

Another issue is to be able to describe spatial and temporal predicates that associated to relationship types will constraint the spatiality and/or the lifecycle of the related object types. In MADS these predicates support semantic enhancements for relationships, i.e., topological and synchronization, which can be freely combined with the other enhancements described in Sect. 2.3, that is, aggregation, transition, and generation. This approach allows maximal flexibility for modeling complex real-world phenomena.

In MADS we adopted a conceptual approach for coping with the continuous view of space and time, where a phenomenon that varies over a spatial and/or temporal extent is represented as a function having as domain the underlying extent and whose range is the set of values measuring the phenomenon. This conceptual view hides the particular implementation technique used for representing continuous phenomena in a computer, which pertains to the logical level. From the users' perspective this facilitates the comprehension of the essential spatial and temporal characteristics of an application, and in particular this allows to easily represent phenomena such as moving points and areas. We have shown that attributes are used for representing continuous phenomena in the database. According to whether the phenomenon varies over a spatial and/or a temporal extent this leads to space-varying, time-varying, or space- and time-varying attributes. Finally, spatial predicates have been extended to the case where constrained geometries are time varying.

2.4 Supporting Multiple Perceptions and Multiple Representations

The data model specifications that we have discussed so far address classical and spatio-temporal data modeling requirements. In this section we analyze and develop specifications for supporting multiple perceptions and multiple representations. We first discuss why this is essential in good data management and then proceed with the analysis of how multiple perceptions and representations can be supported from the data modeling perspective.

2.4.1 Rationale for Multiple Representations

Databases store representations of identifiable real-world phenomena that are of interest to a given set of applications. Which representations are to be stored is determined during the database design process, where application requirements are analyzed (in terms of which real-world entities, links between entities, and properties of entities and links are desirable) and turned into a description of formalized data structures. A known difficulty in database design is to reconcile the divergent requirements of the applications sharing the same database. While the real world is supposed to be unique, its representation depends on the intended purpose. Each application has its own perception of the real world, and its data processing tasks

lead to specific requirements, both in terms of what information is to be kept and in terms of how the information is to be represented. Different applications that have overlapping concerns about real-world phenomena normally require different representations of the same phenomena. Differences may arise in all facets that make up a representation, including the following.

- What information is kept: the set of objects and links of interest is determined by application goals. An organization may store all relevant data in a single database, described by a single schema, but many different *sub-schemas* may be defined, each one supporting a given application. Sub-schemas may share object and relationship types, as they may share object and relationship instances, as well as having specific types or instances.
- How information is described: even if the same object (relationship) type is of interest for several applications, the properties to describe the object (relationship) may well change among applications. The sex of an employee, for instance, may be relevant for management of maternity allowances, but should not be relevant for determining the salary.
- How information is organized (in terms of data structures): a reservoir in a water management system may be seen as an object of its own, or as an attribute of a catchment area object.
- How information is coded: dimensions measured in inches versus centimeters, coordinates of a point in one reference system versus coordinates of the same point in another reference system.
- What constraints, processes, and rules apply: for the staff management application every employee must be assigned to a department, while for the financial management application employees supported by external funds are seen as not being assigned to a department.
- How information is presented: the same information may be extracted in many different ways, such as different orderings in an employee list (by name, by department, ...).
- What are the associated spatial and temporal frameworks: the data today versus the data yesterday, or the data for this county versus the data for that county.

To support such heterogeneity of perceptions, the database has to be able to hold *multiple representations* of the same real-world phenomenon (cf. Fig. 2.44). Notice that multiple representations may be needed also within a single perception, as even a single application may need multiple representations at different levels of detail (for instance, looking at John as a faculty or as a person, or looking at an enterprise as a single entity or as a collection of its component departments).

Current DBMSs partially support multiple, somehow different, representations through the view mechanism. Starting from base representations, the view mechanism allows deriving new representations (views) from the already defined representations. Object-oriented DBMSs provide is-a hierarchies (based on the use of system-generated object identifiers) that allow multiple representations as progressive refinement from a generic representation to more specialized representations. Both approaches, however, are known to be insufficient (in terms of expressive

power, user-friendliness, and practicality) to provide full flexibility in multiple-representation support. The most important weakness is that the concept of *perception*, i.e., the knowledge of which representations together form a consistent whole for an application, is not supported per se. It is only indirectly supported through another mechanism, the definition of *access rights*, i.e., by granting each application access to all views and only those views that belong to its perception.

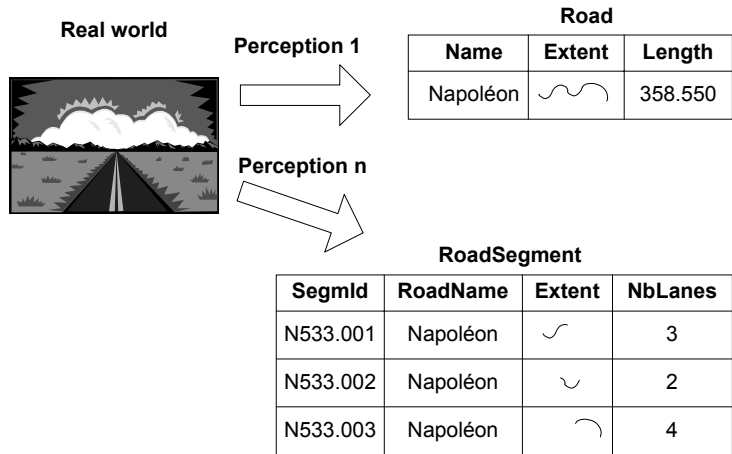


Fig. 2.44. Different perceptions of the same reality, in this case leading to different representations.

The view mechanism relies on a two-step approach. First, all information items that are needed are represented in the database logical schema. Second, personalized data structures (the views) are defined as queries on the stored data structures and previously defined views. Views are new, virtual representations, built through filter and combine operations, providing either an alternative representation for existing objects (object-preserving views) or defining new objects composed from existing objects (object-generating views). Views cannot contain new data that are not derivable from the existing data.

Relational DBMSs support a powerful view mechanism, whose main limitation is in the fact that views that do not rely on a 1:1 mapping between tuples in the view and underlying tuples in the database cannot be used to update the database, because of the inherent ambiguity of such updates. Similarly, to avoid possible inconsistencies, most commercial object-oriented DBMSs restrict views to object preserving ones, defined through simple filtering (selections). The important difference between views and multi-representation as proposed in MADS is that views are isolated and anonymous units of data description. They do not make up a schema that consistently represents a given perception of the world of interest. Further, views miss the information that identifies the perceptions they convey. An application perception can be reconstructed from access control specifications, assuming each transaction is only allowed access to the data structures (basic

structures or views) that correspond to its perception. However, mixing access rights concerns with representational concerns is conceptually disturbing and potentially harmful.

What a view-based or otherwise centralized representation mechanism can definitely not support is the case where different application viewpoints are not derivable from each other (irreducible viewpoints). Assume a hospital information system, such that patients are identified by medical teams based on a patient number inscribed on a bracelet that the patient always carries, and the same patients are identified by the administrative staff based on a social security number. If the two viewpoints do not share other information (such as name and birth date) that could provide a common identification scheme, when the patient leaves the hospital two different update operations have to be made for the medical and the administrative realms (no update propagation from one realm to the other is possible). This has evident drawbacks in financial terms (double cost for updates) and in terms of consistency of the database, that can only be guaranteed if appropriate procedures are explicitly defined by users and stored in the DBMS to be automatically triggered whenever needed.

From a traditional, centralized database perspective, the coexistence of irreducible viewpoints in a database may be considered as a design error. From a user perspective, it is not. In current DBMSs it is up to application designers and users to cope with the situation, whenever it arises, relying on primitive system functionality, such as foreign keys in relational DBMSs or generalization links in object-oriented DBMSs, to interrelate different representations of the same phenomenon. It is again up to users and application designers to define and enforce the appropriate consistency rules that may constrain the set of representations.

The centralized representation paradigm is even more uncomfortable when a database results from the integration of different pre-existing data sets, as it is the case in federated or cooperative information systems and in data warehouses. Such systems are more and more frequently required to support interoperation among different organizations, as well as for a single organization that needs to coalesce data from different sources, including the Web, to support its enterprise strategy. When data from various sources come together into a single data store, the situation where different representations of the same phenomena coexist is likely to happen and cannot be considered as a design error.

In summary, modern data management requires a new representation paradigm, such that multiple representations of the same phenomenon may coexist in a database, and this should be explicitly described and made known to the system so that it may manage the situation accordingly. In other words, supporting multiple perceptions and representations means that the users and the system are aware that two (or more) stored representations are describing the same real-world phenomenon, and are aware of which representations together form a perception. To achieve this, existing data models need to be extended with new concepts such as a means to identify perceptions and their representations, and a multi-representation link, with a well-defined semantics (which says “this representation describes the same real-world phenomenon that this other representation”), complemented with associated constraints and operators. Expected benefits include

better real-world modeling, enhanced understanding of schema diagrams and database content, improved consistency management, automatic update propagation, and data cleaning facilities (when two representations are used to check one against the other and determine if there has been some erroneous data acquisition).

In short

- *Applications require multiple representations of the same real-world phenomena.*
 - *The set of representations needed by an application defines a perception.*
 - *The view mechanism provided by DBMSs does not provide an adequate solution for supporting multiple representations.*
-

2.4.2 Multiple Representation and Spatial Databases

Geographical applications show additional requirements in terms of multiple representations, as they need flexibility also in the perception and representation of spatial features.

In the continuous view, diversity of application requirements may lead to different choices in terms of the space granularity needed by the application to capture spatial distribution, the space zones to be considered, which properties are measured and stored (and which technique is used for that purpose), and the level of detail in the value domain associated to these properties. Diversity of perception and representation of spatial features in the discrete view may also exist. For instance, the same road entity may be given a linear extent to comply with the requirements of a traffic management application, and a surface extent to comply with the requirements of a cadastral application working at a much more detailed level. The concept of level of detail in spatial data acquisition is referred to as *spatial resolution*, defined as the smallest granule of space that can be individually denoted and therefore separated from the neighboring granules. For example, if the chosen granule is one square meter in a 2D database, objects whose size is less than one square meter will be either emphasized and given the minimal extent of one square meter, or will be given no extent, or will simply not be recorded in the database. In the latter case resolution acts as a filter to determine the universe of discourse. Spatial resolution is a fundamental characteristic of a geographic database. Depending on spatial resolution, objects may have to be merged. If, for instance, it is not possible to represent two close buildings (e.g., a villa and its garage) as separate buildings, each one with its own extent, because they are too close, they will be either represented as a single building with an extent that covers both the villa and the garage, or only the larger one will be spatially represented (e.g., a visualization will show the villa but not the garage). The geometries of a set of building may collapse to form a single built-up area when the details on individual buildings are no more of interest. Finally, spatial resolution also has a smoothing effect: given a detailed geometry, a less-detailed representation will retain a simplified geometry that leaves out all irregularities whose size is less than a

given threshold. Thus, for instance, aerial shapes are more or less approximate (with small details smoothed away).

Choosing an appropriate level of resolution is essential in map production. Maps are the most natural way to provide location information, and also serve as an excellent means of visualizing analytical data about phenomena that have a geo-graphical correlation. Visualizations include geography-compliant maps, that show items of interest as faithfully as possible with respect to their real-world location and shape, as well as schematic maps (e.g., city transport systems, airline connections diagrams, train networks, facility management networks). Schematic maps focus on correct connections and readability rather than on precisely locating lines and nodes.

A map is drawn according to a given scale, i.e., the actual drawing depends on the chosen scale. For instance, a rectangular building can be represented by a rectangle on a 1:10'000 map, by a point on a 1:25'000 map and have no geographic representation at coarser scales. Thus, a physical zoom-in or zoom-out operation that would only enlarge or shrink geographic representations is simply inadequate. Drawing standards change from one scale to another one, items may (dis)appear or be (dis)aggregated because their size make them (in)visible depending on the scale, their shape may be modified (made simpler or more precise), or simply the information is not available at the requested scale. To maintain consistency and avoid redundancy, the ideal setting would be of course to maintain a database where geometry information is kept at the most precise scale, and geometries at less precise scales are automatically computed from those at more precise scales, a process called *cartographic generalization* [Müller 95] [Weibel 99].

Unfortunately, there is no complete set of algorithms that automatically derives a map at some scale from a map at a more precise scale. Some algorithms exist and more are being investigated. Since in addition cartographic generalization may be a long and costly process, the alternative is to perform cartographic generalization off-line and to store its result for direct reusability. Given this situation, map production systems tend to keep a separate database per scale, leading to problems such as lack of consistency and uncertain update propagation. Another alternative is to associate to a spatial object in a database a variety of geometric representations that are scale dependent. Databases with such a facility are called *multi-scale databases*. Supporting multiple spatial resolutions within a single multi-scale database is an on-going effort in GIS research.

Geographical databases are also subject to classical semantic resolution differences, such as an application may see the road as a single object, while another one may see it as a sequence of road sections, each one represented as an object. Semantic resolution, as spatial resolution, allows filtering out objects/relationships/attributes that are not relevant while working at a specific level of detail. In traditional databases, semantic resolution may characterize the level of detail that is appropriate within is-a hierarchies (how many levels in the specialization tree for a given root are relevant?) and aggregation hierarchies (how far is it relevant to go in the decomposition of a given object?). Geographical databases add further possibilities, such as supporting *hierarchical value domains* for attributes, where values are chosen depending on level of detail. These domains contain a hierarchy

of values, each level in the hierarchy corresponding to a given level of detail, such that values at a node are more precise than the value in the parent node. A typical example is the value domain for a land use attribute, where at the coarser level the possible values may be (built-up area, rural area, natural land), at a finer semantic level the values for built-up areas may be more precisely defined as (industrial area, residential area, dense habitat area, commercial zone, business district), such classification refinement going on for as many levels as needed by the different applications.

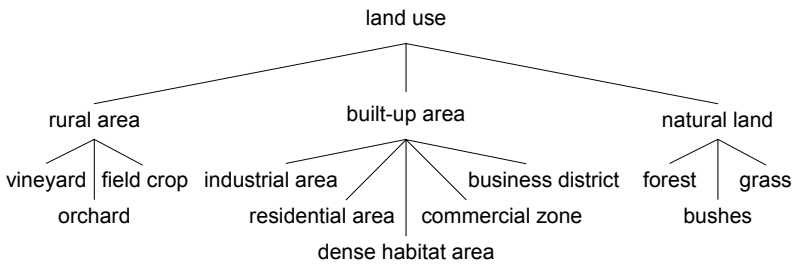


Fig. 2.45. An example of a hierarchical domain.

The idea is that each application may see the values that correspond to the semantic resolution level the application is interested in. Geographical databases also frequently illustrate the fact that semantic levels of detail may not be comparable to each other (i.e., it is not always possible to define a complete ordering of all levels in semantic resolution). For instance, a road segmentation based on crossroads (for traffic analysis) may coexist with a different segmentation based on the number of lanes (addressing transportation problems), making the two levels of detail not directly comparable.

In the following sections we use the term resolution to cover both spatial and semantic resolution. Indeed, these concepts are strongly tight together as both correspond to the idea that one looks at data using a certain focus. Usually, a precise thematic description induces a precise description of spatial features. For instance, the choice of a detailed value domain to describe the land use implies the choice of an adapted spatial resolution: with a detailed value domain the size of homogeneous land use zones tends to become smaller and needs a more detailed spatial resolution. Similarly, an application working on national-level data will deal with objects such as regions, counties, cities, and so on, while an application addressing local-level needs will describe buildings, roads, etc., using a much more precise thematic and spatial resolution. The choice of a resolution depends also on the perception that is to be represented. For instance, a database used for embedded navigation within a city needs a precise resolution to be exhaustive in the description of streets and their geometry.

The resolution of information in the database is the resolution that either was used at data acquisition, or the one that results from a cartographic generalization process. Objects with a regular extent are often acquired with a less precise resolution to reduce data acquisition costs and data storage. Resolution may also be

adapted to prevent from eliminating important objects. Therefore, a database is likely to hold data at different resolution levels, including in particular the same data with different representations for different resolution levels.

In short

- *Spatial applications put forward further multi-representation requirements.*
- *This includes in particular supporting multiple geometries of the same real-world phenomena at different resolutions.*

A multi-resolution spatial item (object, link, or attribute) is an item that is associated with multiple geometries at different resolutions. The potential variety of representations of spatial objects extends over different facets, such as:

- Multiple geometries, possibly belonging to different spatial types, like surface and point, or surface and line, may characterize the same object in different contexts (e.g., different resolutions),
- Multiple abstraction levels that make a set of objects coexist with the object(s) that represents their aggregation (whether the aggregation is based on geometric, temporal, or semantic criteria),
- Multiple abstraction levels that result in hierarchical value domains for attributes, and
- Multiple representations in terms of thematic information, which corresponds to maintaining several perceptions as in traditional databases.

Several approaches have been proposed to support multiple resolutions:

- The representations of a real-world entity are embedded in a single instance which includes multiple geometries, and all object instances are stored in a single-schema database,
- Each object has multiple, interconnected representations (i.e., database instances) and one of the following solutions apply:
 - There is a single schema that describes all representations,
 - There are multiple schemas that describe the representations with either 1) one schema per resolution range and per perception, 2) one multi-resolution schema per perception, 3) one multi-perception schema per resolution range, or 4) one intrinsic schema and several schemas (one per resolution and/or per perception) that jointly describe all representations.

A further dimension that adds multiple representations is time. A wide range of applications needs to manage time-varying information for analysis, planning, and forecast, in particular for decision support systems. This includes geographical applications, where the need for temporal support is critical in the great majority of cases. Typical examples include cadastral, risk management, and environmental applications. A map is also characterized by a given time period of validity. Of particular importance is the manipulation of moving objects [Güting 05], such as cars, vessels, and pollution disasters, where the geographic characteristics of an

object are time varying. Various multi-representation requirements derive from taking time into account. The first and most obvious one is that any phenomenon may get different representations at different points in time. The database today may differ from the database two months ago or two months ahead, in terms of values as well as in terms of relevant information structures. Supporting changes in data values has been extensively addressed by research in temporal databases, as discussed in the previous section. Supporting changes in data structures (i.e., at the meta data level) is known as the schema evolution problem. Versioning techniques are most frequently proposed as a solution to the problem. Additional facets may arise if multiple time reference systems (i.e., calendars) or if multiple time granularities are used. The former is equivalent to using multiple reference systems for space coordinates. It means that the calendar in use must be recorded as part of the metadata and that conversion functions must be available (see [Odberg 94] for an insight into such functions). The latter is somehow similar to the spatial resolution level, but with different behavioral characteristics. Moving from finer to coarser granularity (e.g., from day to month) is just a computational problem (while in space it raises cartographic generalization issues). Moving from coarser to finer granularity results in imprecise temporal specifications (also called *temporal vagueness* or *indeterminacy*). This has attracted attention from several research groups (see, for instance, [Dyreson 98] [Bettini 00] [Combi 01]).

Commercial systems poorly support the need for multiple representations. Few GISs can explicitly represent objects with multiple geometries. Current DBMSs provide limited support for multiple thematic representations. However, the situation may soon evolve as the database and GIS research communities have been active in developing proposals for new representational schemes. A summary of the state of the art is presented in Chap. 7.

2.4.3 Identifying Perceptions

Perception is guided by a specific interest in data management and determines a corresponding data representation fitting that interest. Perception acts as a complex abstraction process that includes a sequence of filtering levels:

- A first level filters objects and links leaving out whatever in the real world is not of interest. This delimits the universe of discourse.
- A second level filters the properties of interest that will describe the objects and links selected in the first step.
- A third level filters, among all possible representations of the selected properties, those that best fit with the objectives of the perception.

The sub-schema (or external schema, in ANSI/SPARC terminology) concept has been used to denote the set of data descriptions for a given perception. A sub-schema, as the term suggests, identifies a subset of the database schema, where descriptions may somehow differ from the corresponding descriptions in the database schema, such that a mapping exists between the sub-schema elements and the

database schema elements. The existence of the mapping guarantees that the sub-schema and the schema are compatible.

We shall abstract from this sub-schema/schema architecture, since it refers to a possible implementation. We shall simply assume that there is a set of identifiable perceptions supported by the database, and analyze different approaches to manage the perception information such that users can be provided with the information corresponding to their own perception.

Differences in perception does automatically entail that different representations are needed. It may be that the same representation is sharable by two or more perceptions. On the other hand, as we already stated, dealing with a single perception does not imply that no multiple representation facility is needed. For instance, a financial application may need the price of an item both in euros and in Swiss francs, which is nothing but two representations of the same information. A GIS application may need a simultaneous display of the same data at different scales. A temporal application may need to compare the values of similar data sets at different times.

As already stated, perception is driven by many parameters. Previous examples illustrated different measurement units (euros versus Swiss francs), spatial and semantic resolution, and time. Which parameters are worth being taken into account varies from one database to another. The choice is a matter of how the human/enterprise organization is mapped into the database organization, so it should be the responsibility of the database designer. At a generic level (i.e., independently from what the actual perception parameters may be), a perception for which n parameters have been chosen as relevant is denoted by a vector: $s = \langle p_1, p_2, \dots, p_n \rangle$, where each p_i is the value for this perception of its i^{th} parameter. For instance, assuming a three-parameter framework based on viewpoint (here understood as user category), spatial resolution (with a meter granularity), and time (with a year granularity), a given perception may be identified as:

$s_1 = \langle \text{Viewpoint: "Management", Spatial resolution: 10, Time: 2002} \rangle$.

Value domains for the different parameters are not necessarily homogeneous. Some domains are discrete and unordered sets, as it is likely to be the case for the viewpoint dimension (e.g., if names of user categories are used). Others may be an interval of \mathbb{R} , for instance for spatial resolution. We assume that the database designer chooses the parameters that are appropriate as well as the value domain for each parameter (how this is done is beyond the scope of this book). This defines the value domain for the perception vectors in the database.

2.4.4 Stamping

To simplify notations and discourse without loss of generality, we abstract from the multidimensionality of perception and simply refer to perception vectors as

perception stamps, or just *stamps*³⁴, denoted as s_1, s_2, \dots, s_n . In summary, we assume that the database designer defines the set of stamps that will characterize the perceptions existing for the database. This set is designated by the predefined name `DBPerceptions`. We show hereinafter how these stamps are used, in association with both metadata (data descriptions in the schema) and data (data values in the database). While we discuss consistency rules for correct stamping, we do not discuss inter-perception relationships and consistency rules that may exist in a given framework due to the specific semantics of stamps. For instance, two geometric representations of the same real-world entity whose stamps differ only in the resolution parameter (e.g., two lines representing the same road segment for different scales) may be constrained by cartographic rules enforcing that the less detailed line is the result of a given cartographic generalization algorithm applied to the more detailed line. In terms of relationships between perceptions, an example would be the case where data perceived by a given user category is by definition included into the data perceived by another user category (e.g., the perception for a manager includes all what is relevant to employees plus some extra managerial data). A full analysis of such semantic relationships and related consistency rules remains to be done.

From data definitions (metadata) to data values, anything in a database relates to one or several perceptions. The first step for the database administrator is to identify the perceptions that are to be supported by the database and to associate a unique stamp to each one of them. This defines the set of stamps that are allowed for use with the database. We say that the database schema is stamped with this set. For instance, in the Risks application that served as case study for the MurMur project, stamps were limited to include the two facets we already mentioned: viewpoint and resolution. The former was seen as the primary stamp, expressing whether a representation was part of the user's world or not. The resolution stamp was intended to filter data to only retain what was significant for certain usages by a given user category. For example, assuming one perception for risk managers (in charge of decision-making processes), and two perceptions for risk technicians (in charge of observations, measurements and risk map preparation, working at either a finer or a coarser scale) were defined, the schema would include the following stamps:

```
s1 = < Viewpoint: "Management", SpatialResolution: 50>
s2 = < Viewpoint: "Technician", SpatialResolution: 10>
s3 = < Viewpoint: "Technician", SpatialResolution: 100>
```

Stamping an element of a schema defines for which perceptions the element is relevant. Thus, an element that has a single representation may bear multiple stamps, meaning that the same representation is shared by several perceptions. Consistency mandates that stamps associated to an object (or relationship) type form a subset of the stamps associated to the schema. Similar rules apply to prop-

³⁴ The term *stamping* is due to the similarity with timestamping techniques in temporal databases, where timestamping denotes the technique that associates a period to data for identifying the timeframe for which the data is relevant.

erties within a type. An object or relationship type relevant to several perceptions may show different properties depending on the perception. Consequently, a property may be stamped with a subset of the stamps associated to the type it belongs to. The same applies at the value level. A multi-perception attribute may have different values that are specific to given perceptions. For instance, in multilingual databases, a property such as `riverName` may take different values according to the language in use, the language determining the perception.


Complementarily to stamping database elements, transactions accessing the database should be given a means to specify which perceptions (one or many) they adhere to. That will determine the representations (data types and values) relevant to them. We assume that transactions issue an `openDatabase` command to specify which perceptions (stamps) they want to use. Matching this set with the sets of stamps associated with the object and relationship types defines which object and relationship types are actually visible to the transaction, and with which properties and which populations. Thus, stamping provides functionality similar to a sub-schema definition capability, with the advantage that this approach maintains an integrated view of all perceptions, while subschema definition (as provided in Coadasy-like database systems) isolates each schema definition.

In short

- *A stamp identifies a particular perception with which real-world phenomena captured in a database may be viewed.*
- *Elements of the database (types, properties, instances) are stamped for defining for which perceptions they are relevant.*

2.4.5 Multiple Representation Modeling

Stamping provides an easy way to identify which representations stem from a given perception. But how can we design a schema to make the DBMS aware of multiple coexisting representations of the same phenomena? Let us assume, as a running example, that there are two perceptions of the same real-world entities, e.g., road segments, which need slightly different representations. There are two complementary techniques to organize multiple representations.

One solution is to build a single object type that contains both representations, the knowledge of “which is which” being provided by the stamps of the properties of the type. Following this approach, in Fig. 2.46 the designer defines a single object type, `RoadSegment`, and associates to it the stamps identifying the two perceptions, say `s1` and `s2`. As shown in the figure (notations and semantics are detailed in Sect. 2.4.6), the perception stamps of an object type are shown in a box under the name box using the  icon. We say `RoadSegment` is a *perception-varying object type*, as the actual representation of road segments changes from one perception to another. `RoadSegment` is both a multi-representation (it holds two representations) and a multi-perception (it relates to two perceptions) object type. At the instance level, the fact that two representations relate to the same real-

world entity is in this case conveyed by the fact that the two representations are part of the same object instance. Hence, this solution only applies to cases where representations in the two perceptions are interrelated at the instance level by a (total or partial) 1:1 mapping.

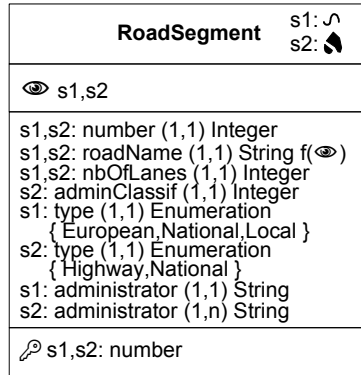
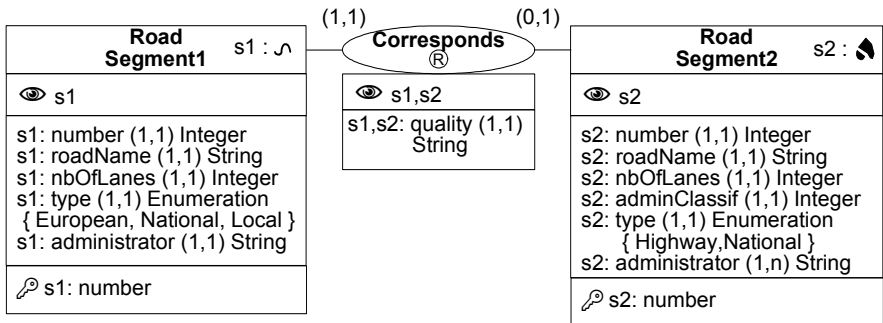


Fig. 2.46. An illustration of a bi-representation type, bearing stamps s1 and s2.



Integrity Constraint : $\forall c \in \text{Corresponds (}$
 $c.\text{RoadSegment1.number} = c.\text{RoadSegment2.number} \wedge$
 $c.\text{RoadSegment1.nbOfLanes} = c.\text{RoadSegment2.nbOfLanes})$

Fig. 2.47. The RoadSegment type (from Fig. 2.46) split into two mono-representation object types and an inter-representation relationship type.

Another solution to organize alternative representations is to define two separate object types, each one holding the representation for the corresponding perception (each object type bears the corresponding stamp). The knowledge that the two representations describe the same entities is then conveyed by linking the object types with a relationship type that holds a specific inter-representation semantics (cf. Fig. 2.47). In this example the same real-world road segment is materialized in the database as two object instances. Instances of the relationship type Corresponds tell which object instances represent the same road segment. To make the system aware of the semantics of the relationship, a specific semantic

annotation is added to the definition of the relationship type. We call such a relationship type an *inter-representation relationship type*. Its inter-representation semantics is visually indicated on schema diagrams by the ® icon. Cardinalities of Corresponds show that buildings that have a representation at the most detailed level, s2, do not necessarily have one at the less detailed level, s1.

When more than two representations are needed, it is up to the designer to decide which ones are to be instantiated separately from others, and which ones should be integrated to form a single instance. Any mix of multi-perception types and inter-representation links can convey the solution that best fits application requirements. If more than two types are used to describe and store the desired set of representations for a given set of entities, these types have to be linked by as many inter-representation links as appropriate. Inter-representation links at the type level are not transitive. It is possible to have inter-representation links between types T_1 and T_2 and between types T_2 and T_3 without an inter-representation link between T_1 and T_3 . Consider, for instance, a database with object types Person, Company, and CarOwner, and rules for this database stating that persons and companies are separate sets of objects, while both persons and companies may own cars. In this case there will be an inter-representation link between Person and CarOwner, another between Company and CarOwner, but no link between Person and Company. Alternatively, one could think of using n-ary inter-representation links to bind in a single instance link all object instances that represent the same real-world entity. But usually this would be inappropriate. N-ary links can only be instantiated if all linked instances exist at the same time (data modeling practices do not support links with pending roles), which cannot be assumed to be the general case for multi-representation.

In short

- *Multiple perceptions of a real-world phenomenon can be embedded into a single object type.*
- *An alternative solution is to use inter-representation relationship types for relating object types holding different representations of the same phenomenon.*

2.4.6. Perception-Varying Object Types

Stamping object types and relationship types, possibly with multiple stamps, defines for which perceptions the type is relevant. We call this the *visibility of the type*. Only transactions whose associated set of stamps intersects the set of stamps characterizing a type (its visibility) will see the type. Consistency mandates that stamps associated to a type form a subset of the stamps associated to the schema. Similar rules apply to properties within a type. A property cannot be visible when its type is not. Hence, a property is stamped with a subset (possibly the whole set) of the stamps associated to the type it belongs to (cf. Fig. 2.46).

Non-stamped types could be allowed as a shortcut for bearing all stamps defined in the schema they belong to. Such types convey a real-world representation

that is independent of any perception, i.e., the representation holds whatever perception is considered. Such types, their properties, and instances are always visible whatever the transaction stamp is, unless their visibility is restricted as described hereinafter. Similar rules apply at the property level. In the absence of restrictions, stamps defined at the type level extend over properties, providing properties and their values with the same visibility as the type they belong to.

Let us resume the *RoadSegment* example (cf. Fig. 2.46), where the designer has decided to merge the representations needed by two perceptions, identified by stamps *s1* and *s2*, into a multi-representation object type *RoadSegment*. Below the object type name are shown the stamps associated to the type: *s1* and *s2*. Road segments are spatial objects (i.e., objects whose spatial extent is relevant for the applications). In schema diagrams the spatiality associated to a type is shown right-hand to the type name. As shown by the icons, the spatial extent is represented either as a surface (more precise description, stamp *s2*) or as a line (less precise description, stamp *s1*) depending on resolution.

Representation *s1* needs attributes road segment number, road name, number of lanes, type, and administrator (denoting the maintenance firm in charge). Representation *s2* needs attributes road segment number, road name, number of lanes, administrative classification, type, and administrator. While the road segment number and the number of lanes are the same for *s1* and *s2*, the name of the road is different, although a string in both cases. For instance, the same road may have name “RN85” in representation *s1* and name “Route Napoléon” in *s2*. The road segment type takes its values from predefined lists, the lists being different for *s1* and *s2*. Finally, *s2* may record several administrators for a road segment, while *s1* records only one. More precisely, the list of attributes shows that:

- *number* is a monovalued and mandatory attribute (minimum and maximum cardinalities equal to 1) shared by *s1* and *s2*.
- *roadName* is a shared monovalued mandatory attribute whose value is a function of stamps. We call this a *perception-varying attribute*, identified as such by the $f(\text{👁})$ notation.
- *nbOfLanes* is a monovalued and mandatory attribute shared by *s1* and *s2*.
- *adminClassif* is a monovalued mandatory attribute that only belongs to representation *s2*.
- *type* is a monovalued mandatory attribute in both *s1* and *s2*, with specific enumerated domains for each representation.
- *administrator* is a mandatory attribute in both *s1* and *s2*, but it is monovalued for *s1* and multivalued for *s2*.

Moreover perceptions *s1* and *s2* share a common key, the attribute *number*, i.e., no two instances, belonging to the same perception or different perceptions, may have the same *number* value.

As shown by the example, the geometry attribute in spatial objects often has several representations that correspond to different spatial resolutions. The lifecycle attribute in temporal objects can also be perception varying. For example, the designer could add to the *RoadSegment* object type a perception-varying lifecycle

cle. That would allow him/her to describe the lifespan of an instance to begin, for perception *s1*, at the beginning of the construction of the road segment, and, for perception *s2*, at the end of the construction. Attribute variation depending on perception extends to all facets of the attribute: visibility, cardinalities, value domain, value, as well as time and space variability.

Instances are another component of a type; hence they obey the same stamping consistency rule as properties. In our example, a *RoadSegment* instance can be created by a transaction using stamp *s1*, or using stamp *s2*, or using both stamps. If the transaction uses both *s1* and *s2*, it may create the whole value of the instance, as in a normal insert operation in traditional databases. If the transaction holds or uses only one stamp, it can create values only for the attributes that exist for that stamp. In short, it can create only a part of the instance. Partial creation of an instance means that two transactions using different stamps must be able to share an identification mechanism (e.g., the road segment number) guaranteeing that their data can correctly be merged by the DBMS into a single instance.

Consider the creation of a *RoadSegment* instance. It can be done by two transactions. The first one creates a new instance. This transaction has to provide a value for all the mandatory attributes corresponding to the format at the given stamp. For instance, the following insert operation inserts a new instance of *RoadSegment* stamped with *s1*³⁵:

```
id = insertObject (RoadSegment, {s1}, (geometry: Line <(x1,y1), ..., (xn,yn)>,
number: 1, roadName: "E66", nbOfLanes: 4, type: "European",
administrator: "Bouygues" ))
```

The second transaction adds a new representation to the instance that was previously defined, and thus, the transaction has to identify the instance it wants to extend. For example, the following operation adds a new representation to the previously created road segment instance³⁶:

```
addORepresentation (RoadSegment, id, {s2}, (
geometry: SimpleSurface <(x'1,y'1), ..., (x'n,y'n)>, roadName: "A41",
adminClassif: 1, type: "Highway",
administrator: {"Bouygues", "SaraCie"} ))
```

Stamps associated to a type may be restricted at the instance level, to limit visibility of an instance to a subset of the type visibility. This allows defining different subsets of instances that are visible for different stamps among those supported at the type level. Thus, multi-perception types have a system attribute (not accessible to users) called *perceptions* allowing to keep track of the visibility of its instances. For example, as the object type *RoadSegment* has two stamps *s1* and *s2*, it is possible to define instances that are only visible to *s1*, instances that are only visible to *s2*, and instances that are visible to both *s1* and *s2*. An instance is visible for at least one of the stamps of its type, the stamp that has been used while inserting the instance in the database. A transaction with stamp *s1* only will see the instances

³⁵ The *insertObject* operation is defined in Chap. 5.

³⁶ The *addORepresentation* operation is defined in Chap. 5.

stamped s1 or {s1, s2}. Similarly for transactions stamped s2 only. A transaction with both stamps s1, s2 will see all RoadSegment instances, but it has to be aware that the actual format of each instance varies according to its stamps. Indeed, a stamp restriction at the instance level also applies to attributes and values within the instance: An instance that bears only a subset, say S, of the stamps of the object type will have only the properties that have a representation for some (or all) of the perceptions defined by S. An example was given above for the road segment number 1 within the time interval defined by the insertion of its s1 representation and the addition of its s2 representation.

In short

- *Stamping allows defining for which perceptions a type is relevant.*
- *Properties and instances may also be stamped.*
- *The stamps of a type (respectively, of a property or an instance) are included in the set of stamps of its schema (respectively, of its type).*
- *The system attribute **perceptions** allows keeping the perceptions at which an instance is visible.*

Visibility restrictions can be independently specified for several attributes within the same type. In particular, stamp restriction can be performed at all levels of a complex attribute. The restriction of a complex attribute stamp applies to all of its components. The attributes composing a complex stamped attribute implicitly have the same stamp as the complex attribute they belong to. If needed, they can restrict this implicit stamp by keeping only a subset. However, visibility restrictions must obey the constraint that the set of attributes for each stamp must form a consistent data structure for the type. Consistency here means that the attribute tree built by pruning according to the stamp filter must be a subtree of the original attribute tree (with the type as root) such that: for every attribute in the subtree its parent attribute in the original tree must also be in the subtree, and for every complex attribute in the original tree that appears in the subtree at least one of its original component attributes must be in the subtree.




County
 s1,s2,s3
s1,s2,s3: name (1,1) String s1,s2,s3: nbInhabitants (1,1) Integer s1,s2,s3: catchmentArea (1,n) s1,s2,s3: number (1,1) Integer s1,s3: river (1,1)  s2: extent (1,1) 

Fig. 2.48. Stamping components of a complex attribute.

In Fig. 2.48, catchmentArea is a multivalued complex attribute composed of number (monovalued), river (monovalued, linear), and extent (monovalued, sur-

face). Attribute `catchmentArea` bears the stamps of its object type, `s1`, `s2`, and `s3`. Attributes `river` and `extent` restrict these stamps: `river` to `s1` and `s3`, and `extent` to `s2`. The object type is therefore visible at `s1` and `s3` with attributes `geometry`, `name`, `nblnhabitants`, `catchmentArea.number`, and `catchmentArea.river`, and at `s2` with attributes `geometry`, `name`, `nblnhabitants`, `catchmentArea.number`, and `CatchmentArea.extent`.




County
 <code>s1,s2,s3</code>
<code>s1,s2,s3: name (1,1) String</code> <code>s1,s2,s3: nblnhabitants (1,1) Integer</code> <code>s1,s2: catchmentArea (1,n)</code> <code>s1,s2: number (1,1) Integer</code> <code>s1,s2: river (1,1)</code>  <code>s1,s2: extent (1,1)</code> 

Fig. 2.49. Stamping a complex attribute.

On the other hand, in Fig. 2.49, the attribute `catchmentArea` restricts the stamps of its object type. In this case, the object type is visible at `s1` and `s2` with attributes `geometry`, `name`, `nblnhabitants`, and `catchmentArea` with its components `number`, `river`, and `extent`, and at `s3` with attributes `geometry`, `name`, and `nblnhabitants`.




County
 <code>s1,s2,s3</code>
<code>s1,s2,s3: name (1,1) String</code> <code>s1,s2,s3: nblnhabitants (1,1) Integer</code> <code>s1,s2: catchmentArea (1,n)</code> <code>s1,s2: number (1,1) Integer</code> <code>s1: river (1,1)</code>  <code>s1,s2: extent (1,1)</code> 

Fig. 2.50. Stamping a complex attribute and its components.

Finally, in Fig. 2.50, the attribute `catchmentArea` restricts the stamps inherited from its object type, and the attribute `river` further restricts the stamps. In this case, the object type is visible at `s1` with attributes `geometry`, `name`, `nblnhabitants`, `catchmentArea.number`, `catchmentArea.river`, and `catchmentArea.extent`, at `s2` with attributes `geometry`, `name`, `nblnhabitants`, `catchmentArea.number`, and `catchmentArea.extent`, and at `s3` with attributes `geometry`, `name`, and `nblnhabitants`.

In short

- Stamp restriction can be applied to all levels of a complex attribute.

In a multi-perception framework, generalization/specialization hierarchies may include object (relationship) types that belong to various perceptions and possibly hold multiple representations. As we know from Sect. 2.2.5, is-a links are characterized by population inclusion semantics and inheritance of properties and links. These characteristics put strong consistency constraints that should be preserved while allowing for multi-perception and multi-representation types. As a consequence, is-a links have also to be characterized with respect to perception, i.e., they should be stamped so that the system knows exactly which instances are to be constrained by the link. An is-a link must belong to the same perception as the object (relationship) types it links. Otherwise stated, the supertype and the subtype must share one or several stamps, and the is-a link is stamped with a non-empty subset of these common stamps.

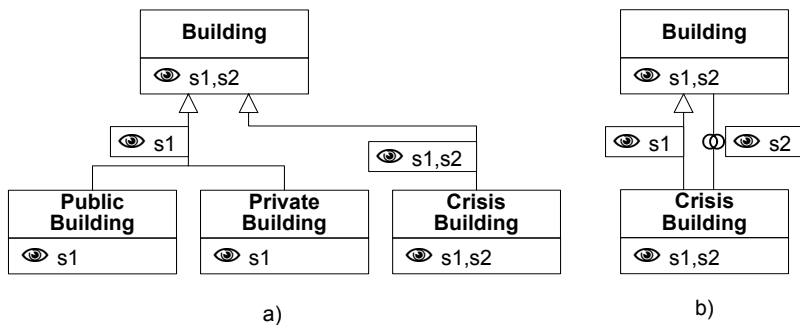


Fig. 2.51. Example of perception-dependent generalization hierarchies.

Fig. 2.51 a) presents an example where the above rules are obeyed. Perception *s1* sees all four object types, with *PublicBuilding*, *PrivateBuilding*, and *CrisisBuilding* as subtypes of the *Building* supertype. Perception *s2* sees only the *Building* and *CrisisBuilding* object types, with *CrisisBuilding* subtype of *Building*. Transactions with only perception *s2* see only this portion of the hierarchy.

Overlapping links, like is-a links, are perception dependent. Fig. 2.51 b) shows an example where, for perception *s1*, the set of *CrisisBuilding* instances stamped *s1* is a subset of the *Building* instances stamped *s1*. But this inclusion rule does not hold for perception *s2*: There may exist *CrisisBuilding* instances stamped *s2* that have no corresponding (i.e., with the same oid) instance stamped *s2* in *Building*. For perception *s2* *CrisisBuilding* and *Building* are in multi-instantiation only.

In short

- *Is-a and overlapping links are also perception dependent, and therefore stamped.*
- *The stamps of is-a and overlapping links are included in the set of stamps of each of the object or relationship types they link.*

2.4.7 Perception-Varying Relationship Types

Relationship types are as dependent on perception as object types are. Therefore they may have different representations, and representations may be stamped with the same semantics and according to the same rules as for object types: Attributes and instances may be stamped with the whole set of stamps of the relationship type, or with a subset only. Attributes may have different representations (i.e., definitions) associated to the perception stamps. Moreover, the structure (roles and association/multi-association kind) and the semantics (e.g., aggregation, topology, synchronization) may also have different definitions depending on the perception. An example of different sets of attributes and of different definitions of attributes, depending on perception, for a relationship type, could be exactly as the example of RoadSegment of Fig. 2.46.

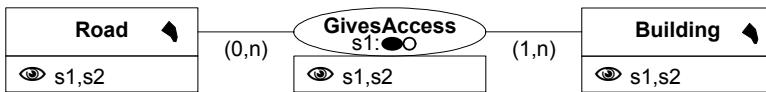


Fig. 2.52. A relationship type with perception-varying semantics.

Fig 2.52 shows an example of different semantics, where the designer defined the relationship **GivesAccess** as 1) a topological adjacent relationship type for perception *s1*, and 2) a plain relationship for perception *s2* (i.e., an association, without any peculiar semantics or constraint). This would allow users to link, through **GivesAccess** instances stamped *s1*, couples of **Road** and **Building** instances whose geometries are indeed adjacent. **GivesAccess** instances stamped *s2* and not *s1* will allow users to link a road to a building that is not adjacent to the road, but that is somehow connected to the road, for instance by a private driveway. The kind of constraint that defines a relationship type as topological or synchronization may also be perception varying. In other words, the same relationship type may, for instance, hold a topological constraint “inside” for perception *s1*, and a topological constraint “adjacent” for perception *s2*.

The kind association/multi-association of a relationship type may also depend on the perception. Let us again refer to a relationship type **GivesAccess** linking the two object types **Road** and **Building**, but with different characteristics this time. Assume that the designer has specified that **GivesAccess** is a topological adjacent association for perception *s1*, and a topological adjacent multi-association for perception *s2* with cardinalities permitting to link, by one **GivesAccess** instance, one road to the set of buildings that are adjacent to this road. This would create two disjoint sets of **GivesAccess** instances: the association instances linking one road and one building, and the multi-association instances linking one road and a set of buildings.

The roles of a relationship type may also be perception dependent. For example, in Fig. 2.53, the cardinalities of the role linking **LandPlot** to the relationship **Intersects** are $(0,1)$ in perception *s1* and $(0,n)$ in perception *s2*. Further, the same relationship type may be a binary relationship type in one perception and a ternary

relationship type in another perception. For example, Fig. 2.54 (a variant of Fig. 2.21) shows that an observation is perceived in s_1 as a relationship between an observer and an avalanche event, while perception s_2 sees the same observation as also involving a validator having validated the observation.



Fig. 2.53. A role with perception-varying cardinalities.

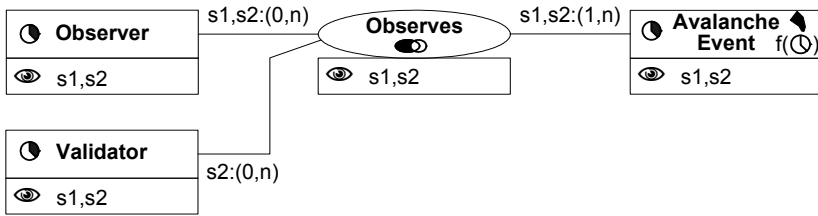


Fig. 2.54. A relationship type with an additional role specific to perception s_2 .

However, the objects (or sets of objects) linked by a relationship instance cannot change from one perception to another one within the same relationship instance. This is because the linked objects are inherently part of the relationship instance, i.e., they participate in the identification of the relationship instance. If any of them is replaced with another object, it is not anymore the same relationship instance. So, for a given relationship instance and role, the object instance(s) that is (are) linked is (are) always the same, whatever the perception is. On the other hand, as already stated, a perception may see only a subset of the roles. But, as a perception must always abide by the basic rules of the data model, a perception of a relationship must contain at least two roles.

As already mentioned, the basic rule for stamping, stating that “a component element may be stamped with all or a subset of the stamps of the element to which it belongs”, is valid for attributes and instances of a relationship type, exactly as for an object type. The rule is also valid for the structure and the semantics of a relationship type. In other words, the attributes, instances, roles, association/multi-association kind, semantics of a relationship type may bear a subset (or the set) of the stamps attached to the relationship type.

Contrarily to is-a links, a relationship type may link object types that do not belong to the same perception as the relationship itself. These relationship types are bridges that relate different perceptions. For example, in Fig. 2.47, the Corresponds relationship type links the s_1 object type RoadSegment1 and the s_2 object type RoadSegment2. All Corresponds instances, whatever their stamps, link one s_1 object and one s_2 object. As in this specific case Corresponds has inter-representation semantics, it allows transactions holding both stamps s_1 and s_2 to

navigate from one representation of road segments the other one. In other words, *Corresponds* is a bridge between perceptions and between representations³⁷.

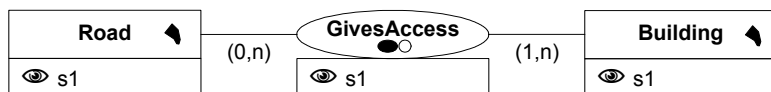


Fig. 2.55. A topological relationship type that is local to perception s1.

Let us now consider the relationship type in Fig. 2.55. In this example, the relationship type and the linked object types all bear the same, single stamp s1. All *GivesAccess* instances necessarily link objects belonging to the same perception as the relationship, they are local to the s1 perception. We say that a relationship type is *local* if each of its instances, for each of its perceptions, is local. *GivesAccess* is the simplest example of a local relationship type. Non-local relationship types are bridge relationship types, i.e., they have at least one instance that, for at least one perception, links at least one object that does not belong to that perception.

Let us now refer back to the *GivesAccess* relationship type as defined in Fig. 2.52, where the two object types and the relationship type in between hold the same two stamps, s1 and s2. Depending on the configuration of its instances, this *GivesAccess* relationship type may be local or bridge. It is local if every instance stamped s1 links a *Road* instance and a *Building* instance both stamped at least s1, and if every instance stamped s2 links a *Road* instance and a *Building* instance both stamped at least s2. Otherwise, it is a bridge relationship type. As a bridge relationship type may contain local instances, transactions may use its local instances to navigate within the corresponding perception, and its bridge instances to navigate between perceptions. It is worth noting that bridge instances of a relationship type do not belong to any mono-perception view of the database.

It is also worth noting that, contrarily to *Corresponds*, the *GivesAccess* relationship type of Fig. 2.52 may be a bridge between perceptions, but is not a bridge between representations. The characteristic, local or bridge, is independent from the other characteristics of relationship types. In particular, bridge relationship types can bear any kind of semantics. They can be inter-representation, like *Corresponds* in Fig. 2.47, as well as plain relationships with no peculiar semantics, or constraining relationships like *GivesAccess* in Fig. 2.52.

Access to bridge relationships is ruled by the basic principle that relationships are only meaningful if they come together with the objects they link. This definitional constraint must be enforced also with respect to perceptions. Hence, a rule that must obviously be enforced is that a transaction may see a representation of a relationship type only if its stamps allow seeing the relationship type and the linked object types (i.e., pending roles are not allowed). At the instance level, the same rule applies: Only visible instances of the relationship that link object in-

³⁷ Notice that if *Corresponds* were stamped s3 (instead of s1, s2) the only change would be that transactions using *Corresponds* would have to hold stamp s3 in addition to stamps s1 and s2.

stances visible to a transaction may be delivered to the transaction. Therefore, seeing a bridge relationship always requires several perception stamps. For example in Fig. 2.47, transactions must hold at least both stamps, s1 and s2, to be able to see the Corresponds relationship type.

Topological and synchronization relationship types rely on the geometry and lifecycle attributes of the linked object types. Whenever an instance of these relationship types is to be created for a given perception, the geometries or lifecycles in the linked object instances have to be retrieved to check that the topological or temporal constraint is satisfied. This means that the transaction creating or accessing the relationship instance must see the linked geometries or lifecycles, and not only the linked object types (i.e., it must have at least one stamp of the relationship type and one stamp of the constrained geometry – or lifecycle – of both linked object types).

If the linked objects have several definitions, depending on perception, for the geometry or lifecycle attributes, the specification of the topological or temporal predicate has to explicitly denote which definitions are to be used. Similarly, if the linked object types define geometry or lifecycle as perception-varying attributes, the topological or temporal predicate has to explicitly specify which is the intended semantics (whether one value or all values have to satisfy the predicate), in the same way it is done in case of time-varying geometries.

As a practical note, it is worth mentioning that in our experiments with application-oriented designers, we have seen that they tend to use the default stamping for relationship types, i.e., giving them all the stamps of the database or all the stamps of the linked object types. On the other hand they are very keen on choosing a specific subset of stamps for each object type.

In short

- *Relationship types may be perception dependent, and therefore stamped.*
 - *The rules for property and instance stamping are the same as for object types.*
 - *In addition, the structure and the semantics of relationship types may also be perception dependent.*
 - *Relationship types can either link object types belonging to the same perception as themselves, or link object types of other perceptions.*
 - *The latter are bridges allowing users to go from one perception to another one.*
-

2.4.8 Consistency of a Multi-Perception Database

A multi-perception database defined for say N perceptions contains the equivalent of N classic mono-perception databases, obeying usual consistency rules of conceptual data models, plus a number of implicit and explicit links among the mono-perception databases. Explicit links are the bridge relationship types that relate object types of different perceptions. Implicit links are all the multi-perception object and relationship types that, per definition, belong to several perceptions.

A multi-perception database, taken as a whole, does not necessarily obey the usual consistency rules for classic mono-perception databases. It can contain different (one per perception) representations for the same fact that may be conflicting at the schema or at the instance level. It may be, for instance, that for one perception s_1 the entity type A is a subtype of the entity type B , and that the *is-a* link does not hold in another perception s_2 . Or, in perception s_1 the entity types A and B are related by an aggregation relationship, and they are not linked in the perception s_2 . As a last example, the instance i_0 of the entity type A has different values in perceptions s_1 and s_2 . The rules defining the consistency of a multi-perception database, D , containing perceptions s_1, s_2, \dots, s_n are:

- *Consistency rule for each perception:* Each of the mono-perception databases composing D must abide by all the classic consistency rules for classic databases. The mono-perception database of D for perception s is obtained as follows. It contains each s perception of each object type that has a s perception, each *is-a* link that is defined for the s perception, and the s perception of each relationship that is local to the s perception (i.e., such that all its s roles link object types belonging to the s perception).
- *Inter-perception consistency rule 1:* If two perceptions share a common element (object type, relationship type, attribute, role of a relationship type) that has the same definition for both perceptions and this definition is not perception-varying, then at the instance level, these two elements must have the same instances or values. For example in Fig. 2.46, the s_1 and s_2 perceptions of the object type *RoadSegment* share the attribute *number* which is not perception-varying, therefore any *RoadSegment* instance belonging to perceptions s_1 and s_2 must have the same value for the *number* attribute for both perceptions.
- *Inter-perception consistency rule 2:* A relationship type, be it local or bridge, must obey all classic consistency rules of semantic data models for relationships, including the fact that no dangling roles are allowed. Therefore, a relationship representation is delivered only if the perception stamps hold by the requesting transaction make visible the complete instance for at least one perception or combination of perceptions. Similarly, a relationship type is visible only to transactions whose stamps make visible a representation of the relationship type that is complete for at least one perception or combination of perceptions. This restriction ensures that, for the database users, the bridge relationships behave like classic, mono-perception relationships.

All these consistency rules are precisely defined in the formalization of the MADS data model provided in Annex A.

2.4.9 Summary on Multi-Representation Modeling

Nowadays it is crucial for information management to support multiple representations of the same real-world phenomena. The fundamental activity of the abstraction process is to determine the characteristics of real-world phenomena that are essential to an application. However, determining what is essential and what it

is not depends on many aspects, including in particular operational objectives. As a consequence, several applications sharing real-world phenomena of interest usually require different representations of the same phenomena. In addition, spatial and temporal information add further requirements with respect to multiple representations. A typical example for space arises when the spatial extent of phenomena has to be kept at different levels of detail, for instance for producing maps at different scales.

We have seen that traditional solutions for this problem, like the view mechanism in database management systems, or the generalization and aggregation links in the object-oriented paradigm, are not sufficient for coping with the requirements for multi-representation. The problem with these approaches is that they presuppose a centralized representation paradigm, where there is a “canonical” or common viewpoint from which all other perspectives of the same phenomena can be derived. This solution definitely does not address current information management requirements where different data sources have to be put together for building cooperative information systems (whether federated or peer-to-peer). In such a setting no particular viewpoint can be favored, each one of them has its *raison d’être*, and the divergences in viewpoints must be reconciled.

A first necessary step to cope with this problem is to identify the different perceptions that are needed for considering a real world of interest. These perceptions are driven by many parameters, and the number and characteristics of the parameters vary from one database to the other. For example, in our Risks application the perceptions were defined by a couple of user category and spatial resolution. The approach followed by MADS is to identify perceptions by (multi-dimensional) stamps. The stamps are then used for defining which elements of the database (i.e., types, properties, instances) are relevant for the corresponding perception.

We have shown that there are two complementary ways to deal with multi-represented objects. In the first, integrated approach, a single object (or relationship) type groups the different perceptions associated to a particular phenomenon. In this approach one real-world phenomenon is represented by a single database instance. An alternative solution is to capture the different representations in distinct object types that are linked by inter-representation relationship types. Such links keep track of the different representations (or more precisely, the different database instances) corresponding to the same real-world phenomenon. We have shown that MADS allows to the designer to decide the particular mix of multi-perception types and inter-representation links that best fits application requirements.

The stamping solution advocated by MADS for coping with multiple representations (i.e., stamping object and relationship types as well as their attributes and characteristics) leads to perception-varying types, where the structure of the database, of its types, and of their corresponding instances depends on the perception in use. Further, following a similar approach as for space- and time-varying attributes, perception-varying attributes allow to represent attributes whose value for a particular instance depends on the perception. This provides in particular an elegant solution to the problem of multi-scale databases, where the geometry of features depends on spatial resolution. Stamping is also associated to instances, al-

lowing the population of a type to vary according to the perception. Finally, we have shown how stamping also applies to is-a links, thus creating generalization hierarchies that depend on the perception.

MADS data manipulation languages (presented in Chap. 5) allow users to correctly manipulate the database in one of two modes: mono-perception database or multi-perception database. In the second mode, the user sees several representations, can access any one of them, and can navigate from one representation to another one.

2.5 Integrity Constraints

Integrity constraints provide a way to more precisely define the semantics of data and play an essential role in establishing the quality of a database and its correct evolution. *Integrity constraints* are assertions that restrict the data that may appear in the database, to prevent the insertion of data that are obviously incorrect with respect to rules governing the real world and its representation in the database. Different kinds of restrictions can be specified. Restrictions on a value domain are possibly the simplest form of integrity constraints. Typical examples include the specification of a limited allowed range over the underlying domain (e.g., stating that allowed values for the *windForce* attribute are decimal numbers in the 1.0-10.0 range), and the explicit enumeration of allowed values (e.g., stating that allowed values for a *gender* attribute only include two values, *female* and *male*). These restrictions are intended to limit the possibility of entering erroneous data, or generating erroneous data by some inappropriate update of existing data. Unfortunately, they can only avoid errors that may be detected. They cannot avoid entering a value that is plausible but is not the value that was to be entered (e.g., entering 16.2 as value for the *windForce* attribute can be rejected if the domain has been restricted to the 1.0-10.0 range, but entering 6.2 instead of 2.6 cannot be prevented by a simple value domain restriction). Integrity constraints may be of arbitrary complexity, involving data retrieval operations and computations on many values stored in the database. Most often they restrict attributes values, forcing conformance to application rules. But they may also restrict the creation of object or relationship instances. For example, cardinality constraints associated to roles in relationships limit the number of relationships that can be created.

Most data models come with *embedded integrity constraints*, i.e., constraints that can be specified using predefined clauses of the associated data definition language. For example, the SQL language supported by major relational DBMSs allows the specification of uniqueness constraints (using PRIMARY KEY and UNIQUE clauses), and referential constraints (using FOREIGN KEY clauses). Embedded constraints cannot cope with all the complexity of application rules that concur in defining correctness criteria for data in use. Consequently, a data definition language has to provide a way for expressing ad-hoc integrity constraints. SQL, for example, offers a CHECK clause to define ad-hoc value domain restrictions and generic constraints among attributes of one or more relations, whenever

Conceptual Modeling for Traditional and
Spatio-Temporal Applications

The MADS Approach

Parent, C.; Spaccapietra, S.; Zimányi, E.

2006, XVIII, 466 p. 115 illus., Hardcover

ISBN: 978-3-540-30153-0