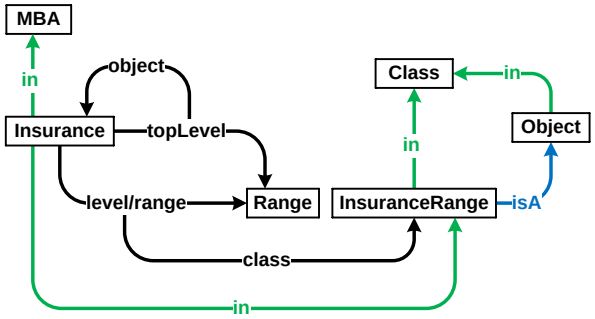


Fig. 3.4. MBA *Insurance* as an instance of its top-level class

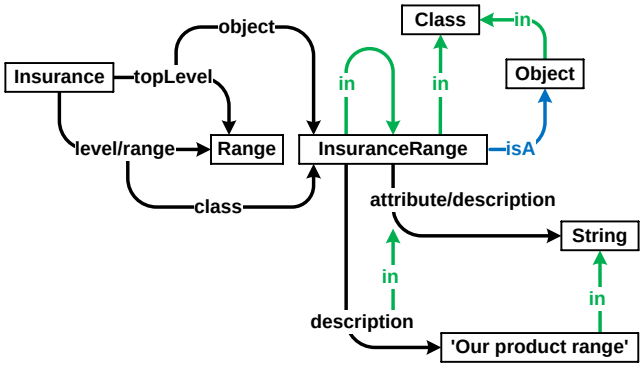


Fig. 3.5. MBA *Insurance* associating a clabject with its top level

major advantage: Even without knowledge of the MBA metamodel, a program can use an MBA just like any other instance of some class. If other levels are not important for a particular program, there is no need to rewrite this program in order to cope with MBAs.

Figure 3.5 presents an alternative representation of the class/object duality of MBAs. In this case, the top level of an MBA associates a clabject, that is, an object presenting characteristics of both class and object. This is possible in O-Telos since an object can be an instance of itself. Thus, an object can define its own class schema. This approach can be combined with the previous approach to class/object duality, yielding an object which defines its own class schema for the top level.

The dynamics of MBAs must also be represented in the O-Telos language. Figure 3.6 illustrates how state machines are handled. Any life cycle model is an instance of object *StateMachine*. For example, in Figure 3.6, object *RangeLifeCycle* represents the life cycle model for the *range* level of MBA *Insurance*. An instance of a life cycle model is a life cycle trace. A life cycle trace references concrete states of objects, not "state classes", and concrete transitions, not "transition classes". For example, in Figure 3.6, the calling

of method *AddCategory_1* with argument *'Health'* triggers transition *AnalyzingToAnalyzing_1* from *Analyzing_1* to *Analyzing_2*. Objects *Analyzing_1* and *Analyzing_2* are both instances of the *Analyzing* state. The life cycle trace is not created by the modeler but is created dynamically by the system, triggered by method calls. The dynamic generation of a life cycle trace could be done in ConceptBase using event condition action (ECA) rules. Together with a life cycle trace, useful information could be stored which could later be analyzed, for example the start and end time of an object being in a particular state.

3.4 Applications

Multilevel business process models could be employed in operational and strategic information systems alike. A multilevel business process management system could improve the alignment of business processes within a company. Multilevel business process intelligence, on the other hand, could improve the quality of performance analysis for business processes.

3.4.1 Business Process Management

Frequently, each hierarchical unit within a company has its own business process solution. For example, the decision processes of management are completely separated from the processes of the production or research departments. The processes at different levels of the hierarchy are not related with each other although a better alignment of these processes could improve performance. A central repository of MBAs may support modelers in coordinating the diverse, traditionally isolated solutions within a company. This central repository contains a multilevel business process model which provides an integrated view of the various processes within the company and their interdependencies.

The central MBA repository should not be limited to a mere documentation of the existing processes within a company, though. Rather, the repository could provide an execution environment on its own or complement the existing solutions. Two alternative approaches are available for the integration of the central MBA repository with existing software. These alternatives are non-exclusive and could occur simultaneously. First, the business logic is executed locally. Prior to the execution, however, the local program sends a request to the server in order to verify whether the execution is permissible. Second, the business logic is executed on the server as a stored procedure. In this case, the locally executed program is concerned only with establishing the connection to the database server and calling the corresponding stored procedure. ConceptBase, for example, allows the execution of Prolog programs stored on the database server.

3.4.2 Business Process Intelligence

Business intelligence (BI) technologies provide the means for analyzing the performance of a company and thus support business executives in their decisions. Among the most important BI technologies is the data warehouse. A data warehouse organizes strategic, time-variant, and subject-oriented data about a company, gathered from the company's various operational data sources [21]. A typical example for data stored in a data warehouse are sales figures and their evolution over time. Various modeling approaches have been proposed for data warehouses, for example, the dimensional fact model [10]. Such modeling approaches commonly organize data about business events of interest within an n-dimensional space commonly referred to as (hyper-)cube, with each dimension of this space being hierarchically organized. Measures associated with these business events can be aggregated along the dimension hierarchies. For example, sales figures could be organized in a cube with a product, time, and location dimension. Each cell of this cube represents the sale of a product at a particular point in time in a particular location. A typical measure for sales are revenues. With a location dimension consisting of levels city and country, revenues could be viewed by city or country alike.

An important issue in data warehouse modeling is the representation of heterogeneities. For example, sales in the United States could be aggregated on a federal state level, a political entity not present in other countries. Likewise, due to fiscal reasons, an additional measure could be recorded for sales in the France only. M-objects have been adapted for representing such heterogeneities in data warehouse models. Neumayr et al. [39] introduce hetero-homogeneous dimensions and cubes modeled with m-objects and m-relationships. Hetero-homogeneous dimensions and cubes feature a globally homogeneous schema but allow for the introduction of additional aggregation levels and measures, respectively, in well-defined partitions of the data. Using m-objects, depending on which partition of the data is analyzed, the analyst may leverage a different schema with additional information.

With the rise of business process re-engineering and the desire to constantly improve existing processes, performance measurement for business processes has gained in popularity. Business process intelligence refers to the application of BI technologies to the analysis of business process performance [13]. Numerous approaches to multidimensional modeling have been developed for the analysis of business processes [34, 5, 63].

The hetero-homogeneous approach to data warehouse modeling could be adapted for business process analysis. Different processes could be associated with different measures and aggregated along different aggregation levels. Given the insurance example, the performance of claim handling could be measured by the time it takes to process a claim. In addition, for claims under a health insurance contract, the time for the medical examination could be recorded. The proof-of-concept prototype system for the management of hetero-homogeneous data warehouses [57] could be extended in order to sup-

port performance measurement for business processes, leading to the development of a hetero-homogeneous process warehouse.

3.5 Summary and Future Work

This chapter is a proposal for a dissertation about multilevel business process modeling. While research on data modeling has recognized the need for powerful abstraction mechanisms with metamodeling capabilities, research on abstraction hierarchies in data-centric business process models is more or less limited to hiding details from the user. A flexible approach on multilevel business process modeling, however, could lead to a better alignment of processes at different levels of abstraction. Both operational and analytical applications could benefit from the explicit consideration of multilevel abstraction hierarchies in business process models. A dissertation on multilevel business process modeling will have to address the following design issues:

- Representing interactions of MBAs by adapting m-relationships for MBAs;
- Representing flexible process models and dynamic change, especially the introduction of additional abstraction levels;
- Representing information about the actors who carry out the tasks associated with an MBA;
- Providing an implementation of MBAs as a foundation for a multilevel business process management system and a hetero-homogeneous process warehouse.

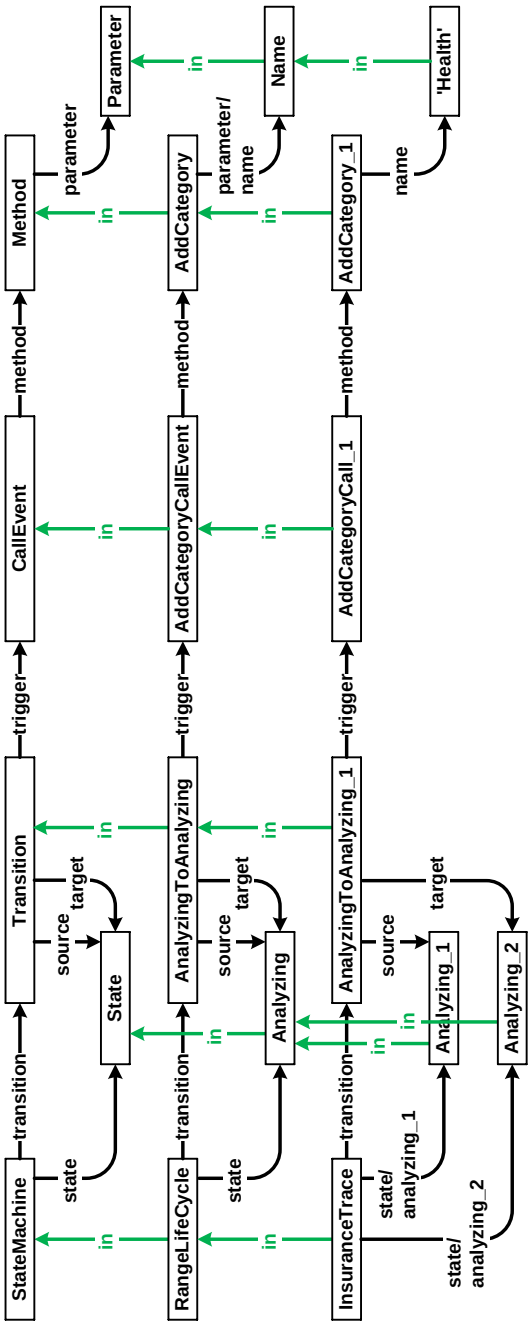


Fig. 3.6. The dynamics of MBAs represented in O-Telos: Metamodel, model, and model instance

4

Multilevel Business Artifacts

The representation of many real-world scenarios in conceptual models benefits from the use of multilevel abstraction hierarchies. Product models, for example, are typically grouped into product categories which, in turn, constitute the company's range of products. Multilevel abstraction hierarchies often reflect the organizational structure of a company and the different information needs of the various departments. Current modeling techniques, however, lack extensive support for the representation of multilevel abstraction hierarchies in business process models. The explicit consideration of multilevel abstraction hierarchies in business process models might improve the alignment of processes across different organizational entities. In this chapter, we introduce the concept of the multilevel business artifact (MBA) for representing multilevel abstraction hierarchies of both data and process models. An MBA encapsulates in a single object the data and process models of various levels, thereby expanding consequently the idea of business artifacts to the realm of multilevel abstraction hierarchies.

4.1 Introduction

In many modeling situations, data objects are arranged in multilevel abstraction hierarchies. In such hierarchies, data objects at lower levels of abstraction are collected into more abstract, higher-level objects. These higher-level objects provide an alternative view of the represented problem domain, carrying information that is not present in, yet related to, the lower-level objects. Product models, for example, are typically grouped into product categories which, in turn, constitute the company's range of products. A product model typically has a list price. The actual selling price, however, might be influenced by the tax rate attached to the corresponding product category.

Multilevel abstraction hierarchies can support differing information needs within a company. Different departments process data objects at different

levels of abstraction but the processes dealing with these data objects are interdependent. For example, top management decides which product categories the company should offer whereas the marketing and production departments are concerned with individual products. The decisions of top management, although they concern data on a different level of abstraction, affect the marketing and production departments. If top management decides to focus, for example, luxury goods rather than budget products, marketing may adjust its pricing strategy for individual products and production may shift priorities to rigorous quality management rather than low-cost production.

The explicit consideration of multilevel abstraction hierarchies in process models might improve the alignment of processes across different organizational entities. Current modeling techniques lack extensive support for the representation of multilevel abstraction hierarchies in business process models. Note that the use of multilevel abstraction hierarchies in business process models as presented in this chapter differs from other approaches with abstraction which represent the same process at varying levels of detail.

In this chapter, we introduce the multilevel business artifact (MBA) for representing multilevel abstraction hierarchies of both data and process models. We base the MBA approach on multilevel objects (m-objects) which offer a compact and flexible formalism in conceptual models for the representation of multilevel abstraction hierarchies with possibly heterogeneous levels [38, 40]. M-objects, however, focus mainly on the static aspects of the conceptual model, lacking any information about the execution order of methods. An MBA, on the other hand, encapsulates in a single object the data and process models of various levels, thereby expanding the idea of a business artifact – a "chunk of information that can be used to run a business" [41] – to multilevel abstraction hierarchies.

The remainder of this chapter is organized as follows. In Section 4.2, we introduce the MBA approach for a compact representation of interdependent processes at various levels of abstraction. In Section 4.3, we present multilevel concretization for increased modeling flexibility. In Section 4.4, we formally define the MBA metamodel and its semantics in terms of UML. In Section 4.5, we discuss the potential benefits of the MBA approach in relation to existing work. In Section 4.6, we conclude with a summary and outline future research on multilevel business process modeling.

4.2 Multilevel Business Artifact

In multilevel abstraction hierarchies, data objects at higher levels of abstraction are considered aggregates of more concrete, lower-level objects. Nevertheless, the data objects at each level have their own distinct features. Consider, for example, the data model of a fictitious travel agency with a wide range of guided tours. Within various categories, the company offers several tour packages. A tour package represents a proposed set of travel activities over

a number of days. Based on these tour packages, the travel agency organizes particular trips with a specific start and end date.

Besides their static features, data objects at different levels of abstraction have interdependent life cycles. For example, a travel company's range of guided tours is constantly being reassessed, resulting in the addition of new tour categories. Likewise, within each tour category, the development of new packages is regularly initiated. And each tour package undergoes a development phase before the launch puts the package on offer. A tour package may be selected for organizing a trip only when the package is on offer. Note that attributes of higher-level data objects can constrain the processes at lower levels of abstraction. For example, a trip's start and end date are constrained by the corresponding package's number of days.

Multilevel objects (m-objects), as introduced by Neumayr et al. [38, 40], encapsulate in a single object information about an entire abstraction hierarchy. An m-object defines a number of abstraction levels and their hierarchical order as well as a class for each of these levels. The different classes are associated by aggregation relationships along the abstraction level hierarchy. M-objects, however, omit the dynamic aspects of the represented information.

A multilevel business artifact (MBA) accounts for the dynamic aspects of multilevel abstraction hierarchies. Basically, an MBA is an m-object extended with life cycle models where each abstraction level has a single life cycle model that defines the legal execution order of the methods of the class.

Figure 4.1 illustrates MBA *Tour* (and MBA *CityTour*) for the management of tour data. Each box on the left-hand side represents a class. Within each box, the top compartment contains, in arrow brackets, the name of the level the class is associated with. MBA *Tour* defines classes, connected by dotted lines, for levels *range*, *category*, *package*, and *trip*, where *range* is the most abstract and *trip* is the most concrete level. MBA *CityTour* defines classes for levels *category*, *package*, and *trip*, where *range* is the most abstract and *trip* is the most concrete level. The other compartments contain attribute and method definitions, respectively.

We use UML state machine diagrams [43] for modeling object life cycles. A state machine consists of states and transitions between these states. Transitions are triggered by events. For an MBA, the triggering events are *call events* raised by the invocation of a method. Thus, the state machine models the legal execution order of methods.

In Figure 4.1, for example, the state machine diagram at the *package* level of MBA *Tour* restricts changes in the number of days (*nrOfDays*) to the development phase. Consequently, method *setNrOfDays* may be invoked only on objects in the *Developing* state. The invocation of method *launch* puts an object in state *On Offer*. In this state, only method *requestTrip* may be invoked. Note that this method cannot be invoked when the object is in the development phase.

For each transition in a state machine, pre- and post-conditions may be specified. These pre- and post-conditions relate the life cycle models of dif-

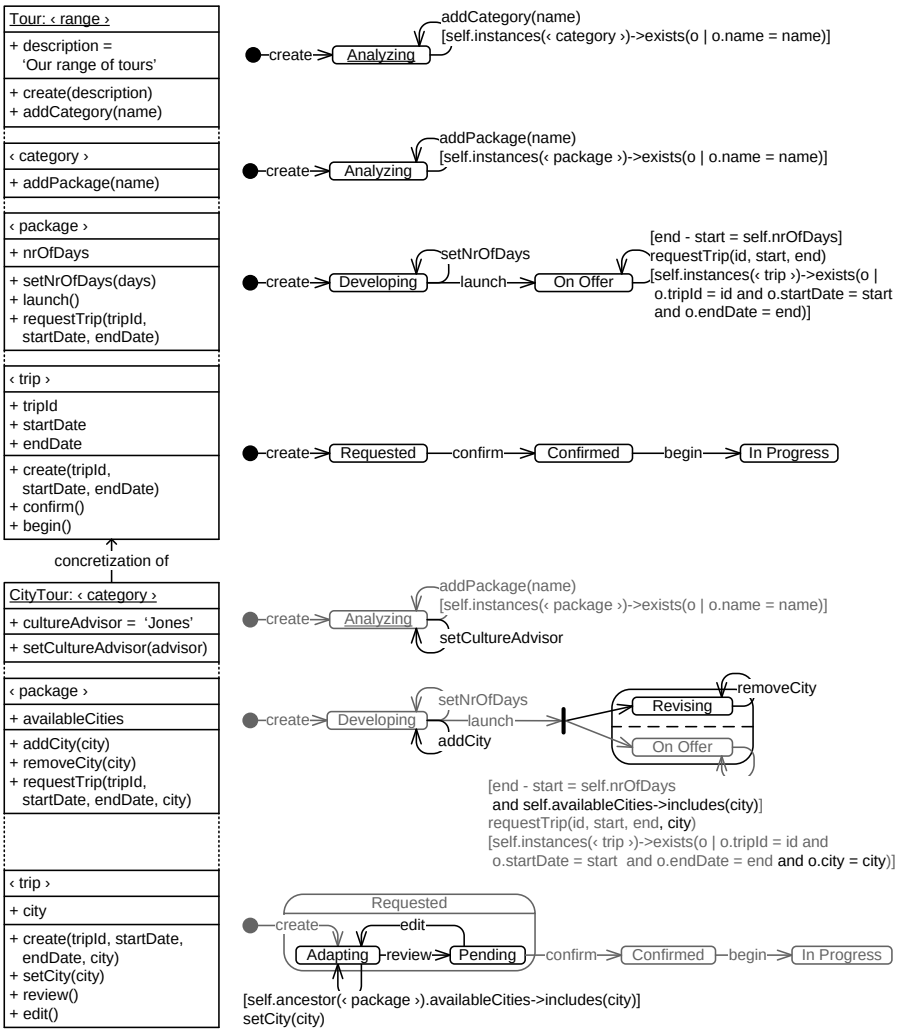


Fig. 4.1. An MBA and one of its concretizations for the management of tour data

ferent abstraction levels. For example, at the *range* level, the post-condition of the recursive transition of the *Analyzing* state relates levels *range* and *category* by requiring that, after the execution of method *addCategory*, the set of objects instantiating the class associated with level *category* must contain an object with the specified name. An attribute *name* is implicitly defined for every MBA and is assumed to be unique. Similar conditions relate levels *category* and *package* as well as levels *package* and *trip*. After the invocation of the *addPackage* method at the *category* level, a new MBA at the *package* level exists. After the invocation of the *requestTrip* method at the *package* level, a

new MBA at the *trip* level exists. Pre- and post-conditions may also be used to relate different levels by defining invariants. For example, the number of days at the *package* level constrains the selection of the dates at the *trip* level.

An MBA instantiates the class at the single most abstract level. An MBA therefore assigns values to attributes associated with the top level and executes the corresponding life cycle model. MBA *Tour* in Figure 4.1 instantiates the class at the *range* level, assigning a value to attribute *description*, and sets the current state in the life cycle to *Analyzing*. MBA *CityTour*, on the other hand, instantiates the class at the *category* level.

4.3 Multilevel Concretization

An MBA presents characteristics of both class and object. An MBA defines classes which are arranged in an aggregation hierarchy according to their levels of abstraction. The instances of these classes are again MBAs. Therefore, an MBA describes the schema of a (sub-)hierarchy. An MBA, however, is also part of the abstraction hierarchy. Given this duality of class and object, a special type of relationship is needed to describe an abstraction hierarchy with MBAs. This relationship type is multilevel concretization, adapted from m-objects [38].

Through concretization, MBAs are collected into aggregates. For example, in Figure 4.1, MBA *CityTour* is a concretization of MBA *Tour*. The single most abstract level of *CityTour* is *category*, which is the second level of MBA *Tour*. MBA *CityTour* instantiates the class at its top level, *category*, and becomes part of the range of guided tours represented by MBA *Tour*. MBA *CityTour* is among the instances of the class defined by MBA *Tour* at the *category* level.

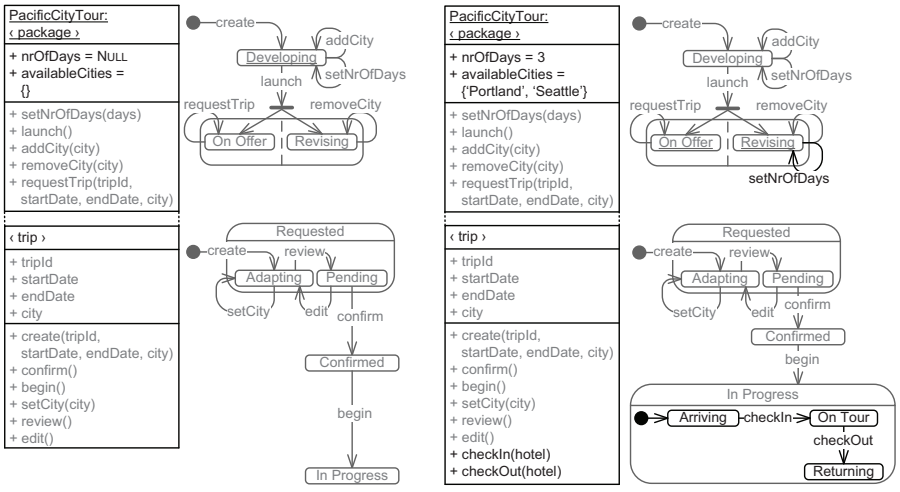
The concretizing MBA must instantiate the class associated with the second level of its abstraction, the concretized MBA. Consequently, the concretizing MBA becomes part of the set of all instances of this class, and is thus part of the aggregate object represented by its abstraction. In this sense, multilevel concretization presents semantics of both instantiation and aggregation.

Multilevel concretization, however, is not merely a mechanism for instantiation and aggregation. The main purpose of concretization is to support modeling flexibility through specialization. For each level an MBA shares with its parent, the concretizing MBA specializes the class that the parent MBA associates with this level. For example, in Figure 4.1, MBA *CityTour* specializes the classes defined by its parent, MBA *Tour*, by adding attributes and methods. Note that the inherited features are not shown in this diagram.

Concretization does not restrict specialization to the static aspects of an MBA. Life cycle models may also be specialized. A concretizing MBA may add additional transitions and states or refine existing states in an inherited life cycle model. The semantics of life cycle specialization is based on existing work which has extensively studied behavior-consistent specialization of life cycle

models [62, 55]. For example, the state machine diagram at the *category* level of MBA *CityTour* has an additional transition with respect to the inherited life cycle model. The state machine diagram at the *package* level, on the other hand, has an additional state, *Revising*, parallel to *On Offer*. At the *trip* level, the state machine diagram presents a refined state *Requested*. Note that gray color marks inherited states and transitions throughout this chapter.

An MBA describes the common, global schema of a hierarchy which, through concretization, may be specialized for a particular sub-hierarchy. The concretizing MBA describes a specialized schema that is valid only for a sub-hierarchy. For this sub-hierarchy, the specialized schema of the concretizing MBA becomes the common schema which, again, may be further specialized by other MBAs through concretization. An employee can select the appropriate MBA which best fits the information demands of a particular task, without the overhead of unnecessary information.



(a) Developing **(b)** {On Offer, Revising}

Fig. 4.2. A concretization of MBA *CityTour* in different life cycle states

Concretization is not a one-shot activity. Rather, concretization itself is an incremental process. Consider, for example, MBA *PacificCityTour*, a concretization of *CityTour*, in Figure 4.2. Initially, MBA *PacificCityTour* has only the inherited class and life cycle models, is in the *Developing* state, and has NULL values assigned to the top-level attributes (Figure 4.2a). During the development phase, values are assigned to attributes *nrOfDays* and *availableCities* and the inherited class and life cycle models are specialized. The invocation of method *launch* terminates the *Developing* phase and puts MBA *PacificCityTour* simultaneously in the states *On Offer* and *Revising* (Figure 4.2b). In this state, the attributes at the *package* level already have values

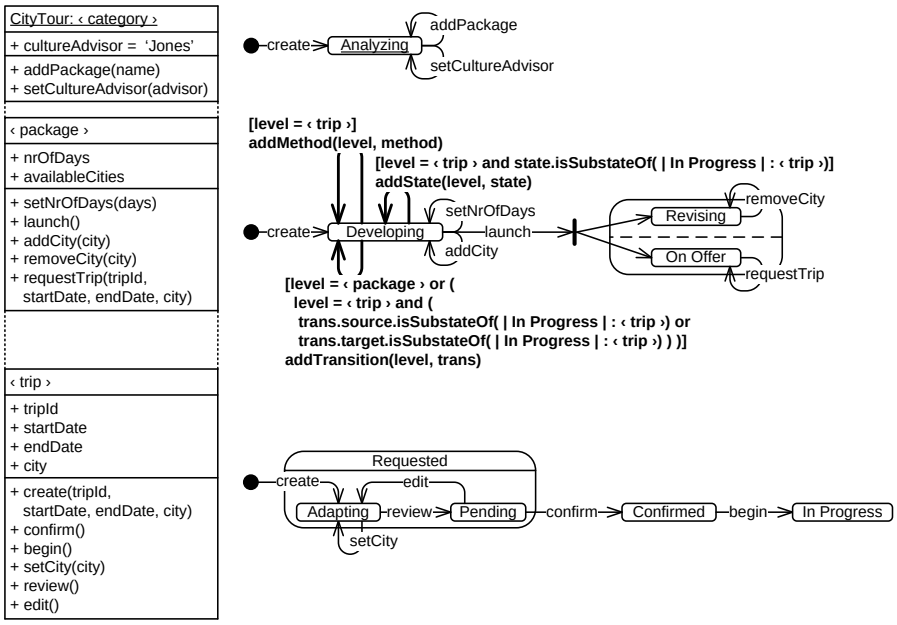


Fig. 4.3. MBA *CityTour* with meta-process model elements (in boldface)

assigned and the class and life cycle models differ from the inherited models. For the *PacificCityTour* package, the number of days can now also be altered after the product launch. At the *trip* level, the process of what happens after beginning the trip has been further clarified. Note that inherited pre- and postconditions have been omitted in Figure 4.2.

Since concretization itself is a process, an MBA may also account for meta-process activities in order to control local changes made to the imposed business process models. Every MBA implicitly has pre-defined reflective methods which enable changes of class and life cycle models at runtime. By default, the reflective methods of an MBA can be invoked in any state. In this case, classes and life cycle models can be extended as long as the semantics of class and life cycle specialization are obeyed. The explicit mention of reflective methods in the life cycle model, however, allows the modeler to further restrict specialization and thus deliberately limit flexibility.

Figure 4.3 shows an alternative version of MBA *CityTour* with reflective methods in one of its life cycle models. Inherited pre- and postconditions have been omitted. According to the life cycle model for the *package* level, methods, states, and transitions can be added only in the *Developing* state. Using pre-conditions, flexibility can be constrained even further. In this example, new methods may only be added to the model associated with the *trip* level. Likewise, new states may only be added for the *trip* level and only as substates of *In Progress*. Transitions may be added for levels *package* and *trip*.

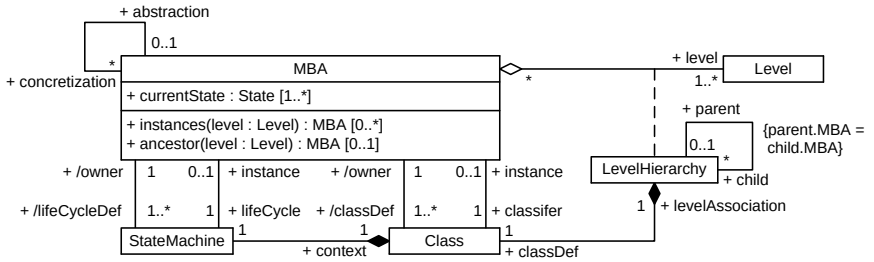


Fig. 4.4. The MBA metamodel

For the *trip* level, transitions may only be added if they come from or lead to a sub-state of *In Progress*.

4.4 Metamodel and UML Semantics

The Meta-Object Facility [42] and the Unified Modeling Language (UML) [43] provide the framework for the formal definition of the MBA approach. The Object Constraint Language (OCL) [44], in turn, allows for the specification of additional consistency criteria. Figure 4.4 illustrates the MBA metamodel. Figures 4.5 and 4.6 describe adapted consistency criteria from m-objects [38] using OCL constraints.

An MBA references several levels which exist independently from an MBA. Attached to each link between an MBA and a level is a reference to a parent level. Note that the same level may have different parent levels, depending on the MBA. The links between an MBA and its levels together with the records of the parent levels constitute a level hierarchy. In this hierarchy, a level cannot be its own ancestor (Constraint 1). Furthermore, an MBA has a single most abstract level, the top level, which has no parent level within the MBA (Constraint 2).

An MBA defines classes, one for each associated level. Each link between an MBA and a level references a class. Each class, in turn, has a UML state machine as life cycle model. For convenience, an MBA directly references each class and life cycle model. Therefore, the associations between *MBA* and *Class* as well as between *MBA* and *StateMachine* are derived from the level hierarchy (as indicated by a slash before the role names in Figure 4.4).

The classes associated with the levels of an MBA are instantiated by MBAs. For this reason, class *MBA* is a specialization of *InstanceSpecification* from the UML Kernel [43]. From this class, *MBA* inherits its association to *Classifer*, referenced by the role name *classifier*, which *MBA* restricts to *Class*. This *classifier* references the class that is associated with the MBA's top level (Constraint 3). Similarly, *lifeCycle* references the top-level life cycle model. The *instances* method of an MBA retrieves all instances of a class

Constraint 1: Acyclic level hierarchy

```
context LevelHierarchy
 def: ancestor : Set(LevelHierarchy) = self->closure(parent)
 inv: not self.ancestor->collect(level)->includes(self.level)
```

Constraint 2: Single top level

```
context MBA
 def: topLevel : Collection(Level) = self.LevelHierarchy
   ->select( h | h.parent.oclIsUndefined() )->collect(level)
 inv: self.topLevel->size() = 1
```

Constraint 3: Instantiate the top-level class and life cycle model

```
context MBA
 inv: self.classDef->select(c | c.levelAssociation.level
   = self.topLevel->any(true))->includes(self.classifier)
 inv: self.lifeCycleDef->select(l | l.context.levelAssociation.level
   = self.topLevel->any(true))->includes(self.lifeCycle)
```

Constraint 4: All instances at a given level

```
context MBA::instances(level : Level) : Set(MBA)
 body:
   let levelClass : Class = self.classDef
     ->any(c | c.levelAssociation.level = level) in
   levelClass.allInstances()
```

Constraint 5: Ancestor at a given level

```
context MBA::ancestor(level : Level) : MBA
 body:
   let ancestors : Set(MBA) = self->closure(abstraction) in
   ancestors->any( o | o.topLevel->any(true) = level )
```

Constraint 6: Instantiate second level of parent

```
context MBA
 inv: self.abstraction.LevelHierarchy->exists (
     h | h.level = self.topLevel->any(true) and
         h.parent.level = self.abstraction.topLevel->any(true)
 )
 or self.abstraction.oclIsUndefined()
```

Fig. 4.5. Consistency criteria for the MBA metamodel (Constraints 1 to 6)

associated with a particular level using the pre-defined *allInstances* operation (Constraint 4). The result of this query includes instances of sub-classes.

The recursive one-to-many association of class *MBA* represents concretization. Besides its immediate parent, an MBA will frequently access ancestors at more abstract levels. Method *ancestor* of class *MBA* retrieves the ancestor having a particular top level (Constraint 5).

Constraints 6-10 must be satisfied in order for an MBA to be a consistent concretization of its parent. First, the top level of the concretizing MBA must be a child of the top level of the parent MBA (Constraint 6). Second, the concretizing MBA contains every level of the parent MBA from the concretizing MBA's top level downwards (Constraint 7). Third, the relative order of the levels is the same in both the concretizing MBA and its parent MBA (Constraint 8). Fourth, the class and life cycle models defined by the concretizing MBA are specializations of the corresponding models in the parent MBA (Constraints 9 and 10).

In this chapter, we do not formally define the notions of class and life cycle specialization. Specialization of classes is extensively described by the UML standard. The notion of life cycle specialization, on the other hand,

Constraint 7: Inheritance of levels

```
context MBA
 inv: self.abstraction.LevelHierarchy->select (
   h | h.ancestor->exists( p | p.level = self.topLevel->any(true) ) or
       h.level = self.topLevel->any(true) )->forAll (
   h | self.level->includes(h.level) ) or self.abstraction.oclIsUndefined()
```

Constraint 8: Stability of level order

```
context MBA
 inv: self.level->asSet()->intersection( self.abstraction.level->asSet() )
   ->forAll( l1, l2 | (
       self.abstraction.LevelHierarchy->exists( h |
         h.level = l1 and h.ancestor->exists( i : LevelHierarchy | i.level = l2 )
       ) implies
       self.LevelHierarchy->exists(h |
         h.level = l1 and h.ancestor->exists( i : LevelHierarchy | i.level = l2 )
       ) ) and (
       self.LevelHierarchy->exists(h |
         h.level = l1 and h.ancestor->exists( i : LevelHierarchy | i.level = l2 )
       ) implies
       self.abstraction.LevelHierarchy->exists( h |
         h.level = l1 and h.ancestor->exists( i : LevelHierarchy | i.level = l2 )
       ) )
   ) or self.abstraction.oclIsUndefined()
```

Constraint 9: Specialization of class models

```
context MBA
 inv: self.level->asSet()->intersection( self.abstraction.level->asSet() )
  ->forAll( l | self.classDef->any(c : Class | c.levelAssociation.level = l)
     ->collect(generalization)->includes(
       self.abstraction.classDef->any( c : Class | c.levelAssociation.level = l )
 ) ) or self.abstraction.oclIsUndefined()
```

Constraint 10: Specialization of life cycle models

```
context StateMachine
 inv: self.context.generalization->forAll( g : Class |
   self.isSpecializationOf(g.StateMachine)
)
```

Fig. 4.6. Consistency criteria for the MBA metamodel (Constraints 7 to 10)

depends largely on the modeling formalism employed. In general, a specialized process model may refine states by adding sub-states; it may also add parallel paths. We refer to Stumptner and Schrefl [62] for a formal definition of behavior-consistent specialization in UML state machine diagrams. These authors [55] also provide a more in-depth analysis of behavior-consistent specialization of life cycle models, including rules for consistency checking. We refer to Grossmann et al. [14] for an analysis of the complexity of checking behavior-consistent specialization.

Finally, Figure 4.7 illustrates the UML semantics of the running example used throughout this chapter. This UML diagram consists of several aggregation hierarchies of classes. The topmost class of each hierarchy is a singleton class with one instance. The other classes are specialized, and thus relate the different aggregation hierarchies. Together with the instances on the right-hand side, each of these aggregation hierarchies corresponds to an MBA. Note that instances are shown with their name followed by their type, separated by a colon and both underlined.

The leftmost aggregation hierarchy in Figure 4.7 consists of classes *TourRange*, *TourCategory*, *TourPackage*, and *TourTrip*. Each of these classes is

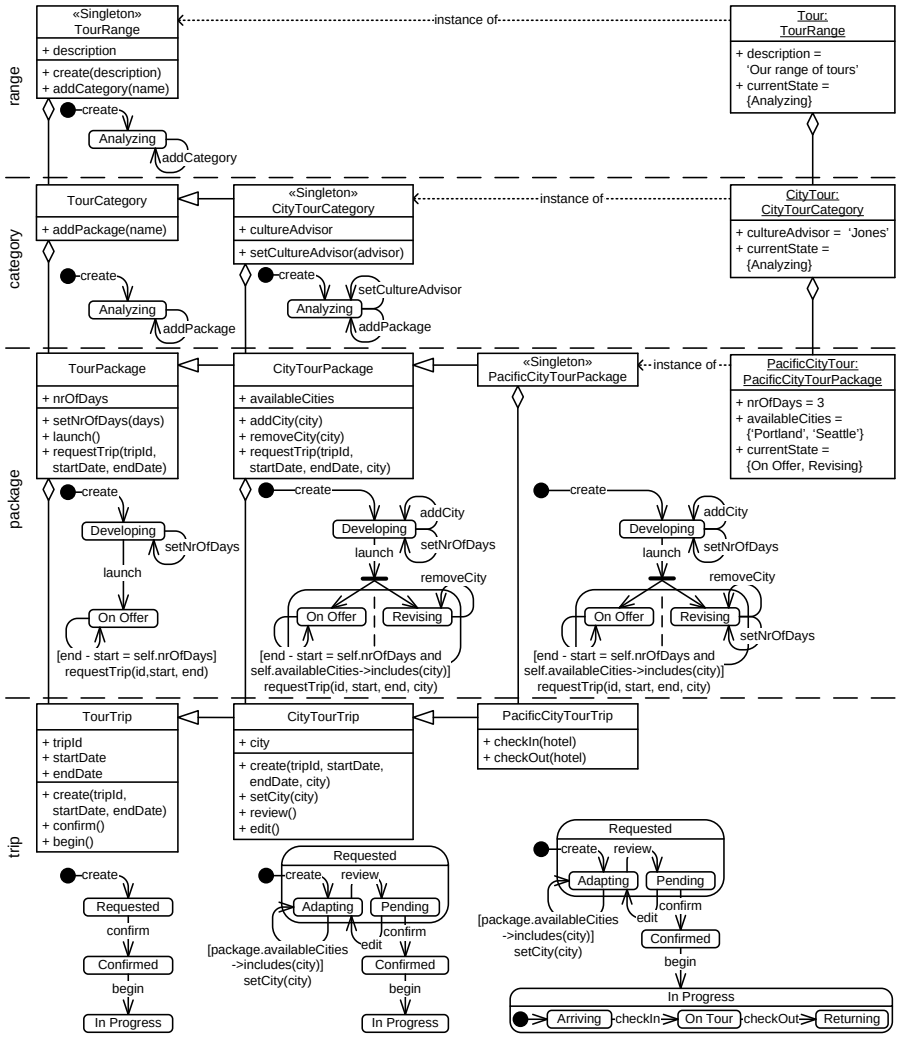


Fig. 4.7. UML class and state machine diagrams for the management of tour data

associated with a life cycle model. This aggregation hierarchy corresponds to the class and life cycle definitions of MBA *Tour*.

The aggregation hierarchy in the middle consists of classes *CityTourCategory*, *CityTourPackage*, and *CityTourTrip*. These classes are specializations of classes *TourCategory*, *TourPackage*, and *TourTrip*, respectively. This aggregation hierarchy corresponds to the class and life cycle definitions of MBA *CityTour*, which is a concretization of MBA *Tour*.

The rightmost aggregation hierarchy consists of classes *PacificCityTourPackage* and *PacificCityTourTrip*. These classes are specializations of classes

TourPackage and *TourTrip*, respectively. This aggregation hierarchy corresponds to the class and life cycle definitions of MBA *PacificCityTour*, which is a concretization of MBA *CityTour*.

4.5 Related Work

An early predecessor of current data-centric approaches to business process modeling are object/behavior diagrams [24]. The principle of describing the structure and dynamics of data in a single object has been successfully advanced by the business artifact approach [41]. The MBA approach as presented in this chapter adopts this idea and expands it to multiple levels of abstraction. Instead of describing only a single abstraction level, an MBA combines structure and dynamics of multilevel data in a single object. This view on business process model abstraction differs from other approaches, for example by Smirnov et al. [61] or the guard-stage-milestone modeling approach for business artifacts [16]. These approaches describe the same process at different levels of detail. The MBA approach, on the other hand, considers interdependent processes of objects at various levels of abstraction.

In recent years, the interest in flexibility and dynamic change in data-centric business process models has been increasing. Flexible process models allow a company to adapt to the changing business environment [53]. In order to better suit a particular business situation, process models should be allowed to change [66], especially when dealing with less-structured processes [25]. Similarly, Weidlich et al. [67] stress the importance of managing variants of process models which exist in a company due to differing requirements across departments. Furthermore, in order to support flexibility in business process modeling, process models should be allowed to adapt dynamically during their execution [54]. The MBA approach offers these kinds of flexibility through the concretization mechanism. Moreover, through its reflective capabilities, the MBA approach allows the modeler to explicitly represent and constrain dynamic change in the model.

4.6 Summary and Future Work

This chapter is a first introduction to multilevel business process modeling. We introduced the concept of the MBA which, in the spirit of the business artifact approach, encapsulates in a single object data and life cycle models of an abstraction hierarchy. Through concretization, these data and life cycle models can be specialized for particular sub-hierarchies. Future work will address several issues. First, in order to enable message passing between MBAs that are not in a concretization relationship, a new relationship type should be introduced. Second, actors should be incorporated explicitly within the model. Third, an implementation of the MBA approach should support modelers in creating a central repository of multilevel business process models.

References

1. A. Abelló, J. Samos, and F. Saltor. YAM^2: A multidimensional conceptual model extending UML. *Information Systems*, 31(6):541–567, 2006.
2. C. Atkinson and T. Kühne. The essence of multilevel metamodeling. In *UML'01*, volume 2185 of *LNCS*, pages 19–33. 2001.
3. T. Basten and W. van der Aalst. Inheritance of behavior. *Journal of Logic and Algebraic Programming*, 47(2):47–145, 2001.
4. A. Bauer and H. Günzel. *Data-Warehouse-Systeme: Architektur, Entwicklung, Anwendung*. dpunkt Verlag, Heidelberg, third edition, 2009.
5. F. Casati, M. Castellanos, U. Dayal, and N. Salazar. A generic solution for warehousing business process data. In *Proc. VLDB '07*, pages 1128–1137, 2007.
6. P. Chen. The entity-relationship model – toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.
7. M. Dahchour, A. Pirotte, and E. Zimányi. Materialization and its metaclass implementation. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1078–1094, Sept. 2002.
8. R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems, 5th Edition*. Addison-Wesley-Longman, 2000.
9. A. Fleischmann and C. Stary. Whom to talk to? a stakeholder perspective on business process development. *Universal Access in the Information Society*, 11:125–150, 2012.
10. M. Golfarelli, D. Maio, and S. Rizzi. The dimensional fact model: A conceptual model for data warehouses. *International Journal of Cooperative Information Systems*, 1998.
11. M. Golfarelli, S. Rizzi, and E. Turricchia. Modern software engineering methodologies meet data warehouse design: 4wd. In A. Cuzzocrea and U. Dayal, editors, *DaWaK*, volume 6862 of *LNCS*, pages 66–79. Springer, 2011.
12. C. Gonzalez-Perez and B. Henderson-Sellers. A powertype-based metamodelling framework. *Software & System Modeling*, 5(1):72–90, 2006.
13. D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M.-C. Shan. Business process intelligence. *Computers in Industry*, 53(3):321–343, 2004.
14. G. Grossmann, M. Schrefl, and M. Stumptner. Design for service compatibility – behavioural compatibility checking and diagnosis. *Software and Systems Modeling*, pages 1–27. http://dx.doi.org/10.1007/s10270-012-0229-0.

15. R. Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *OTM'08*, volume 5332 of *LNCS*, pages 1152–1163. 2008.
16. R. Hull, E. Damaggio, R. De Masellis, F. Fournier, M. Gupta, F. T. Heath, S. Hobson, M. H. Linehan, S. Maradugu, A. Nigam, P. N. Sukaviriya, and R. Vaculín. Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events. In *Proceedings of the Fifth ACM International Conference on Distributed Event-Based Systems (DEBS'11)*, pages 51–62, 2011.
17. R. Hull, E. Damaggio, F. Fournier, M. Gupta, F. T. Heath, S. Hobson, M. H. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, and R. Vaculín. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *Proceedings of the Seventh International Workshop on Web Services and Formal Methods (WS-FM'10)*, pages 1–24, 2010.
18. C. A. Hurtado and C. Gutierrez. Handling structural heterogeneity in OLAP. In R. Wrembel and C. Koncilia, editors, *Data Warehouses and OLAP: Concepts, Architectures and Solutions*, pages 27–57. IGI Global, 2007.
19. C. A. Hurtado, C. Gutiérrez, and A. O. Mendelzon. Capturing summarizability with integrity constraints in OLAP. *ACM Transactions on Database Systems*, 30(3):854–886, 2005.
20. N. Iftikhar and T. B. Pedersen. Schema design alternatives for multi-granular data warehousing. In P. G. Bringas, A. Hameurlain, and G. Quirchmayr, editors, *DEXA (2)*, volume 6262 of *LNCS*, pages 111–125. Springer, 2010.
21. W. H. Inmon. *Building the data warehouse*. Wiley, fourth edition, 2005.
22. M. Jeusfeld. A deductive view on process-data diagrams. In *Proceedings of the Fourth IFIP WG 8.1 Working Conference on Method Engineering (ME'11)*, volume 351 of *IFIP AICT*, pages 123–137. 2011.
23. M. Jeusfeld, M. Jarke, and J. Mylopoulos. *Metamodeling for method engineering*. MIT Press, Cambridge, 2009.
24. G. Kappel and M. Schrefl. Object/behavior diagrams. In *Proceedings of the Seventh International Conference on Data Engineering (ICDE'91)*, pages 530–539, 1991.
25. V. Künzle, B. Weber, and M. Reichert. Object-aware business processes: Fundamental requirements and their support in existing approaches. *International Journal of Information System Modeling and Design*, 2(2):19–46, 2011.
26. C. Kurze and P. Gluchowski. Computer-aided warehouse engineering (CAWE): Leveraging mda and adm for the development of data warehouses. In M. Santana, J. N. Luftman, and A. S. Vinze, editors, *AMCIS*, page 282. Association for Information Systems, 2010.
27. J. Lechtenbörger and G. Vossen. Multidimensional normal forms for data warehouse design. *Inf. Syst.*, 28(5):415–434, 2003.
28. W. Lehner, J. Albrecht, and H. Wedekind. Normal forms for multidimensional databases. In M. Rafanelli and M. Jarke, editors, *SSDBM*, pages 63–72. IEEE Computer Society, 1998.
29. H.-J. Lenz and A. Shoshani. Summarizability in OLAP and statistical data bases. In Y. E. Ioannidis and D. M. Hansen, editors, *SSDBM*, pages 132–143. IEEE Computer Society, 1997.
30. R. Liu, F. Y. Wu, F. Pinel, and Z. Shan. A two-tier data-centric framework for flexible business process management. In *Proc. AMCIS '12*, 2012.

31. S. Luján-Mora, J. Trujillo, and I.-Y. Song. A UML profile for multidimensional modeling in data warehouses. *Data & Knowledge Engineering*, 59(3):725–769, 2006.
32. E. Malinowski and E. Zimányi. Hierarchies in a multidimensional model: From conceptual modeling to logical representation. *Data & Knowledge Engineering*, 59(2):348–377, 2006.
33. E. Malinowski and E. Zimányi. *Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications*. Springer, 2008.
34. S. Mansmann, T. Neumuth, and M. Scholl. Multidimensional data modeling for business process analysis. In *ER'07*, volume 4801 of *LNCS*, pages 23–38. 2007.
35. J.-N. Mazón, J. Lechtenbörger, and J. Trujillo. A survey on summarizability issues in multidimensional modeling. *Data Knowl. Eng.*, 68(12):1452–1469, 2009.
36. J.-N. Mazón and J. Trujillo. An mda approach for the development of data warehouses. *Decision Support Systems*, 45(1):41–58, 2008.
37. F. Melchert, A. Schwinn, C. Herrmann, and R. Winter. Using reference models for data warehouse metadata management. In D. Khazanchi and I. Zigurs, editors, *AMCIS*, page 25. Association for Information Systems, 2005.
38. B. Neumayr, K. Grün, and M. Schrefl. Multi-level domain modeling with m-objects and m-relationships. In *Proceedings of the Sixth Asia-Pacific Conference on Conceptual Modeling (APCCM'09)*, pages 107–116, 2009.
39. B. Neumayr, M. Schrefl, and B. Thalheim. Hetero-homogeneous hierarchies in data warehouses. In *Proceedings of the Seventh Asia-Pacific Conference on Conceptual Modeling (APCCM'10)*, pages 61–70, 2010.
40. B. Neumayr, M. Schrefl, and B. Thalheim. Modeling techniques for multi-level abstraction. In R. Kaschek and L. Delcambre, editors, *The Evolution of Conceptual Modeling*, volume 6520 of *LNCS*, pages 68–92. Springer, Heidelberg, 2011.
41. A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.
42. Object Management Group. *OMG Meta Object Facility (MOF), Core Specification, version 2.4.1*, 2011. http://www.omg.org/spec/MOF/2.4.1/.
43. Object Management Group. *OMG Unified Modeling Language (OMG UML), Superstructure, version 2.4.1*, 2011. http://www.omg.org/spec/UML/2.4.1/.
44. Object Management Group. *OMG Object Constraint Language (OCL), version 2.3.1*, 2012. http://www.omg.org/spec/OCL/2.3.1/.
45. J. Odell. *Advanced object-oriented analysis and design using UML*, chapter Power types, pages 23–32. Cambridge University Press, 1998.
46. J. Pardillo and J.-N. Mazón. Model-driven development of OLAP metadata for relational data warehouses. *Computer Standards & Interfaces*, 34(1):189–202, 2012.
47. T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. A foundation for capturing and querying complex multidimensional data. *Inf. Syst.*, 26(5):383–423, 2001.
48. F. Pinet and M. Schneider. A unified object constraint model for designing and implementing multidimensional systems. *J. Data Semantics*, 13:37–71, 2009.
49. A. Pirotte, E. Zimányi, D. Massart, and T. Yakusheva. Materialization: A powerful and ubiquitous abstraction pattern. In *Proceedings of the Twentieth International Conference on Very Large Data Bases (VLDB'94)*, pages 630–641, 1994.
50. J. Poole, D. Chang, D. Tolbert, and D. Mellor. *Common Warehouse Metamodel developer's guide*. Wiley Publishing, Indianapolis, 2003.

51. N. Prat, J. Akoka, and I. Comyn-Wattiau. A uml-based data warehouse design method. *Decision Support Systems*, 42(3):1449–1473, 2006.
52. G. Redding, M. Dumas, A. ter Hofstede, and A. Iordachescu. A flexible, object-centric approach for business process modelling. *Service Oriented Computing and Applications*, 4(3):191–201, 2010.
53. M. Reichert, S. Rinderle-Ma, and P. Dadam. Flexibility in process-aware information systems. In *ToPNoC II*, volume 5460 of *LNCS*, pages 115–135. 2009.
54. S. Rinderle, M. Reichert, and P. Dadam. Correctness criteria for dynamic changes in workflow systems – a survey. *Data and Knowledge Engineering*, 50(1):9–34, 2004.
55. M. Schrefl and M. Stumptner. Behavior-consistent specialization of object life cycles. *ACM ToSEM*, 11(1):92–148, 2002.
56. C. Schütz. Hetero-homogene Data Warehouses. *BI-Spektrum*.
57. C. Schütz. Extending data warehouses with hetero-homogeneous dimension hierarchies and cubes: A proof-of-concept prototype in Oracle. Master's thesis, Johannes Kepler University, Linz, Austria, 2010.
58. C. Schütz, L. Delcambre, and M. Schrefl. Multilevel business artifacts. In *Proceedings of the First Workshop on Data- and Artifact-centric Business Process Management (DAB'12)*, 2012. To appear. https://sites.google.com/site/dabworkshop2012/home.
59. C. Schütz, M. Schrefl, and L. Delcambre. Multilevel business process modeling: motivation, approach, design issues, and applications. In *Proceedings of the Fifth Workshop for Ph.D. Students in Information and Knowledge Management (PIKM'12)*, 2012.
60. C. Schütz, M. Schrefl, B. Neumayr, and D. Sierninger. Incremental Integration in Data Warehouses: The Hetero-Homogeneous Approach. In *Proceedings of the ACM Fourteenth International Workshop on Data Warehousing and OLAP (DOLAP'11)*, 2011.
61. S. Smirnov, H. Reijers, and M. Weske. A semantic approach for business process model abstraction. In *CAiSE'11*, volume 6741 of *LNCS*, pages 497–511. 2011.
62. M. Stumptner and M. Schrefl. Behavior consistent inheritance in UML. In *ER 2000*, volume 1920 of *LNCS*, pages 451–530. 2000.
63. A. Sturm. Supporting business process analysis via data warehousing. *Journal of Software: Evolution and Process*, 24(3):303–319, 2012.
64. W. van der Aalst. The application of Petri Nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.
65. W. van der Aalst, M. Weske, and D. Grnbauer. Case handling: A new paradigm for business process support. *Data and Knowledge Engineering*, 53(2):129–162, 2005.
66. B. Weber, M. Reichert, and S. Rinderle-Ma. Change patterns and change support features - enhancing flexibility in process-aware information systems. *Data and Knowledge Engineering*, 66(3):438–466, 2008.
67. M. Weidlich, J. Mendling, and M. Weske. A foundational approach for managing process variability. In *CAiSE'11*, volume 6741 of *LNCS*, pages 267–282. 2011.
68. S. Yongchareon, C. Liu, and X. Zhao. A framework for behavior-consistent specialization of artifact-centric business processes. In *BPM '12*, volume 7481 of *LNCS*, pages 285–301. 2012.