5

Modularity in Databases

Christine Parent[1], Stefano Spaccapietra[2], Esteban Zimányi[3]

[1] HEC ISI, Université de Lausanne, CH-1015 Lausanne, Switzerland. christine.parent@epfl.ch
[2] Database Laboratory, Ecole Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland. stefano.spaccapietra@epfl.ch
[3] Department of Computer and Decision Engineering (CoDE), Université Libre de Bruxelles, Belgium. ezimanyi@ulb.ac.be

Summary. Modularization can be sought for as a technique to provide context-dependent perspectives over a given shared information repository. This chapter presents an approach to database modularization where the modules represent application-specific perspectives over the shared database. The approach is meant to support the creation/definition of the modules as part of the conceptual schema definition process, that is to say the modules and the database they are a subset of are simultaneously defined. This is similar to Cyc's approach to ontological microtheories definition. The chapter develops both intuitive and formal definition of the proposed approach. It also shows the basics of how the modules are used by user transactions and of how the overall multiperception database can be implemented on a commercial database management system.

5.1 Introduction

A database stores a *representation* of the part of the real world that is of interest for a set of applications. Usually, information requirements vary from one application to another and call for different representations of the real world. For example, given a database describing vineyards, one application may focus on production data (e.g., which wines, which qualities and quantities) while another application focuses on cultivation aspects (e.g., which plants, fertilizers, harvesting techniques). Traditional database models[4] poorly comply with such situations as they do not explicitly support the definition of several representations for the same real-world phenomenon. Database designers have the choice between two unsatisfactory solutions. One is to define two tables (assuming a relational database) with different names (e.g., VineyardProduction and VineyardCultivation), each one with its attributes. To maintain the consistency of two instances (one in each table) describing the same vineyard, integrity constraints have to be defined to force attributes shared by the two focuses

[4] This chapter employs the database terminology. The term "model" means the schema language that allows designers to define the schema (description) of their database.

(e.g., an attribute holding the surface of the vineyard) to have the same value in the two instances. Querying such a database is uneasy as the user has to pay attention on which tables to query (one or the other or both). The second solution, more frequently used, is to merge all the desired representations into a common unique representation, the schema of the database. The *view* mechanism is then used to construct alternate representations that differ from those stored. In the example, the designer would first define in the schema a unique Vineyard table with all attributes (those common to the two focuses as well as those relevant for only one focus). Second, the designer would define two views, a VineyardProduction view and VineyardCultivation view, where each view extracts from the Vineyard table the attributes relevant to the targeted application. Notice that this second solution can only cope with compatible representations, where differences can be readily adjusted using the facilities of the manipulation language (e.g., SQL). Unfortunately, there are situations where differences between representation requirements go beyond the restructuring capabilities of SQL. For example, two applications may need the same information, e.g., an attribute A, but in incompatible formats, which would typically lead current systems to define in the schema table two attributes with different names, A1 and A2, and have each application view separately recover the A attribute from the base table attributes A1 and A2. The drawback of this solution is that the system ignores that A1 and A2 represent the same information and is consequently not in a position to guarantee the consistency of the two representations. Situations of this kind typically arise when creating a federated database out of a set of existing databases that represent the same phenomena in different ways, or in geographic applications that need to store the spatial extent of objects at different spatial resolutions. Current database management systems (DBMS) and geographic information systems (GIS) provide a few tools for explicitly supporting *multiple representations*. DBMS use generalization/specialization links to provide users with several representations of the same real-world entity with different levels of details. Some GIS allow storing several geometries for each spatial object.

Thanks to the flexibility it supports and to its relative simplicity, the view mechanism has become extremely popular with database users and designers, and has also influenced work on ontology modularization (see Part II of this book). This is despite the fact that views do not provide a complete solution to the problem. Inherently to the approach, each view is a single virtual table whose instances are derived from the stored database. Most applications need instead access to a virtual database holding (as any database does) sets of interrelated data from different tables. In current relational technology, these applications need to define one view per table they need, make sure they do not loose the external key defining the connections between the tables (e.g., use precomputed joins rather than the original tables, as referential integrity between the views is not supported), and acquire access rights to all their view tables. This is not that simple and risks of inconsistency are high. The idea of a virtual database has been initially proposed in the 1960s under the term subschema and was implemented in legacy systems such as Codasyl DBMSs. Unfortunately (for application developers) it was discarded once the view mechanism was invented for the benefit of DBMS developers.

As the name says, *subschemas* relied on the idea that each application needs only a subset of the database. In this chapter we propose an approach to resume this idea, while making it more general. Instead of associating an application with a subschema we make it possible for each application to have its own schema (and

database) while keeping the correlations with the schemas of the other applications. All application schemas (and databases) are stored within a single database, which we call a "multiperception" database. We say each application has its own "perception" of the multiperception database, and automatically gets from the DBMS the data corresponding to its perception. Equally correct would be to say that we change a traditional database into a multiperception database, and then each application can have its subschema corresponding to its perception of the multiperception database. To be precise, a *perception* in our approach is defined as the set of representations of all objects and links corresponding to a specific usage of the database. For instance, in a geographic database used for producing maps of a country at two different scales, say 1:20'000 for hikers and 1:300'000 for car drivers, it would be useful to group in a first perception all the 1:20'000 representations and in another one all the 1:300'000 representations. Users of this database could then open the database with the perception they need and get the corresponding homogeneous set of representations, i.e., a virtual, consistent single perception database.

The multiperception idea provides a possible approach for modularization. Given a database DB that one wants to modularize into modules $M_1, \ldots, M_n$, the process simply requires to define $M_1, \ldots, M_n$ as the desired perceptions and then tag each element of the database with the perception(s) it belongs to. This applies whatever the chosen technique for splitting the database into modules is. The approach is also independent of the data model (relational, UML, ...) used by the database designer, and can therefore also apply to ontologies. Indeed, a very similar approach is the one followed by Cyc to define *microtheories* within an ontology [Cyc06].

However, implementing a multiperception approach is not just a matter of using a set of tags. Its full specification requires the definition of tagging consistency rules to guarantee that each perception defines a coherent database, the definition of how a multiperception database can be manipulated by applications that may or may not want to share information, and the definition of how interperception processes can be supported, in particular with interperception links.

This chapter describes the capabilities we have defined as an answer to this need for multiperception data. These capabilities are embedded into a conceptual data model, Mads, that we had developed for classic as well as geographic and temporal databases. The Mads mechanism for multiple perceptions and representations allows any kind of element of the database to have several representations, and allows each user to get his/her own perception of the database. In the following we use Mads terminology, in particular the term "perception", which the reader can read as "module".

Mads is primarily intended for database designers, i.e., persons in charge of specifying the schema of a database in response to user/application requirements. Thus, it is a *conceptual* model: It enables a direct mapping between the perceived world and its representation. Using Mads, designers can focus exclusively on the requirements of their applications without having to care about implementation concerns. Mads is complemented with data manipulation languages that allow users to specify queries and updates at the conceptual level too. A set of tools developed during the European project MurMur automatically implements the conceptual specifications (schema or query) onto a DBMS or GIS [PSZ06b].

A Mads database is defined from the very beginning as containing a set of objects and relationships that may be shared by several perceptions, each object or relationship being possibly perceived in a different way for each perception. Using ontology

terminology we would say that the Mads perceptions share a common interpretation domain: There is a unique global set of object identifiers (oid) and a unique global set of relationship identifiers (rid) which are common to all perceptions. This approach is different from Cyc where assertions of two different microtheories (which are not a super- and its sub-microtheory) are always independent. The basic principles of Mads multiple perceptions and representations are: 1) Any database element, be it composite (e.g., an object type) or atomic (e.g., a simple attribute) may have various representations, one per perception. Any two perceptions may share any kind of element of the database. 2) Two objects belonging to two different perceptions may be linked by a binary relationship or by a multi-instantiation link (is-a or overlap link). These points are developed in the following sections.

Section 5.2 sets the Mads framework by giving an overview of the characteristics of the Mads data model, excluding the perceptions and representations aspect. Section 5.3 defines the various kinds of perceptions, and shows how to design and use perceptions. Sections 5.4 and 5.6 present, respectively, the various kinds of interperception links and the dependencies between perceptions that these links generate. Section 5.8 describes how to implement the Mads model in the relational model, while Section 5.5 gives a formal definition of the Mads model with the perception dimension. Section 5.9 compares the Mads approach with the ones of modular ontologies. Finally, Section 5.10 concludes this chapter and points to future research.

5.2 An Overview of the Mads Data Model

This section briefly presents the thematic, spatial, and temporal modeling dimensions of the Mads data model. All three dimensions can provide criteria for modularization. For example, spatial resolution is frequently used by geographic data providers to build modules that target production of maps at some specific scale. Maps at different levels of detail require data representations tailored to a specific user population: pedestrians, hikers, cyclists, car drivers, truck drivers, trip planners, etc. Similarly, temporal features may be used to identify modules whose data is relevant for a specific timeframe, e..g. the enterprise financial data for this year, for the year before, etc. For sake of brevity, the discussion of the perception dimension (Sections 5.3 to 5.6) does not explicitly address its relationships to spatiotemporal features. They are addressed implicitly by considering them as included in the generic structural concept of attribute. In particular, we only provide a formal definition of structural constructs for a multiperception database. However, the running example used in this chapter uses data with spatial and temporal features.

Readers interested in more detailed presentations of the MADS concepts and rules, including the formal definition of the model, may refer to [PSZ06a, SPZ07, APS07]. The perception and representation characteristics are described and discussed in Sections 5.3 to 5.6. Unless the contrary is explicitly stated, examples in this section refer to Fig. 5.1, which describes districts that are composed of land plots, where some land plots may be built up while others are agricultural and, in particular, vineyards.

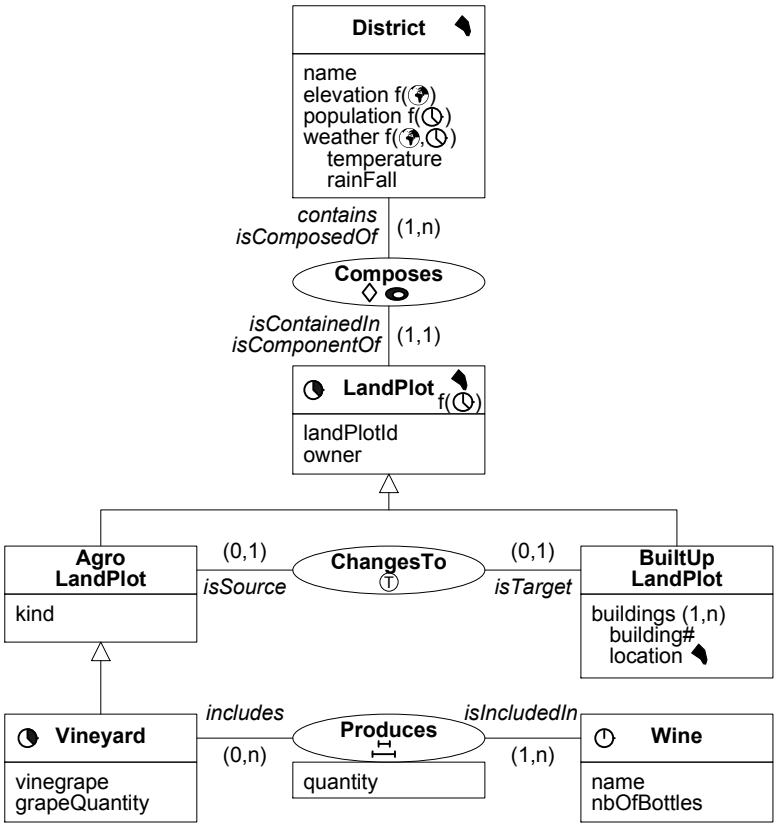

Fig. 5.1. The Mads schema of a spatio-temporal database.

5.2.1 Structural Modeling

We first give an informal presentation. A formal one follows later. Mads structural dimension describes the chosen data structures based on well-known features such as objects and object types, relationships and relationship types, attributes, and methods[5]. Objects and relationships have a system-defined identity, called oid for objects and rid for relationships. Both objects and relationships may bear attributes. Attributes may be mono-valued or multivalued, simple or complex (i.e., composed of other attributes), optional or mandatory, and may be derived (i.e., their value is computed from the values of other attributes). Referring to the example in Fig. 5.1, the attributes weather of District and buildings of BuiltUpLandPlot are both complex attributes, while all other attributes are simple. The buildings attribute is multivalued, as shown by the (1,n) notation following the attribute name: it describes the set of buildings located within the land plot, giving for each building its number

[5] For space reasons, we do not provide in this paper a detailed discussion about methods in the Mads model.

and its spatial extent. All other attributes are monovalued, with default cardinality (1,1) not shown in the figure.

Semantic data models usually provide the capability to link objects through various types of relationships, each one holding a specific semantics. Mads separates the definition of relationships into two facets: (1) its structure (i.e., the roles linking object types and the relationship attributes, if any), (2) its semantics. Each relationship type may bear zero, one, or several specific semantics. Mads supports aggregation, generation, transition, topological, synchronization, and inter-representation semantics for the relationships. Aggregation (identified by the ◊ icon) is the most common one: It defines mereological (also termed component or part-of) semantics. An example is the Composes relationship. Generation relationships record that target objects have been generated by source objects. Transition semantics expresses that an object in a source object type has evolved to a new state that causes it to be instantiated in another target object type. For example, in the schema diagram of Fig. 5.1 the ChangesTo relationship type holds transition semantics (denoted by the Ⓣ icon) expressing that an agricultural land plot may become a built-up land plot. Instances of this relationship type record such transitions.

By definition, transition relationships link two instances of the same object in different states. The fact that an object may have two (or more) instances in different object types[6] is known as *multi-instantiation*. In most semantic data models multi-instantiation is supported through *is-a links*, which by definition relate two instances (one generic, one specific) of the same object (or relationship). Mads adds a complementary kind of multi-instantiation link between either object or relationship types: The *overlap link*. Overlap links are binary links expressing that the two linked object (or relationship) types may contain instances sharing the same identity. They have a less constraining semantics than the inclusion semantics of the is-a link. Overlapping is implicit between two types that share a common subtype. Otherwise it has to be explicitly defined as in databases, contrarily to description logics, two object (or relationship) types that are not related by multi-instantiation links always hold disjoint sets of instances. In other words, they cannot contain two object (or relationship) instances sharing the same oid (or rid). For example, Fig. 5.1 shows that District, LandPlot, and Wine are three disjoint object types. Conversely, Vineyard, AgroLandPlot, LandPlot, and BuiltUpLandPlot form a network of overlapping types: As the ChangesTo transition relationship type implicitly defines an overlap link between AgroLandPlot and BuiltUpLandPlot, a LandPlot object can have instances in any of the four object types.

Multi-instantiation in Mads is by default dynamic: Any object (or relationship) may acquire new instantiations or loose existing instantiations in any of the object (or relationship) types connected by the network of multi-instantiation links it belongs to. This is the case for the ChangesTo relationship. However, database designers may use explicit integrity constraints to constrain multi-instantiation within a set of related types to be static. In this case, an object or relationship, once initially created as instance of one or more types in the constrained set, cannot change its membership, i.e., it cannot acquire a new instantiation nor loose any but all existing ones (which means the object is deleted from the database).

[6] In the case of temporal object types, the object instances linked by a transition relationship may be no longer active. Indeed, disabled instances of temporal types are kept in the database as long as they are needed by the applications.

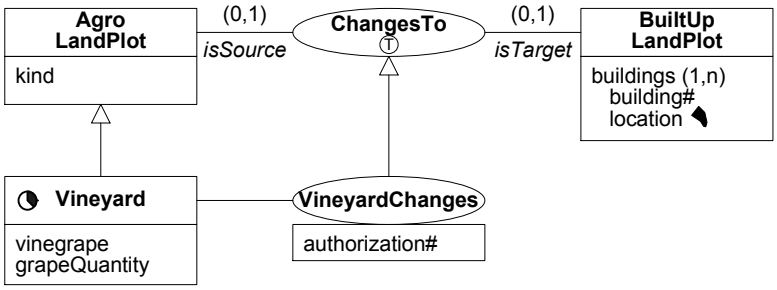

Fig. 5.2. A relationship subtype refining a role to link a subtype of the original object type.

Mads supports is-a links for object and relationship types with inheritance and possibly refinement or redefinition. Such capability is needed for full flexibility in defining spatial and temporal features of subtypes (given that these features are conveyed by attributes with a fixed name). For example, in the Vineyard object type the lifecycle is redefined: It contains a time interval describing when the vineyard was productive, instead of the time interval describing when the land plot was created and deleted. An example of refinement is given in Fig. 5.2, which specifies the VineyardChanges relationship type as a subtype of the ChangesTo transition relationship. The isSource role of the relationship type is refined to link only instances of AgroLandPlot that are instances of Vineyard too. This expresses that transitions of vineyards to built-up land plots are subject to an authorization, whose number is stored in the authorization# attribute.

5.2.2 Formal Definition of Structural Constructs

For the reader unfamiliar with data modeling concepts, this section provides formal definitions for the main structural constructs of the Mads model. Namely, for sake of simplicity, we leave out the spatial and temporal dimensions. In the structural dimension, we only include the concepts that exist in most Entity Relationship data models. We do not include multiassociations, relationship semantics, complex, multivalued, and optional attributes, weak object types, and methods. Let us call SimpleMads this subset of the Mads model that we formalize below. The interested reader can refer to [PSZ06a] for a full and formal description of the additional capabilities. A formal definition of the temporal dimension in the context of description logics, with its associated temporal constraints and all inferred reasoning, has been presented in [APS07]. Below we also leave out the perception dimension. The formal definitions of multiperception schema, multiperception database and perception are given in the following sections.

Definition 1. (SimpleMads schema without perceptions)
A SimpleMads schema without perceptions is a tuple: $\Sigma = (\mathcal{L}, \text{REL}, \text{ATT}, \text{CARD}, \text{ISA}, \text{OVLP}, \text{KEY})$, such that:

- $\mathcal{L}$ is a finite alphabet partitioned into the sets: $\mathcal{O}$ (*object type* symbols), $\mathcal{R}$ (*relationship type* symbols), $\mathcal{A}$ (*attribute* symbols), $\mathcal{U}$ (*role* symbols), and $\mathcal{D}$ (*domain* symbols).

- REL (relationships) is a total function that maps a relationship type symbol R in $\mathcal{R}$ to an $\mathcal{U}$-labeled tuple over $\mathcal{O}$, $\text{REL}(R) = \langle U_1 : O_1, \ldots, U_k : O_k \rangle$, where $k \geq 2$ is the *arity* of R.
- ATT (attributes) is a partial function that maps an object or relationship type symbol X in $\mathcal{O} \cup \mathcal{R}$ to an $\mathcal{A}$-labeled tuple over $\mathcal{D}$, $\text{ATT}(X) = \langle A_1 : D_1, \ldots, A_h : D_h \rangle$.
- CARD (cardinalities) is a partial function $\mathcal{O} \times \mathcal{R} \times \mathcal{U} \rightarrow \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ that defines cardinality constraints associated to the roles of the relationship types. For a relationship type R such that $\text{REL}(R) = \langle U_1 : O_1, \ldots, U_k : O_k \rangle$, we use $\text{CMIN}(O_i, R, U_i)$ and $\text{CMAX}(O_i, R, U_i, P)$ to denote the first and second component of CARD.
- ISA (is-a links) is a transitive binary relation $\text{ISA} \subseteq (\mathcal{O} \times \mathcal{O}) \cup (\mathcal{R} \times \mathcal{R})$ that defines is-a links for object and relationship types.
- OVLP (overlap links) is a symmetric binary relation $\text{OVLP} \subseteq (\mathcal{O} \times \mathcal{O}) \cup (\mathcal{R} \times \mathcal{R})$ that defines overlapping links between object or relationship types.
- KEY is a binary relation, $\text{KEY} \subseteq (\mathcal{O} \cup \mathcal{R}) \times 2^{\mathcal{A}}$, which associates to each object and relationship type symbol a set of keys, each key being composed of a set of attributes of the object or relationship type. □

The model-theoretic semantics associated with the SimpleMads model without the perception dimension is given next.

Definition 2. (Database state of a SimpleMads schema without perceptions)
Let Σ be a SimpleMads schema without perceptions. A *database state* for the schema Σ is a tuple $\mathcal{B} = (\Delta^{\mathcal{B}}_{\mathcal{O}} \cup \Delta^{\mathcal{B}}_{\mathcal{R}} \cup \Delta^{\mathcal{B}}_{D}, \cdot^{\mathcal{B}})$, such that: the three sets $\Delta^{\mathcal{B}}_{\mathcal{O}}$, $\Delta^{\mathcal{B}}_{\mathcal{R}}$, and $\Delta^{\mathcal{B}}_{D}$ are pairwise disjoint; $\Delta^{\mathcal{B}}_{\mathcal{O}}$ is a nonempty set of objects; $\Delta^{\mathcal{B}}_{\mathcal{R}}$ is a nonempty set of relationships, $\Delta^{\mathcal{B}}_{D} = \bigcup_{D_i \in \mathcal{D}} \Delta^{\mathcal{B}}_{D_i}$ is the set of values for all domains used in the schema Σ; and $\cdot^{\mathcal{B}}$ is a function that maps:

- Every domain symbol D_i to a set $D_i^{\mathcal{B}} = \Delta^{\mathcal{B}}_{D_i}$.
- Every object type symbol O, to a set $O^{\mathcal{B}} \subseteq \Delta^{\mathcal{B}}_{\mathcal{O}}$.
- Every relationship type symbol R to a set $R^{\mathcal{B}}$ of couples $\langle r, u \rangle$ where $r \in \Delta^{\mathcal{B}}_{\mathcal{R}_l}$ and u is a $\mathcal{U}$-labeled tuple over $\Delta^{\mathcal{B}}_{\mathcal{O}}$ such that if $\text{REL}(R) = \langle U_1 : O_1, \ldots, U_k : O_k \rangle$, then: $\langle r, u \rangle \in R^{\mathcal{B}} \wedge u = \langle U_1 : o_1, \ldots, U_k : o_k \rangle \Rightarrow \forall i \in \{1, \ldots, k\}\ (o_i \in O_i^{\mathcal{B}})$. Further, $R^{\mathcal{B}}$ is such that: $\forall \langle r_1, u_1 \rangle, \langle r_2, u_2 \rangle \in R^{\mathcal{B}}\ (r_1 = r_2 \Rightarrow u_1 = u_2)$.
- Every attribute symbol A to a set $A^{\mathcal{B}} \subseteq (\Delta^{\mathcal{B}}_{\mathcal{O}} \cup \Delta^{\mathcal{B}}_{\mathcal{R}}) \times \Delta^{\mathcal{B}}_{D}$, such that, for each object or relationship type $X \in (\mathcal{O} \cup \mathcal{R})$, if $\text{ATT}(X)[A] = D_i$, then: $x \in X^{\mathcal{B}} \Rightarrow (\exists a_i \in D_i^{\mathcal{B}}\ (\langle x, a_i \rangle \in A^{\mathcal{B}}) \wedge \forall a_i \in D_i^{\mathcal{B}}\ (\langle x, a_i \rangle \in A^{\mathcal{B}} \Rightarrow a_i \in \Delta^{\mathcal{B}}_{D_i}))$. □

Definition 3. (Consistent database state of a SimpleMads schema without perceptions)
A database state $\mathcal{B}$ is said to be consistent if it satisfies all of the constraints expressed in the schema:

- *Population inclusion*:
  $\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R})\ (\text{ISA}(X_1, X_2) \Rightarrow X_1^{\mathcal{B}} \subseteq X_2^{\mathcal{B}})$.
- *Population intersection*:
  $\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R})\ (X_1^{\mathcal{B}} \cap X_2^{\mathcal{B}} \neq \emptyset \Rightarrow X_1 = X_2 \vee \text{ISA}(X_1, X_2) \vee \text{ISA}(X_2, X_1) \vee$
  $\text{OVLP}\ (X_1, X_2))$

- *Cardinality constraints*:
  For each cardinality constraint $\text{CARD}(O, R, U)$ of a relationship $R \in \mathcal{R}$: $\forall o \in O^{\mathcal{B}}$ $(\text{CMIN}(O, R, U) \leq \#\{\langle r, u\rangle \in R^{\mathcal{B}} \mid u[U] = o\} \leq \text{CMAX}(O, R, U))$.
- *Key constraints*:
  For each key constraint $\text{KEY}(X, K)$ of an object or relationship type $X \in (\mathcal{O} \cup \mathcal{R})$, where $K = \{A_1, \ldots, A_n\}$:
  $\forall x_1, x_2 \in X^{\mathcal{B}}\ \forall i \in \{1, \ldots, n\}$ $(\langle x_1, a_i^1\rangle \in A_i^{\mathcal{B}} \wedge \langle x_2, a_i^2\rangle \in A_i^{\mathcal{B}} \wedge a_i^1 = a_i^2) \Rightarrow$
  $x_1 = x_2)$. □

Proposition 1. (Logical implication for a SimpleMads schema without perceptions)
As a consequence of the definitions of SimpleMads schema and consistent database state, the following rule can be derived:

- *Inferred overlap links from is-a links*:
  $\forall X_1, X_2, X_3 \in (\mathcal{O} \cup \mathcal{R})$ $(\text{ISA}(X_1, X_2) \wedge \text{ISA}(X_1, X_3) \Rightarrow \text{OVLP}(X_1, X_3))$. □

5.2.3 Spatio-Temporal Modeling

In Mads, space and time description is orthogonal to data structure description, which means that the description of a phenomenon may be enhanced by spatial and temporal features whatever data structure (i.e., object, relationship, attribute) has been chosen to represent it. Mads allows describing spatial and temporal features with either a discrete or a continuous view. These are described next.

The *discrete view* (or *object view*) of space and time defines the spatial and temporal extents of the phenomena of interest. The spatial extent is the set of 2-dimensional or 3-dimensional points (defined by their geographical coordinates $\langle x, y\rangle$ or $\langle x, y, z\rangle$) that the phenomenon occupies in space. The temporal extent is the set of instants that the phenomenon occupies in time. Temporality in Mads corresponds to valid time, which conveys information on when a given fact, stored in the database, is considered valid from the application point of view.

Specific data types support the definition, manipulation, and querying of spatial and temporal values. Mads supports two hierarchies of dedicated data types, one for spatial data types, and one for temporal data types. Generic spatial (respectively, temporal) data types allow describing object types whose instances may have different types of spatial extents. For example, a River object type may contain instances for large rivers with an extent of type Surface and instances for small rivers with an extent of type Line. The Mads hierarchy of spatial data types is simpler that – while compatible with – the one proposed by the Open Geospatial Consortium [Ope06]. Examples of spatial data types are: Geo (), the most generic spatial data type, Surface (), and SurfaceBag (). The latter is useful for describing objects with a non-connected surface, like an archipelago. Examples of temporal data types are: Instant (), Interval (), and IntervalBag (). The latter is useful for describing the periods of activity of non-continuous phenomena.

A spatial (temporal) object type is an object type that holds spatial (temporal) information pertaining to the object itself. For example, District is a spatial object type as shown by the surface () icon on the right of its name, and LandPlot is a spatial and temporal object type with a lifespan of kind Interval (icon on the left

of its name). Following common practice, we call spatio-temporal an object type that either has both a spatial and a temporal extent, separately, or has a time-varying spatial extent, i.e., its spatial extent changes over time and the history of extent values is recorded (e.g., LandPlot). Similarly, spatial, temporal, and spatio-temporal relationship types hold spatial and/or temporal information pertaining to the relationship as a whole, exactly as for an object type. Time-varying and space-varying attributes are described hereinafter.

The spatial and temporal extents of an object (or relationship) type are kept in dedicated system-defined attributes: geometry for the spatial extent and lifecycle for the temporal extent. The attribute geometry is a spatial attribute (see below) with any spatial data type as domain. When representing a moving or deforming object (e.g., LandPlot), geometry is a time-varying spatial attribute. On the other hand, the attribute lifecycle allows database users to record when, in the real world, the object (or link) was (or is planned to be) created and deleted. It may also support recording that an object is temporarily suspended, like an employee who is on temporary leave. Therefore, the lifecycle of an instance says at each instant what is the status of the corresponding real-world object (or link): scheduled, active, suspended, or disabled.

A spatial (temporal) attribute is a simple attribute whose domain of values belongs to one of the spatial (temporal) data types. Each object and relationship type, whether spatial, temporal, or plain, may have spatial, temporal, and spatio-temporal attributes. For example, the BuiltUpLandPlot object type includes, in addition to its spatial extent (inherited from LandPlot), a complex and multivalued attribute buildings whose second component attribute, location, is a spatial attribute describing, for each building, its spatial extent, a surface. Practically, the implementation of a spatial attribute, as well as the one of a geometry attribute, varies according to the domain of the attribute. For instance, in 2D space a geometry of kind Point is usually implemented by a couple of coordinates $\langle x, y \rangle$ for each value, and a geometry of kind Surface by a list of couples $\langle x, y \rangle$ per value.

Spatial and temporal values for an object may have to be consistent with the spatial and temporal values of other related objects. *Constraining relationships* are binary relationships linking spatial (or temporal) object types stating that the geometries (or lifecycles) of the linked objects must comply with a spatial (or temporal) constraint. For example, Composes is both an aggregation and a constraining relationship of kind topological inclusion, as shown by the ⊙ icon. The constraint states that a district and a land plot may be linked only if the spatial extent of the district effectively contains the spatial extent of the land plot. Produces is a synchronization relationship type of kind within (icon): It enforces the temporal extent of the Wine instance – an instant with year granularity describing the year of the wine – to be included within the temporal extent of the Vineyard instance – a time interval describing when the vineyard was productive.

Beyond the discrete view, there is a need to support another perception of space and time, the *continuous view* (or *field view*). In the continuous view a phenomenon is perceived as a function associating to each point (or instant) of a spatial (or temporal) extent a value. Mads supports the continuous view using space- and time-varying attributes, which are attributes whose value is a function that records the history – and possibly the future – of the value. The domain of the function is a spatial (and/or temporal) extent. Its range can be a set of simple values (e.g., Real for temperature, Point for a moving car), a set of composite values if the attribute is complex, and/or a powerset of values if the attribute is multivalued.

The object type District shows three examples of varying attributes and their visual notation in Mads (e.g., f(🌍)). Attribute elevation is a space-varying attribute defined over the geometry of the district: It provides for each geographic point of the district its elevation. Attribute population is a time-varying attribute defined over a constant time interval, e.g., [1900-2007]. Attribute weather is a space and time-varying complex attribute which records for each point of the spatial extent of the district and for each instant of a constant time interval a composite value describing the weather at this location and this instant. Such space- and time-varying attributes are also called spatio-temporal attributes. As we have seen, the geometry attribute can also be time varying, like any spatial attribute. For instance, LandPlot has a time-varying geometry: any change of the spatial extent of land plots can therefore be recorded. Practically, the implementation of a continuous time-varying attribute is usually made up of (1) a list of $\langle \text{instant}, \text{value} \rangle$ pairs that records measured values (called sample values), and 2) a method that performs linear interpolation between two sample values to infer non-measured values. For instance, a time-varying point would be implemented by a list of triples $\langle \text{instant}, x, y \rangle$. On the other hand, time-varying attributes that are not continuous but that vary in a stepwise manner, like the geometry of LandPlot, are recorded by a list of couples $\langle \text{time interval}, \text{value} \rangle$.

A constraining topological relationship may link moving or deforming objects, i.e., spatial objects whose geometries are time-varying. An example is the topological inclusion relationship Composes that links District (a surface) and LandPlot (a time-varying surface). In this case two possible interpretations can be given to the topological predicate, depending on whether it must be satisfied either for at least one instant or for every instant belonging to the time extent of the varying geometries. Applied to the example of Fig. 5.1, this means that the relationship Composes can only link a District and a LandPlot instances such that their geometries intersect for at least one instant or for every instant of the temporal extent of the varying geometry of the land plot. When defining the relationship type, the designer has to specify which interpretation holds.

5.3 Perceptions

As explained in the introductory section of this chapter, the notion of perception in Mads captures a specific perspective that guides the definition of the corresponding content of the database. We first discuss the perception mechanism informally, and provide a formal definition afterwards. As Mads is intended for conceptual modeling, the definition of perceptions is dealt with as part of the conceptual design phase. The resulting conceptual schema will eventually be translated into logical and physical schemas. Perceptions, alike spatial and temporal features, will have to be implemented using the mechanisms provided by the target DBMS. We show in Sect. 5.8 a possible implementation of perceptions into the relational model.

Supporting multiple perceptions within the same database, as Mads does, means that different contents coexist in the database and the system knows how to identify and extract the content that corresponds to a specific perception (which we call *simple* perception) or to a combination of perceptions (which we call *composite* perception). For instance, the schema diagram in Fig. 5.3 illustrates a multiperception schema, separately showing the content of each of three simple perceptions designed to support information requirements from the wine makers, the wine experts, and

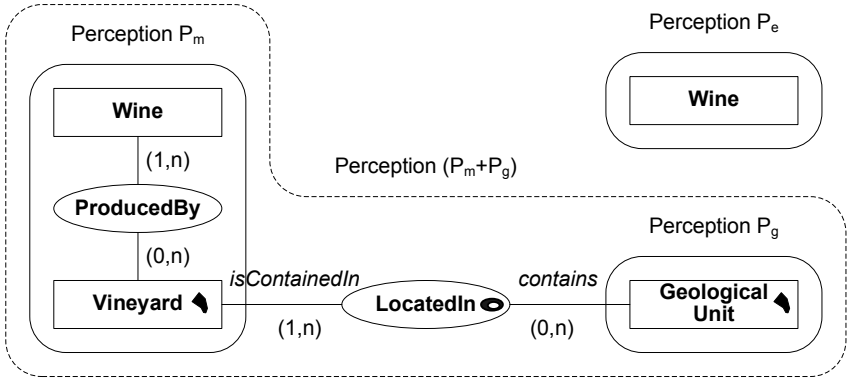

Fig. 5.3. A schema diagram showing three simple perceptions and one composite perception.

the geologists working in the wine area. These perceptions are denoted P_m, P_e, and P_g, respectively. The diagram uses visual duplication to show that the object type Wine belongs to two simple perceptions. Before accessing the database, a typical user transaction will specify which perception it wants to use, and will accordingly see the corresponding subset of the database.

Some applications may need to work simultaneously with data that has been defined as belonging to different perceptions. For example in Fig. 5.3, an application may wish to relate the geological information to the vineyard information, thus spanning over the geologist and wine makers perceptions. Such an application may wish to record the relationships between geological units in P_g and vineyards in P_m, creating instances of the LocatedIn relationship type. Similarly, applications may need to simultaneously use different *representations* of the same phenomena belonging to different perceptions. For instance, in cartographic databases storing data for a set of maps representing the same region at different scales, it is common to organize the database as holding one simple perception per targeted scale. Yet there are applications that can compare the various representations in order to check their spatial consistency.

In summary, the perception mechanism must be able to support users using a single simple perception as well as users using data from multiple perceptions. It must also be able to support storing data that belong to a single perception, data that belong to multiple perceptions, and data that relate together data from different perceptions. We describe hereinafter a mechanism to respond to these requirements based on the combined use of simple and composite perceptions.

A *simple perception* provides an application with a view of the multiperception database that includes whatever data is defined as belonging to this perception, and nothing else. P_m, P_e, and P_g, in Fig. 5.3 are simple perceptions. A multiperception database holds data belonging to various simple perceptions, say $(p_1 + p_2 + \ldots + p_n)$. A simple perception can be seen as a component of a multiperception database characterized by its own schema and its own instances (both materialized), respectively a subset of the multidatabase schema and instances. This subset is equivalent to a traditional database without the perception dimension. The various perceptions

may differ in their scope, i.e., they may describe different sets of real-world entities and links, but these sets may also overlap and in databases they often do overlap in a large proportion. For instance, in Fig. 5.3 both perceptions P_m and P_e describe wines, possibly in different ways. The definition of simple perceptions is part of the schema design process, now ending up with the (basically static) definition of a multiperception schema.

Composite perceptions support working with data from multiple simple perceptions. A *composite perception* is dynamically defined by users depending on the information needs of their transactions. Transactions use an openDatabase command to specify which database they want to work with and which perception(s) they want to work with:

openDatabase(dbName, myView)

where myView is either a simple perception p_i or a composite perception denoted $(p_1 + p_2 + \ldots + p_k)$,

The view provided by a composite perception is created by the system on the fly and contains all the elements (schema and instances) of the component simple perceptions p_i, plus the *interperception links* (relationships, is-a and overlap links, at the schema and instance levels), if any, that relate objects in different perceptions within p_1, p_2, ..., and p_k (e.g., an object of p_1 and an object of p_2). For instance, in the Wine database of Fig. 5.3 perceptions P_m and P_g describe two disjoint parts of the real world, yet they are linked by a relationship type, LocatedIn. This relationship type, contrarily to ProducedBy, does not belong to any of the simple perceptions P_m, P_e, and P_g. It belongs only to the composite perception $(P_m + P_g)$. Therefore, while users of P_m see Wine, Vineyard, and ProducedBy, and users of P_g see GeologicalUnit, users of $(P_m + P_g)$ see Wine, Vineyard, ProducedBy, GeologicalUnit, and LocatedIn.

Interperception links provide an explicit means to navigate between perceptions. By definition, they do not belong to any simple perception. For simplification purposes, we keep with the idea that every element belongs to at least a simple perception by considering that interperception links belong to a special simple perception that is system defined and not visible to users, and is denoted P_{ip}. Referring to Fig. 5.3, the ProducedBy relationship type links two P_m object types and is defined by the administrator as belonging to P_m: We say it is a *local* link. LocatedIn, instead, links a P_m object type and a P_g object type, and is therefore automatically identified as an interperception link, implicitly tagged P_{ip}. The set of local links and interperception links are disjoint. In the schema illustrated in Fig. 5.5, there is one relationship type ProducedBy which is local, even if it belongs to two perceptions, P_m and P_e. ProducedBy in the P_m (resp. P_e) perception links Wine objects that belong to P_m (resp. P_e) to Vineyard objects that also belong to P_m (resp. P_e). Should the application need to link, say, wines of P_m to vineyards of P_e, then the database administrators would have to define another relationship type, say WmProducedByVe, linking Wine objects of P_m to Vineyard objects of P_e.

The first step towards the creation of a multiperception database is for the database administrator to identify the set of simple perceptions that need to be explicitly defined (i.e., the set $SP = \{p_1, p_2, \ldots, p_n\}$). The following step for the database administrator is to define which data belongs to which perception. Any kind of schema element may have several representations. The population of an object or relationship type may also vary with the perception. Whatever methodology is used

to perform this step (definitions organized by perception or by schema element), the result shall conform to the following:

- Each object and relationship type definition includes the specification of the perceptions it belongs to, and for each of these perceptions its corresponding representation, i.e., its attributes, and roles definitions.
- Each *interperception* relationship type*interperception relationship type*, meant to connect objects in different perceptions, is defined as belonging to the peculiar perception P_{ip}.
- An element that has a single representation may belong to multiple perceptions. All its perceptions share its single representation. Conversely, an element can have multiple representations only if it belongs to at least as many perceptions (otherwise stated, an element has only one representations for each perception).
- Each perception has to denote a consistent database obeying the classical consistency rules for databases (e.g., no pending role in a relationship).
- To enforce perception consistency, an element a that is a component of an element b (e.g., an attribute of an object type or a component attribute of a complex attribute) can only belong to perceptions to which the b element belongs, as illustrated in the example of Fig. 5.4.

Figure 5.4 illustrates the definition of a multiperception object type, providing details about its perceptions, and attributes and keys for each of the two perceptions. The drawing of the two perceptions of the Wine object type as a single object type in which the two perceptions are merged is different from the drawing of the same Wine object type in Fig. 5.3 as two boxes, one per perception. Yet the difference only conveys the use of different visualization techniques. The information content is the same. Figure 5.4 directly corresponds to how Wine is defined using the Mads data definition language, which includes the definition of perceptions as part of the definition of each metadata element (i.e., data description element of the schema).

Figure 5.4 describes the representations of the Wine object type for the wine expert's perception P_e and for the wine maker's perception P_m. Attributes name, year, and wineType are common to both perceptions, with a common representation. Attributes degree and barrels are common to both perceptions, but they have a different definition (representation) for each perception and therefore their values will be different too. The value of degree is simplified (integer rather than real) for the perception P_e. Similarly, the attribute barrels is a simple Boolean attribute in perception P_e, stating if the wine has been kept in wooden barrels or not, while in perception P_m it is a complex attribute describing the time period during which the wine has been kept in barrels and the kind of wood of the barrels. Perception P_e has several attributes, rating, color, body, sugar, and food (the food matching the wine) that are specific to it and do not exist in P_m.

As shown, the representations hold by an object (relationship) type may have different sets of attributes, different characteristics for a common attribute (different cardinalities or value domains). Perceptions are also defined at the instance level. Therefore, an object (relationship) type belonging to several perceptions may have different sets of instances according to the perception. Similarly each instance (object or relationship) that belongs to several perceptions may have different values according to the perception. This is obvious when the sets of attributes are different for the various perceptions, but it is also true for an attribute with a unique definition. In this case, the value of the attribute depends upon the perception, and

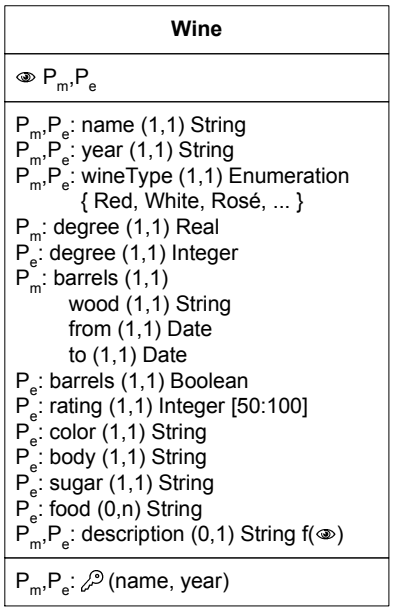

Fig. 5.4. The two perceptions of the Wine object type of Fig. 5.3.

the attribute is said to be *perception-varying*. For example, in Fig. 5.4 the attribute description, recording a text of a few lines describing the wine, is perception-varying, as identified by the f() icon. It has a unique definition common to both perceptions, but it has a different value for each perception, i.e., the text used by the wine maker is different from the text used by the wine expert.

Similarly to object types, when defining a relationship type the designer has to specify to which perceptions it belongs and for each perception its representation, i.e., defining its attributes and roles. Figure 5.5 illustrates two perceptions, P_m and P_e, sharing the relationship type ProducedBy and its linked object types Wine and Vineyard (this schema is different from the one illustrated Fig. 5.3). Let us assume that ProducedBy in P_m has a unique attribute quantity, and in P_e no attribute at all. Attribute quantity says how many kilograms of grapes harvested in this vineyard have been used for producing this wine. We also assume that the populations are different. In P_m, ProducedBy takes into account all contributing vineyards even if the quantity of grapes is small. In P_e, ProducedBy takes into account only the vineyards that have contributed to at least 15% of the total quantity of grapes used for producing this wine. Therefore, there is a constraint linking the two populations of ProducedBy as follows:

$$\text{population}(P_e.\text{ProducedBy}) \subseteq \text{population}(P_m.\text{ProducedBy})$$

that should be defined in the schema by an interperception is-a link.

Relationship types may have different representations for their roles (e.g., have different sets of roles according to the perception) and their semantics (e.g., being an aggregation relationship for a perception and a topological inclusion relationship for another).

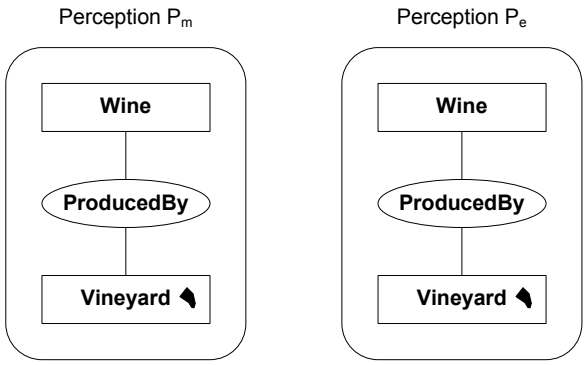

Fig. 5.5. A relationship type belonging to two perceptions.

Given two object types, their representations in different perceptions may be differently related, i.e., in a perception they may be related by an is-a link, in another perception they may be defined as disjoint, and yet in another one they may possibly overlap. They may also be linked by a relationship type in a perception, and not linked in another perception. Such flexibility is needed to allow independence between the perceptions.

Different representations for the same real-world entities and links may even contradict each other. For example, Fig. 5.6 shows a Mads schema with two perceptions. Perception P_1 considers humans to be a specific kind of animals. It therefore defines two object types, Human and Animal, linked by an is-a link making Human a subclass of Animal. Perception P_2 considers humans to be different from animals. It therefore contains another representation of the same two object types, possibly with different attributes and methods, where the two object types are by definition disjoint (they are not interrelated by a multi-instantiation link).

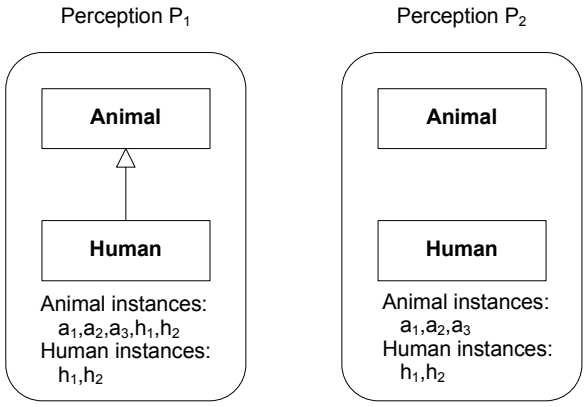

Fig. 5.6. Two perceptions that differ at the schema and the instance level (instances are symbolized by their oid).

In terms of constraints, the database administrator can define interperception constraints on the value of attributes and on instances. Examples of usual constraints for an object or relationship type are: The set of instances – more precisely, the set of oids or rids – is the same for all perceptions, or, on the contrary, they are disjoint. Another constraint could state that the set of instances for a given perception is included in the set for another perception. An example could be in Fig. 5.6: "Every instance of Animal that has a representation in P_2 has also a representation in P_1." This should be defined in the schema by an interperception is-a.

Particularly important constraints are the identification constraints. There is indeed a need for being able to correlate and coordinate the various perceptions if required by application rules. For example, the cartographic application we already mentioned needs to be able to find all representations of an object (e.g., a building) to check their consistency (e.g., the point representing the spatial extent of the building in one perception P_x has to be inside the area representing the same building in another perception P_y). Knowledge about the two representations of buildings is granted by the use of a composite perception ($P_x + P_y$), but this would not help if the user transaction is not able to identify, at the instance level, which P_x building is the same as a given P_y building. In our approach, the correlation between multiple representations of the same object (or relationship) relies on shared object (or relationship) identity, as is the case in semantic databases for the implementation of is-a links. All representations of an (object or relationship) instance share the same oid (or rid in case of a relationship instance), which is defined by the system. Identity provides the shared property that links together all the representations of the same instance. As in object-oriented systems, relying on identity, rather than on user-defined keys, guarantees that the system can keep a correct understanding of instances even if users enter erroneous data in the database.

Identity, however, is not enough. How would the system know that the P_x user inserting a building new to her is actually creating her representation of a building already inserted in the database by a P_y user? One solution would be to enforce that instances of shared object types, e.g., Building in both P_x and P_y, can only be created by users with the composite perception ($P_x + P_y$). This solution is in our opinion overly restrictive. We prefer the solution (adopted by relational DBMS) where users of multiperception elements rely on a shared identification mechanism, i.e., a shared key, to correlate the multiple representations of the same object or relationship. This solution is presented in more detail in Sect. 5.7.

5.4 More on Interperceptions Links

As we have already seen, two simple perceptions may describe either the same part of the real world, or disjoint or overlapping parts. The common part may be described by different representations of the same object types, like the two representations of Wine for perceptions P_m and P_e in Figs. 5.3 and 5.4, or the two representations of Animal for perceptions P_1 and P_2 in Fig. 5.6. However, a set of real-world entities may also be described in different perceptions by different object types. For example, Fig. 5.7 shows a variant of Fig. 5.3 where the wine expert's perception P_e, instead of providing a generic Wine object type provides three disjoint object types RedWine, WhiteWine, and RoseWine.

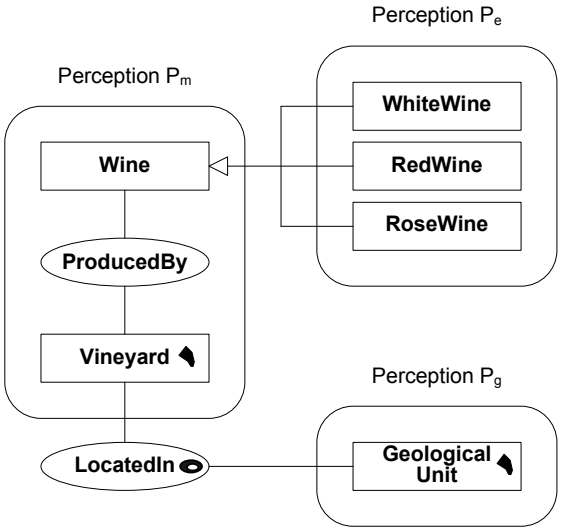

Fig. 5.7. Interperception is-a links

In this kind of situation where some instances of two object types belonging to two different perceptions describe the same real-world entities, designers using the Mads data model have two possibilities:

- If the mapping between the instances of the two object types is injective on both sides, the object types may be defined as sharing oids, i.e., an *interperception is-a* or *overlap link* can relate the two object types. As shown in Fig. 5.7, if we assume that all wines described in P_e are also described in P_m in the Wine object type, designers may assert:
  P_e.WhiteWine is-a P_m.Wine
  P_e.RedWine is-a P_m.Wine
  P_e.RoseWine is-a P_m.Wine
- If the mapping between the instances is not injective, i.e., an instance of an object type may correspond to several instances of the other object type, designers may relate these object types through an interperception relationship type. Mads supports a specific kind of semantics for these relationship types that link objects representing the same real-world entities, the *inter-representation* semantics. For example, let us assume a perception containing an object type Person and another one containing an object type Marriage. Designers could relate these two object types through an interperception and inter-representation relationship type that would link each instance of Marriage to two instances of Person, the husband and the wife.

Notice that, in the case of a mapping that is injective on both sides, designers may choose between the two solutions: either an interperception multi-instantiation link (is-a or overlap according to the cardinalities of the mapping) or an interperception and inter-representation relationship type. If the cardinalities of the mapping are (1,1)–(0,1), the is-a link is equivalent to an inter-representation relationship type with the same cardinalities. However, the is-a link is a more direct representation of

the semantics of the mapping and hence should be preferred. If the mapping is (0,1)–(0,1), the two solutions, an interperception overlap link and an interperception and inter-representation relationship type, are not equivalent. In the latter, each simple perception may create instances in its object type without worrying about the other perception. Afterwards, users of the composite perception can create the interperception and inter-representation relationship instances that will link together the corresponding instances of the two object types. On the other hand in the former solution, the interperception overlap, the creation of an instance i_1 in one object type, requires to know if the corresponding instance, say i_2, already exists in the other perception because the insertion of i_1 requires the oid of i_2, in order to create i_1 with the same oid.

In summary, Mads supports both interperception multi-instantiation links and interperception relationship types, thus allowing designers to explicitly describe many kinds of situations where the real world described by two perceptions overlap.

5.5 Formal Definition of a SimpleMads Multiperception Database

In this section we give a formal definition of a Mads multiperception database. If all specifications related to perceptions are taken out of the following definition, the definition reduces to the definition of a Mads database without perceptions, which we provided in Sect. 5.2.2. We keep here the same simplifying assumptions as in Sect. 5.2.2. For sake of simplicity, we omit in this formalization the definition of perception-varying attributes.

The model-theoretic semantics associated with the SimpleMads model with the perception dimension is given next.

Definition 4. (SimpleMads multiperception schema)
A SimpleMads multiperception schema is a tuple: $\Sigma = (\mathcal{L}, \text{PERC}, \text{REL}_l, \text{REL}_{ip}, \text{ATT}, \text{CARD}_l, \text{CARD}_{ip}, \text{ISA}_l, \text{ISA}_{ip}, \text{OVLP}_l, \text{OVLP}_{ip}, \text{KEY})$, such that:

- $\mathcal{L}$ is a finite alphabet partitioned into the sets: $\mathcal{P}$ (*perception* symbols) $\mathcal{O}$ (*object type* symbols), $\mathcal{R}$ (*relationship type* symbols), $\mathcal{A}$ (*attribute* symbols), $\mathcal{U}$ (*role* symbols), and $\mathcal{D}$ (*domain* symbols). Further, $\mathcal{R}$ is partitioned into the sets $\mathcal{R}_l$ and $\mathcal{R}_{ip}$ denoting, respectively, the *local* and the *interperception* relationship type symbols. Also, $\mathcal{P} = \{P_{ip}\} \cup \mathcal{P}_s$ where P_{ip} is a peculiar perception to which are attached all interperception relationship types and $\mathcal{P}_s$ is the set of simple perception symbols.
- PERC (perceptions) is a total function that maps each object or relationship type symbol X in $\mathcal{O} \cup \mathcal{R}$ to a nonempty set of perceptions $\text{PERC}(X) \subseteq 2^{\mathcal{P}}$ such that $\forall X \in (\mathcal{O} \cup \mathcal{R}_l)\ (\text{PERC}(X) \subseteq \mathcal{P}_s \wedge \text{PERC}(X) \neq \emptyset)$ and $\forall R \in \mathcal{R}_{ip}\ (\text{PERC}(R) = \{P_{ip}\})$.
- REL_l (local relationships) is a total function that maps a couple made up of a local relationship type symbol R in $\mathcal{R}_l$ and a perception symbol P in $\text{PERC}(R)$ to an $\mathcal{U}$-labeled tuple over $\mathcal{O}$, $\text{REL}_l(R, P) = \langle U_1 : O_1, \ldots, U_k : O_k \rangle$, where $k \geq 2$ is the *arity* of R in P, and $\forall i \in \{1, \ldots, k\}\ (P \in \text{PERC}(O_i))$.
- REL_{ip} (interperception relationships) is a total function that maps an interperception relationship type symbol R in $\mathcal{R}_{ip}$ to an $\mathcal{U} \times \mathcal{P}_s$-labeled tuple over $\mathcal{O}$,

$\text{REL}_{ip}(R) = \langle (U_1, P_1) : O_1, \ldots, (U_k, P_k) : O_k \rangle$, where $k \geq 2$ is the *arity* of R, and $\forall i \in \{1, \ldots, k\}$ $(P_i \in \text{PERC}(O_i))$.

- ATT (attributes) is a partial function that maps a couple made up of an object or relationship type symbol X in $\mathcal{O} \cup \mathcal{R}$ and a perception symbol P in $\text{PERC}(X)$ to an $\mathcal{A}$-labeled tuple over $\mathcal{D}$, $\text{ATT}(X, P) = \langle A_1 : D_1, \ldots, A_h : D_h \rangle$.
- CARD_l (local cardinalities) is a partial function $\mathcal{O} \times \mathcal{R}_l \times \mathcal{U} \times \mathcal{P}_s \rightarrow \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ that defines cardinality constraints associated to the roles of the local relationship types in perceptions. For a local relationship type R and one of its perceptions P such that $\text{REL}_l(R, P) = \langle U_1 : O_1, \ldots, U_k : O_k \rangle$, we use $\text{CMIN}_l(O_i, R, U_i, P)$ and $\text{CMAX}_l(O_i, R, U_i, P)$ to denote the first and second component of CARD_l.
- CARD_{ip} (interperception cardinalities) is a partial function $\mathcal{O} \times \mathcal{R}_{ip} \times \mathcal{U} \times \mathcal{P}_s \rightarrow \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ that defines cardinality constraints associated to the roles of interperception relationship types. For an interperception relationship type R such that $\text{REL}_{ip}(R) = \langle (U_1, P_1) : O_1, \ldots, (U_k, P_k) : O_k \rangle$, we use $\text{CMIN}_{ip}(O_i, R, U_i, P_i)$ and $\text{CMAX}_{ip}(O_i, R, U_i, P_i)$ to denote the first and second component of CARD_{ip}.
- ISA_l (local is-a links) is a ternary relation $\text{ISA}_l \subseteq (\mathcal{O} \times \mathcal{O} \times \mathcal{P}_s) \cup (\mathcal{R}_l \times \mathcal{R}_l \times \mathcal{P}_s)$ that defines is-a links for object and relationship types in each perception. The transitive closure ISA_l^+ of ISA_l in each perception is defined as follows:
  $\forall X_1, X_2, X_3 \in (\mathcal{O} \cup \mathcal{R}_l)$ $\forall P \in \mathcal{P}_s$
  $\text{ISA}_l(X_1, X_2, P) \Rightarrow \text{ISA}_l^+(X_1, X_2, P)$
  $\text{ISA}_l^+(X_1, X_2, P) \wedge \text{ISA}_l^+(X_2, X_3, P) \Rightarrow \text{ISA}_l^+(X_1, X_3, P)$.
- ISA_{ip} (interperception is-a links) is a quaternary relation $\text{ISA}_{ip} \subseteq (\mathcal{O} \times \mathcal{P}_s \times \mathcal{O} \times \mathcal{P}_s) \cup (\mathcal{R}_l \times \mathcal{P}_s \times \mathcal{R}_l \times \mathcal{P}_s)$ that defines is-a links for object and relationship types belonging to different perceptions. ISA_{ip} is such that:
  $\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R}_l)$ $\forall P_1, P_2 \in \mathcal{P}_s$ $(\text{ISA}_{ip}(X_1, P_1, X_2, P_2) \Rightarrow P_1 \neq P_2)$.
  The transitive closure ISA_{ip}^+ of ISA_{ip} is defined as follows:
  $\forall X_1, X_2, X_3 \in (\mathcal{O} \cup \mathcal{R}_l)$ $\forall P_1, P_2, P_3 \in \mathcal{P}_s$
  $\text{ISA}_{ip}(X_1, P_1, X_2, P_2) \Rightarrow \text{ISA}_{ip}^+(X_1, P_1, X_2, P_2)$
  $\text{ISA}_{ip}^+(X_1, P_1, X_2, P_2) \wedge \text{ISA}_{ip}^+(X_2, P_2, X_3, P_3) \Rightarrow \text{ISA}_{ip}^+(X_1, P_1, X_3, P_3)$.
- OVLP_l (local overlap links) is a ternary relation $\text{OVLP}_l \subseteq (\mathcal{O} \times \mathcal{O} \times \mathcal{P}_s) \cup (\mathcal{R}_l \times \mathcal{R}_l \times \mathcal{P}_s)$ that defines overlapping links between object or relationship types for each perception. OVLP_l is symmetric for each perception:
  $\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R}_l)$ $\forall P \in \mathcal{P}_s$ $(\text{OVLP}_l(X_1, X_2, P) \Rightarrow \text{OVLP}_l(X_2, X_1, P))$.
- OVLP_{ip} (interperception overlap links) is a quaternary relation $\text{OVLP}_{ip} \subseteq (\mathcal{O} \times \mathcal{P}_s \times \mathcal{O} \times \mathcal{P}_s) \cup (\mathcal{R}_l \times \mathcal{P}_s \times \mathcal{R}_l \times \mathcal{P}_s)$ that defines overlapping links between object or relationship types belonging to different perceptions. OVLP_{ip} is such that:
  $\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R}_l)$ $\forall P_1, P_2 \in \mathcal{P}_s$ $(\text{OVLP}_{ip}(X_1, P_1, X_2, P_2) \Rightarrow P_1 \neq P_2)$.
  Further, OVLP_{ip} is symmetric:
  $\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R}_l)$ $\forall P_1, P_2 \in \mathcal{P}_s$ $(\text{OVLP}_{ip}(X_1, P_1, X_2, P_2) \Rightarrow$
  $\text{OVLP}_{ip}(X_2, P_2, X_1, P_1))$.
- KEY is a ternary relation, $\text{KEY} \subseteq (\mathcal{O} \cup \mathcal{R}) \times \mathcal{P} \times 2^{\mathcal{A}}$, which associates to each object and relationship type symbol and a given perception a set of keys, each key being composed of a set of attributes of the object or relationship type for the perception. □

The model-theoretic semantics associated with the SimpleMads model with the perception dimension is given next.

Definition 5. (Database state of a SimpleMads multiperception schema) Let Σ be a SimpleMads multiperception schema. A *database state* for the schema

Σ is a tuple $\mathcal{B} = (\Delta^{\mathcal{B}}_{\mathcal{O}} \cup \Delta^{\mathcal{B}}_{\mathcal{R}} \cup \Delta^{\mathcal{B}}_{D}, \cdot^{\mathcal{B}(P)})$, such that: the three sets $\Delta^{\mathcal{B}}_{\mathcal{O}}$, $\Delta^{\mathcal{B}}_{\mathcal{R}}$, and $\Delta^{\mathcal{B}}_{D}$ are pairwise disjoint; $\Delta^{\mathcal{B}}_{\mathcal{O}}$ is a nonempty set of objects; $\Delta^{\mathcal{B}}_{\mathcal{R}} = \Delta^{\mathcal{B}}_{\mathcal{R}_l} \cup \Delta^{\mathcal{B}}_{\mathcal{R}_{ip}}$ is a nonempty set of local and interperception relationships, where $\Delta^{\mathcal{B}}_{\mathcal{R}_l}$ and $\Delta^{\mathcal{B}}_{\mathcal{R}_{ip}}$ are disjoint; $\Delta^{\mathcal{B}}_{D} = \bigcup_{D_i \in \mathcal{D}} \Delta^{\mathcal{B}}_{D_i}$ is the set of values for all domains used in the schema Σ; and $\cdot^{\mathcal{B}(P)}$ is a function that, for some $P \in \mathcal{P}$, maps:

- Every domain symbol D_i, for every simple perception $P \in \mathcal{P}_s$, into a set $D_i^{\mathcal{B}(P)} = \Delta^{\mathcal{B}}_{D_i}$, such that:
  $\forall P_1, P_2 \in \mathcal{P}_s \ (D_i^{\mathcal{B}(P_1)} = D_i^{\mathcal{B}(P_2)})$.
- Every object type symbol O, for any of its perceptions $P \in \text{PERC}(O)$, to a set $O^{\mathcal{B}(P)} \subseteq \Delta^{\mathcal{B}}_{\mathcal{O}}$.
- Every local relationship type symbol R, for any of its perceptions $P \in \text{PERC}(R)$, to a set $R^{\mathcal{B}(P)}$ of couples $\langle r, u \rangle$ where $r \in \Delta^{\mathcal{B}}_{\mathcal{R}_l}$ and u is a $\mathcal{U}$-labeled tuple over $\Delta^{\mathcal{B}}_{\mathcal{O}}$ such that if $\text{REL}_l(R, P) = \langle U_1 : O_1, \ldots, U_k : O_k \rangle$, then: $\langle r, u \rangle \in R^{\mathcal{B}(P)} \wedge u = \langle U_1 : o_1, \ldots, U_k : o_k \rangle \Rightarrow \forall i \in \{1, \ldots, k\} \ (o_i \in O_i^{\mathcal{B}(P)})$. Further, $R^{\mathcal{B}(P)}$ is such that:
  $\forall \langle r_1, u_1 \rangle, \langle r_2, u_2 \rangle \in R^{\mathcal{B}(P)} \ (r_1 = r_2 \Rightarrow u_1 = u_2)$.
- Every interperception relationship type symbol R, for the P_{ip} perception, to a set $R^{\mathcal{B}(P)}$ of couples $\langle r, u \rangle$ where $r \in \Delta^{\mathcal{B}}_{\mathcal{R}_{ip}}$ and u is a $\mathcal{U} \times \mathcal{P}_s$-labeled tuple over $\Delta^{\mathcal{B}}_{\mathcal{O}}$ such that if $\text{REL}_{ip}(R) = \langle (U_1, P_1) : O_1, \ldots, (U_k, P_k) : O_k \rangle$, then: $\langle r, u \rangle \in R^{\mathcal{B}(P)} \wedge u = \langle (U_1, P_1) : o_1, \ldots, (U_k, P_k) : o_k \rangle \Rightarrow \forall i \in \{1, \ldots, k\} \ (o_i \in O_i^{\mathcal{B}(P_i)})$. Further, $R^{\mathcal{B}(P)}$ is such that:
  $\forall \langle r_1, u_1 \rangle, \langle r_2, u_2 \rangle \in R^{\mathcal{B}(P)} \ (r_1 = r_2 \Rightarrow u_1 = u_2)$.
- Every attribute symbol A, for a perception $P \in \mathcal{P}$, to a set $A^{\mathcal{B}(P)} \subseteq (\Delta^{\mathcal{B}}_{\mathcal{O}} \cup \Delta^{\mathcal{B}}_{\mathcal{R}}) \times \Delta^{\mathcal{B}}_{D}$, such that, for each object or relationship type $X \in (\mathcal{O} \cup \mathcal{R})$ and perception $P \in \mathcal{P}$, if $\text{ATT}(X, P)[A] = D_i$, then: $x \in X^{\mathcal{B}(P)} \Rightarrow (\exists a_i \in D_i^{\mathcal{B}} \ (\langle x, a_i \rangle \in A^{\mathcal{B}(P)}) \wedge \forall a_i \in D_i^{\mathcal{B}} \ (\langle x, a_i \rangle \in A^{\mathcal{B}(P)} \Rightarrow a_i \in \Delta^{\mathcal{B}}_{D_i}))$. □

Definition 6. (Consistent database state of a multiperception Simple-Mads schema)
A database state $\mathcal{B}$ is said to be consistent if it satisfies all of the constraints expressed in the schema:

- *Local population inclusion*:
  $\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R}_l) \ \forall P \in \mathcal{P}_s \ (\text{ISA}_l(X_1, X_2, P) \Rightarrow X_1^{\mathcal{B}(P)} \subseteq X_2^{\mathcal{B}(P)})$.
- *Interperception population inclusion*:
  $\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R}_l) \ \forall P_1, P_2 \in \mathcal{P}_s \ (\text{ISA}_{ip}(X_1, P_1, X_2, P_2) \Rightarrow X_1^{\mathcal{B}(P_1)} \subseteq X_2^{\mathcal{B}(P_2)})$.
- *Local population intersection*:
  $\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R}) \forall P \in \mathcal{P}_s \ (X_1^{\mathcal{B}(P)} \cap X_2^{\mathcal{B}(P)} \neq \emptyset \Rightarrow X_1 = X_2 \vee$
  $\text{ISA}_l^+ \ (X_1, X_2, P) \vee \text{ISA}_l^+(X_2, X_1, P) \vee \text{OVLP}_l(X_1, X_2, P))$.
- *Interperception population intersection*:
  $\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R}) \ \forall P_1, P_2 \in \mathcal{P}_s \ (X_1^{\mathcal{B}(P_1)} \cap X_2^{\mathcal{B}(P_2)} \neq \emptyset \wedge P_1 \neq P_2 \Rightarrow X_1 = X_2 \vee$
  $\text{ISA}_{ip}^+(X_1, P_1, X_2, P_2) \vee \text{ISA}_{ip}^+(X_2, P_2, X_1, P_1) \vee \text{OVLP}_{ip}(X_1, P_1, X_2, P_2))$.
- *Local cardinality constraints*:
  For each cardinality constraint $\text{CARD}_l(O, R, U, P)$ of a local relationship $R \in \mathcal{R}_l$ and a perception $P \in \text{PERC}(R)$: $\forall o \in O^{\mathcal{B}(P)} \ (\text{CMIN}_l(O, R, U, P) \leq \#\{\langle r, u \rangle \in R^{\mathcal{B}(P)} \mid u[U] = o\} \leq \text{CMAX}_l(O, R, U, P))$.
- *Interperception cardinality constraints*:
  For each cardinality constraint $\text{CARD}_{ip}(O, R, U, P)$ of an interperception relationship $R \in \mathcal{R}_{ip}$ and a perception $P \in \text{PERC}(O)$: $\forall o \in O^{\mathcal{B}(P)}$
  $(\text{CMIN}_{ip}(O, R, U, P) \leq \#\{\langle r, u \rangle \in R^{\mathcal{B}} \mid u[(U, P)] = o\} \leq \text{CMAX}_{ip}(O, R, U, P))$.

- *Key constraints*:
  For each key constraint $\text{KEY}(X, P, K)$ of an object or relationship type $X \in (\mathcal{O} \cup \mathcal{R})$ in a perception $P \in \mathcal{P}$, where $K = \{A_1, \ldots, A_n\}$: $x_1, x_2 \in X^{\mathcal{B}(P)} \Rightarrow (\forall i \in \{1, \ldots, n\}\ (\langle x_1, a_i^1 \rangle \in A_i^{\mathcal{B}(P)} \wedge \langle x_2, a_i^2 \rangle \in A_i^{\mathcal{B}(P)} \wedge a_i^1 = a_i^2) \Rightarrow x_1 = x_2)$.
- *Attributes common to several perceptions*:
  For each $X \in (\mathcal{O} \cup \mathcal{R}_l)$, for each $P_1, P_2 \in \text{PERC}(X)$, for each $A \in \mathcal{A}$, for each $D \in \mathcal{D}$ such that $\text{ATT}(X, P_1)[A] = \text{ATT}(X, P_2)[A] = D$: $\forall x \in (X^{\mathcal{B}(P_1)} \cap X^{\mathcal{B}(P_2)})\ \forall \langle x, a_1 \rangle \in A^{\mathcal{B}(P_1)}\ \forall \langle x, a_2 \rangle \in A^{\mathcal{B}(P_2)}\ (a_1 = a_2)$.
- *Roles common to several perceptions*:
  For each $R \in \mathcal{R}_l$, for each $P_1, P_2 \in \text{PERC}(R)$, for each $U \in \mathcal{U}$ such that $\text{REL}_l(R, P_1)[U] = \text{REL}_l(R, P_2)[U]$:
  $\forall \langle r_1, u_1 \rangle \in R^{\mathcal{B}(P_1)}\ \forall \langle r_2, u_2 \rangle \in R^{\mathcal{B}(P_2)}\ (r_1 = r_2 \Rightarrow u_1[U] = u_2[U])$. □

5.6 Dependencies Between Perceptions

In databases, the existence of some elements may depend upon other ones (let us call these other elements the reference elements): As databases follow the closed-world assumption, existence-dependent elements cannot be created if these reference elements do not exist, and conversely the deletion of a reference element has to be propagated to its dependent elements or prevented as long as it has dependants. This is the case of all relationship instances: A relationship instance cannot exist without the objects that it links. The other elements that are existence dependent are: object types linked to a relationship type (the reference element) through a mandatory role, and object and relationship types that have one or several super-types (the reference elements). Classic (i.e., without perception) database systems, which assume the closed-world assumption, enforce these existence constraints. When dealing with a multiple perceptions and representations database, if the dependent element, say DE, belongs to a perception, say P_1, while the reference element, say RE, belongs to another one, say P_2, then insertions of instances of DE cannot be local operations in P_1, and deletions of instances of RE cannot be local operations in P_2. We say that the perceptions P_1 and P_2 are mutually dependent. Insertions of instances of the dependent element and deletions of the reference element require using the composite perception $(\mathsf{P}_1 + \mathsf{P}_2)$.

For example in Fig. 5.7, the perceptions P_m and P_g are mutually dependent because the cardinalities of the relationship type LocatedIn (shown in Fig. 5.9) say that each Vineyard object must be linked to at least one GeologicalUnit object. This implies that, when creating a Vineyard object in P_m, it should be linked straight away to a GeologicalUnit object of P_g, thus requiring the composite perception $(\mathsf{P}_\mathsf{m} + \mathsf{P}_\mathsf{g})$. On the other hand, a GeologicalUnit object of P_g can be deleted only if it is no longer linked to Vineyard objects of P_m, or the deletion should be propagated to the Vineyard objects, which requires the $(\mathsf{P}_\mathsf{m} + \mathsf{P}_\mathsf{g})$ perception. The perceptions P_m and P_e are also mutually dependent for the creation of instances of WhiteWine, RedWine, and RoseWine and for the deletion of instances of Wine.

An interperception overlap link also creates a dependency between the perceptions, because – as we have seen in previously – adding a new instance to an already-existing object requires knowing its oid. For example, let us assume a variant of Fig. 5.7 where WhiteWine, RedWine, and RoseWine of perception P_e describe some wines

of perception $\mathsf{P_m}$ but also some other wines not recorded in $\mathsf{P_m}$. Let us assume that the designer expresses this knowledge by three interperception overlap links:

Overlap ($\mathsf{P_m}$.Wine, $\mathsf{P_e}$.WhiteWine)
Overlap ($\mathsf{P_m}$.Wine, $\mathsf{P_e}$.RedWine)
Overlap ($\mathsf{P_m}$.Wine, $\mathsf{P_e}$.RoseWine)

Then, an insertion of a wine in either perception, $\mathsf{P_m}$ or $\mathsf{P_e}$, requires to access the other perception in order to know if the wine already exists and then get its oid. The two perceptions are mutually dependent for insertions. Yet deletions are local operations.

In conclusion, any interperception link, be it a relationship type, an is-a or an overlap link, causes a dependency between the two perceptions for the insertion or deletion of the linked elements.

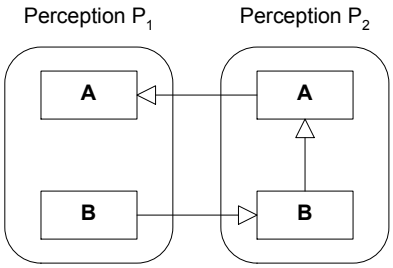

Fig. 5.8. Is-a propagation from one perception to another one.

Another kind of dependency between perceptions is the propagation of reasoning from one perception to another one. The reasoning that takes place in Mads schemas without perceptions (inferred overlap links from is-a links) is extended to multiperception Mads schemas. It allows to infer a local is-a link from two interperception is-a links as shown in Fig.5.8, exactly like in distributed description logics where generalized subsumptions are propagated between modules through bridge rules (see Chapter 12 in this book). In the same way, local and interperception overlap links are inferred from interperception is-a links.

Proposition 2. (Logical implication for a SimpleMads multiperception schema)
As a consequence of the definitions of SimpleMads multiperception schema and consistent database state, the following rules can be derived:

- *Inferred local overlap links from local is-a links*:
  $\forall X_1, X_2, X_3 \in (\mathcal{O} \cup \mathcal{R})\ (\text{ISA}(X_1, X_2) \wedge \text{ISA}(X_1, X_3) \Rightarrow \text{OVLP}(X_1, X_3))$.
- *Inferred local is-a links from interperception is-a links*:
  $\forall X_1, X_2, X_3 \in (\mathcal{O} \cup \mathcal{R}_l)\ \forall P_1, P_2 \in \mathcal{P}_s$
  $(\text{ISA}^+_{ip}(X_1, P_1, X_2, P_2) \wedge \text{ISA}^+_{ip}(X_2, P_2, X_3, P_1) \Rightarrow \text{ISA}_l(X_1, X_2, P_1))$.
- *Inferred local overlap links from interperception is-a links*:
  $\forall X_1, X_2, X_3 \in (\mathcal{O} \cup \mathcal{R}_l)\ \forall P_1, P_2 \in \mathcal{P}_s$
  $(\text{ISA}^+_{ip}(X_1, P_1, X_2, P_2) \wedge \text{ISA}^+_{ip}(X_1, P_1, X_3, P_2) \Rightarrow \text{OVLP}_l(X_1, X_3, P_2))$.

- *Inferred interperception overlap links from interperception is-a links*:
  $\forall X_1, X_2, X_3 \in (\mathcal{O} \cup \mathcal{R}_l)\ \forall P_1, P_2, P_3 \in \mathcal{P}_s$
  $(\text{ISA}^+_{ip}(X_1,P_1, X_2, P_2) \wedge \text{ISA}^+_{ip}(X_1, P_1, X_3, P_3) \Rightarrow \text{OVLP}_{ip}(X_2, P_2, X_3, P_3))$. □

5.7 Using Perceptions

As already stated, user interaction with a multiperception database starts with the specification of which perception the user wants to work with. The following Mads command provides this functionality:

openDatabase(dbName, myView)

where dbName is the name of a multiperception database and myView denotes either a simple perception $\mathsf{p_i}$ or a composite perception $(\mathsf{p_1} + \mathsf{p_2} + \ldots + \mathsf{p_k})$.

Upon receiving this command, the system creates a new virtual database (its schema and instantiation) out of the multiperception database dbName. This new virtual database is the "view" provided by the perception myView to the user. If myView contains a composite perception the virtual database will be a true multiperception database, otherwise it will be a monoperception database, equivalent to a classic database without perception. Hereinafter, we formally define the schema and semantics of a perception, be it a simple perception or a composite one.

Definition 7. (Schema and semantics of a perception)
Let $\Sigma = (\mathcal{L}, \text{PERC}, \text{REL}_l, \text{REL}_{ip}, \text{ATT}, \text{CARD}_l, \text{CARD}_{ip}, \text{ISA}_l, \text{ISA}_{ip}, \text{OVLP}_l, \text{OVLP}_{ip}, \text{KEY})$ be a SimpleMads multiperception schema, where $\mathcal{L} = \mathcal{P} \cup \mathcal{O} \cup \mathcal{R} \cup \mathcal{A} \cup \mathcal{U} \cup \mathcal{D}$, $\mathcal{P} = \{P_{ip}\} \cup \mathcal{P}_s$, $\mathcal{R} = \mathcal{R}_l \cup \mathcal{R}_{ip}$. Let $\mathcal{B} = (\Delta^{\mathcal{B}}_{\mathcal{O}} \cup \Delta^{\mathcal{B}}_{\mathcal{R}} \cup \Delta^{\mathcal{B}}_{D}, \cdot^{\mathcal{B}(P)})$ be a consistent database state for Σ. Let $\mathcal{P}'_s$ be a nonempty set of perception symbols, $\mathcal{P}'_s \subset \mathcal{P}_s$. The perception $\mathcal{P}'_s$ of the multiperception database $(\Sigma, \mathcal{B})$ is a SimpleMads multiperception database, whose schema Σ' and database state $\mathcal{B}'$ are defined as follows: $\Sigma' = (\mathcal{L}', \text{PERC}', \text{REL}'_l, \text{REL}'_{ip}, \text{ATT}', \text{CARD}'_l, \text{CARD}'_{ip}, \text{ISA}'_l, \text{ISA}'_{ip}, \text{OVLP}'_l, \text{OVLP}'_{ip}, \text{KEY}')$, is defined by:

- $\mathcal{L}' \subset \mathcal{L}$ is the finite alphabet partitioned into the sets $\mathcal{P}'$, $\mathcal{O}'$, $\mathcal{R}'$, $\mathcal{A}'$, $\mathcal{U}'$, and $\mathcal{D}'$ defined by:
  $\mathcal{P}' = \{P_{ip}\} \cup \mathcal{P}'_s$,
  $\mathcal{O}' = \{O \mid O \in \mathcal{O} \wedge \text{PERC}(O) \cap \mathcal{P}'_s \neq \emptyset\}$,
  $\mathcal{R}' = \mathcal{R}'_l \cup \mathcal{R}'_{ip}$,
  $\mathcal{R}'_l = \{R \mid R \in \mathcal{R}_l \wedge \text{PERC}(R) \cap \mathcal{P}'_s \neq \emptyset\}$,
  $\mathcal{R}'_{ip} = \{R \mid R \in \mathcal{R}_{ip} \wedge \text{REL}_{ip}(R) = \langle (U_1, P_1) : O_1, \ldots, (U_k, P_k) : O_k \rangle \wedge$
  $\forall i \in \{1, \ldots, k\}\ (P_i \in \mathcal{P}'_s)\}$,
  $\mathcal{A}' = \{A \mid A \in \mathcal{A} \wedge \exists X \in \mathcal{O}' \cup \mathcal{R}'\ \exists P \in \mathcal{P}'\ \exists D \in \mathcal{D}\ \text{ATT}(X, P)[A] = D\}$,
  $\mathcal{U}' = \{U \mid U \in \mathcal{U} \wedge \exists R \in \mathcal{R}'_l\ \exists P \in \mathcal{P}'_s\ \exists O \in \mathcal{O}'\ (\text{REL}_l(R, P)[U] = O)\} \cup$
  $\{U \mid U \in \mathcal{U} \wedge \exists R \in \mathcal{R}'_{ip}\ \exists P \in \mathcal{P}'_s\ \exists O \in \mathcal{O}'\ (\text{REL}_{ip}(R)[(U, P)] = O)\}$,
  $\mathcal{D}' = \{D \mid D \in \mathcal{D} \wedge \exists X \in \mathcal{O}' \cup \mathcal{R}'\ \exists P \in \mathcal{P}'\ \exists A \in \mathcal{A}'\ (\text{ATT}(X, P)[A] = D)\}$.
- PERC' is the total function that maps each object or relationship type symbol $X \in \mathcal{O}' \cup \mathcal{R}'$ to a nonempty set of perceptions defined by:
  $\forall X \in \mathcal{O}' \cup \mathcal{R}'\ (\text{PERC}'(X) = \text{PERC}(X) \cap \mathcal{P}')$.

- REL'_l is the total function that maps a couple made up of a local relationship type symbol R in $\mathcal{R}'_l$ and a perception symbol $P \in \text{PERC}'(R)$ to an $\mathcal{U}'$-labeled tuple over $\mathcal{O}'$ defined by:
  $\forall R \in \mathcal{R}'_l \; \forall P \in \text{PERC}'(R) \; (\text{REL}'_l(R,P) = \text{REL}_l(R,P))$.
- REL'_{ip} is the total function that maps an interperception relationship type symbol R in $\mathcal{R}'_{ip}$ to an $\mathcal{U}' \times \mathcal{P}'_s$-labeled tuple over $\mathcal{O}'$ defined by:
  $\forall R \in \mathcal{R}'_l \; (\text{REL}'_{ip}(R) = \text{REL}_{ip}(R))$.
- ATT' is the partial function that maps a couple made up of an object or relationship type symbol X in $\mathcal{O}' \cup \mathcal{R}'$ and a perception symbol $P \in \text{PERC}'(X)$ to an $\mathcal{A}'$-labeled tuple over $\mathcal{D}'$ defined by:
  $\forall X \in \mathcal{O}' \cup \mathcal{R}' \; \forall P \in \text{PERC}'(X) \; (\text{ATT}'(X,P) = \text{ATT}(X,P))$.
- CARD'_l is the partial function $\mathcal{O}' \times \mathcal{R}'_l \times \mathcal{U}' \times \mathcal{P}'_s \to \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ defined by:
  $\forall O \in \mathcal{O}' \; \forall R \in \mathcal{R}'_l \; \forall U \in \mathcal{U}' \; \forall P \in \text{PERC}'(R) \; (\text{REL}'_l(R,P)[U] = O \Rightarrow$
  $(\text{CARD}'_l(O,U,R,P) = \text{CARD}_l(O,U,R,P)))$.
- CARD'_{ip} is the partial function $\mathcal{O}' \times \mathcal{R}'_{ip} \times \mathcal{U}' \times \mathcal{P}'_s \to \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ defined by:
  $\forall O \in \mathcal{O}' \; \forall R \in \mathcal{R}'_{ip} \; \forall U \in \mathcal{U}' \; \forall P \in \mathcal{P}'_s \; (\text{REL}'_{ip}(R)[U,P] = O \Rightarrow$
  $(\text{CARD}'_{ip}(O,U,R,P) = \text{CARD}_{ip}(O,U,R,P)))$.
- ISA'_l is the ternary relation $\text{ISA}'_l \subseteq (\mathcal{O}' \times \mathcal{O}' \times \mathcal{P}'_s) \cup (\mathcal{R}'_l \times \mathcal{R}'_l \times \mathcal{P}'_s)$ defined by:
  $\forall X_1, X_2 \in \mathcal{O}' \cup \mathcal{R}'_l \; \forall P \in \mathcal{P}'_s \; (\text{ISA}'_l(X_1,X_2,P) \Leftrightarrow \text{ISA}_l(X_1,X_2,P))$.
- ISA'_{ip} is the quaternary relation $\text{ISA}'_{ip} \subseteq (\mathcal{O}' \times \mathcal{P}'_s \times \mathcal{O}' \times \mathcal{P}'_s) \cup (\mathcal{R}'_l \times \mathcal{P}'_s \times \mathcal{R}'_l \times \mathcal{P}'_s)$ defined by:
  $\forall X_1, X_2 \in \mathcal{O}' \cup \mathcal{R}'_l \; \forall P_1, P_2 \in \mathcal{P}'_s$
  $(\text{ISA}'_{ip}(X_1,P_1,X_2,P_2) \Leftrightarrow \text{ISA}_{ip}(X_1,P_1,X_2,P_2))$.
- OVLP'_l is the ternary relation $\text{OVLP}'_l \subseteq (\mathcal{O}' \times \mathcal{O}' \times \mathcal{P}'_s) \cup (\mathcal{R}'_l \times \mathcal{R}'_l \times \mathcal{P}'_s)$ defined by:
  $\forall X_1, X_2 \in \mathcal{O}' \cup \mathcal{R}'_l \; \forall P \in \mathcal{P}'_s \; (\text{OVLP}'_l(X_1,X_2,P) \Leftrightarrow \text{OVLP}_l(X_1,X_2,P))$.
- OVLP'_{ip} is the quaternary relation $\text{OVLP}'_{ip} \subseteq (\mathcal{O}' \times \mathcal{P}'_s \times \mathcal{O}' \times \mathcal{P}'_s) \cup (\mathcal{R}'_l \times \mathcal{P}'_s \times \mathcal{R}'_l \times \mathcal{P}'_s)$ defined by:
  $\forall X_1, X_2 \in \mathcal{O}' \cup \mathcal{R}'_l \; \forall P \in \mathcal{P}'_s \; (\text{OVLP}'_{ip}(X_1,P_1,X_2,P_2) \Leftrightarrow \text{OVLP}_{ip}(X_1,P_1,X_2,P_2))$.
- KEY' is the ternary relation $\text{KEY}' \subseteq (\mathcal{O}' \cup \mathcal{R}') \times \mathcal{P}' \times 2^{\mathcal{A}'}$ defined by:
  $\forall X \in \mathcal{O}' \cup \mathcal{R}' \; \forall P \in \mathcal{P}' \; \forall K \in 2^{\mathcal{A}'} \; (\text{KEY}'(X,P,K) \Leftrightarrow \text{KEY}(X,P,K))$.

The state $\mathcal{B}'$ of the perception is the tuple $\mathcal{B}' = (\Delta^{\mathcal{B}'}_{\mathcal{O}'} \cup \Delta^{\mathcal{B}'}_{\mathcal{R}'} \cup \Delta^{\mathcal{B}'}_{D'}, \cdot^{\mathcal{B}'(P)})$ defined by:

- The function $\cdot^{\mathcal{B}'(P)}$, that, for some $P \in \mathcal{P}'$, maps:
  - Every domain symbol $D_i \in \mathcal{D}'$, for any perception $P \in \mathcal{P}'$, to the set $D_i^{\mathcal{B}'(P)} = \Delta^{\mathcal{B}}_{D_i}$;
  - Every object type symbol $O \in \mathcal{O}'$, for any of its perceptions $P \in \text{PERC}'(O)$, to the set $O^{\mathcal{B}'(P)} = O^{\mathcal{B}(P)}$;
  - Every relationship type symbol $R \in \mathcal{R}'$, for any of its perceptions $P \in \text{PERC}'(R)$, to the set $R^{\mathcal{B}'(P)} = R^{\mathcal{B}(P)}$);
  - Every attribute symbol $A \in \mathcal{A}'$, for a perception $P \in \mathcal{P}'$, to the set
    $A^{\mathcal{B}'(P)} = \{\langle x,a \rangle \mid \langle x,a \rangle \in A^{\mathcal{B}(P)} \wedge \exists X \in \mathcal{O}' \cup \mathcal{R}' \; \exists D \in \mathcal{D}'$
    $(P \in \text{PERC}'(X) \wedge \text{ATT}'(X,P)[A] = D \wedge x \in \text{PERC}'(X,P)\})$.

- $\Delta^{\mathcal{B}'}_{\mathcal{O}'} = \bigcup_{O \in \mathcal{O}' P \in \text{PERC}'(O)} O^{\mathcal{B}'(P)}$,
- $\Delta^{\mathcal{B}'}_{\mathcal{R}'} = \bigcup_{R \in \mathcal{R}' P \in \text{PERC}'(R)} R^{\mathcal{B}'(P)}$,
- $\Delta^{\mathcal{B}'}_{D'} = \bigcup_{D_i \in \mathcal{D}'} D_i^{\mathcal{B}'(P)}$. □

Proposition 3. If the set of perception symbols $\mathcal{P}'_s$ contains only one perception, the multiperception database $(\Sigma', \mathcal{B}')$ reduces to a SimpleMads database without perceptions. Indeed, the set $\mathcal{R}'_{ip}$ is empty, as well as the relations ISA'_{ip}, OVLP'_{ip}, REL'_{ip}, and CARD'_{ip}. □

Theorem 1. The state $\mathcal{B}'$ of a perception is consistent, i.e., it satisfies all the constraints of the schema Σ'. □

When the system receives an OpenDatabase(dbName, myView) command, it performs the following process: it matches the perceptions in myView with the set of perceptions of each object and relationship type of the database in order to determine which object and relationship types (with which properties and which populations) belong to the perception myView. Any element that belongs to at least one of the perceptions in myView belongs to the (composite) perception myView. Obviously, the myView representation of an object type includes all the representations of the attributes that belong to at least one of the perceptions in myView. If myView is a composite perception, this process may select several representations for the same attribute. Local relationship types follow the same selection process. However, whenever myView is a composite perception, the system has to perform an additional selection step to complete the definition of the myView perception: it has to look for interperception relationship types eligible for the given composite perception. The eligible interperception relationship types are those where all roles and linked object types belong to myView. The myView representation of the relationship type is made up of its selected roles and all the representations of all its attributes and semantics that belong to at least one of the perceptions in myView.

Let us refer to the database of Fig. 5.3 for an example of accessing an interperception relationship type. In order to know on which kind of soil – an attribute of GeologicalUnit – a specific vineyard is located, the user query has to go through the relationship type LocatedIn which does not belong to a simple perception. Thus, the user must open the database with the composite perception $(\mathsf{P_m} + \mathsf{P_g})$ for querying the LocatedIn relationship type.

When querying the database with a composite perception, users get for each query a multiperception answer, i.e., a set of answers, one per perception. The component answers are linked together by the fact that all representations describing the same object (respectively, relationship) instance are identified by the same system-defined identifier, oid (respectively, rid). For example, let us refer to the database of Fig. 5.6 and assume a user with the composite perception $(\mathsf{P_1} + \mathsf{P_2})$ who asks the following query: "Give me all animals". The answer will be:

$\mathsf{P_1} : \mathsf{a_1}, \mathsf{a_2}, \mathsf{a_3}, \mathsf{h_1}, \mathsf{h_2}$
$\mathsf{P_2} : \mathsf{a_1}, \mathsf{a_2}, \mathsf{a_3}$

Loading and updating data in a multiperception database may be done collaboratively by several users with different perceptions. An object (or relationship) instance that has several representations, say for perceptions $\mathsf{p_1}, \mathsf{p_2}, \ldots$, and $\mathsf{p_k}$, may either be inserted (or deleted) in two ways:

- A user with the composite perception $(\mathsf{p_1} + \mathsf{p_2} + \ldots + \mathsf{p_k})$ may insert (or delete) the whole instance with all its representations in a single operation; or
- The insertion (or deletion) is done by a sequence of operations: For each simple perception of the set $\{\mathsf{p_1}, \mathsf{p_2}, \ldots, \mathsf{p_k}\}$, a user with this perception inserts

(or deletes) the corresponding representation. When processing the first insert operation, the DBMS creates a new instance with a new oid and a unique representation. Each following insert operation adds a representation to the existing instance (there is no oid creation).

For example, adding in the Wine object type of Fig. 5.4 a new wine instance, say Clos Vougeot 2004, with its two representations may be done by a user with the composite perception ($P_m + P_e$) by giving the data for both representations as follows:

```
p = insertObject(Wine, {Pm, Pe}(
/* attributes of perception Pm and Pe */
name = 'Clos Vougeot', year = 2004, ...
/* attributes of perception Pm */
description.atPerception(Pm) =
    'Sourced from old vines, the soft finish with silky tannins ...',
degree.atPerception(Pm) = 12.25, ...
/* attributes of perceptions Pe */
description.atPerception(Pe) =
    'Full of dark berry fruits on the nose, the palate depth shows ...', ...
degree.atPerception(Pe) = 12, ... ))
```

Alternatively, it can be done in two steps, e.g., a user of perception P_m inserting the P_m representation as in:

```
p = insertObject(Wine, {Pm}(
/* attributes of perception Pm */
name = 'Clos Vougeot', year = 2004, ...
description = 'Sourced from old vines, the soft finish with silky tannins ...',
degree = 12.25, ... ) )
```

and later a user of perception P_e inserting the P_e representation for the same Clos Vougeot 2004 instance as in:

```
p = select [name = 'Clos Vougeot' ∧ year = 2004 ] Wine ;
addObjectRepresentation(Wine, p, {Pe}(
/* attributes of perception Pe */
name = 'Clos Vougeot', year = 2004, ...
description = 'Full of dark berry fruits on the nose, the palate depth shows ...',
...
degree = 12, ... ))
```

As can be seen, users willing to separately (i.e., perception per perception) create different representations for the same instance of a multiperception type have to agree on using the same key. This key must have a unique representation common to all perceptions. In the Wine object type, the common key is made up of two attributes, name and year. In the above example, the user must first obtain the identifier of the Clos Vougeot 2004 instance with a select operation in order to be able to add a representation to that instance.

The existence of a relationship instance depends upon the one of the objects it links. In Entity-Relationship data models pending roles of relationships are prohibited. Thus, inserting or accessing a relationship instance, requires having access to

the relationship type and to the linked object instances. Hence, inserting and accessing an instance of an interperception relationship type is only possible through a composite perception that contains all the perceptions of the linked object types. On the other hand, local relationship types that have several representations may, like object types, be inserted or deleted either by a unique operation on a composite perception that covers all the representations, or by a sequence of operations on simple perceptions. For example, the relationship type ProducedBy of Fig. 5.5 is local, i.e., any two instances, one of Wine and one of Vineyard, linked by a ProducedBy instance belong to the same perception as the ProducedBy instance. Let us assume that the two representations of ProducedBy differ by having different sets of attributes, e.g., number of bottles produced for P_m and vintage description for P_e. Then inserting a new instance of ProducedBy linking the Wine Clos Vougeot 2004 with oid p to the Vineyard Vigne du Clos Vougeot with oid q can be done either by one insert operation with perception $(P_m + P_e)$ as in:

insertRelationship(ProducedBy, {P_m, P_e}, Wine: p, Vineyard: q) (
/* attributes of P_m */
numberOfBottles = 3500, ...
/* attributes of P_e */
description = 'The 2004 vintage had moderate rainfall in the winter ...', ...)

or by two insert operations, one with perception P_m and one with perception P_e.

5.8 Mapping into the Relational Model

For a database design based on Mads to be operational, we have defined an implementation approach that automatically transforms a Mads schema into an equivalent logical schema in the relational or object-relational data model, which can then be loaded into a commercial DBMS. The approach was materialized as a CASE tool, whose detailed description can be found in [PSZ06a, PSZ06b]. This section discusses the general principles of that translation using the example schema in Fig. 5.9, which is an enriched version of the schema in Fig. 5.3 with all the attributes and perceptions shown. Our main intention is to show how multirepresentation features are conveyed into the logical schema.

Logical models target easiness and efficiency of implementation. They consequently support less sophisticated and poorer data structures than those of conceptual models. Therefore, when translating a conceptual schema into a logical schema, the critical issue is to avoid or at least limit the semantic loss due to the poorer expression power of logical data models. Usually, high-level features of conceptual models are translated into a combination of logical-level features, the combination aiming at filling the gap between the conceptual and logical constructs and minimize the semantic loss.

Let us illustrate this using the example schema from Fig. 5.9. Remark that Fig. 5.9 uses the same visual presentation as Fig. 5.4: all perceptions are merged, while Fig. 5.3 uses a visual presentation that separates the perceptions. Still the semantics conveyed by these two visual presentations is the same. For the translation into the relational model, basically there are two ways that are quite similar to the two visual presentations. These two ways generate relational schemas that are different but convey the same semantics. The difference between these two ways of translating is

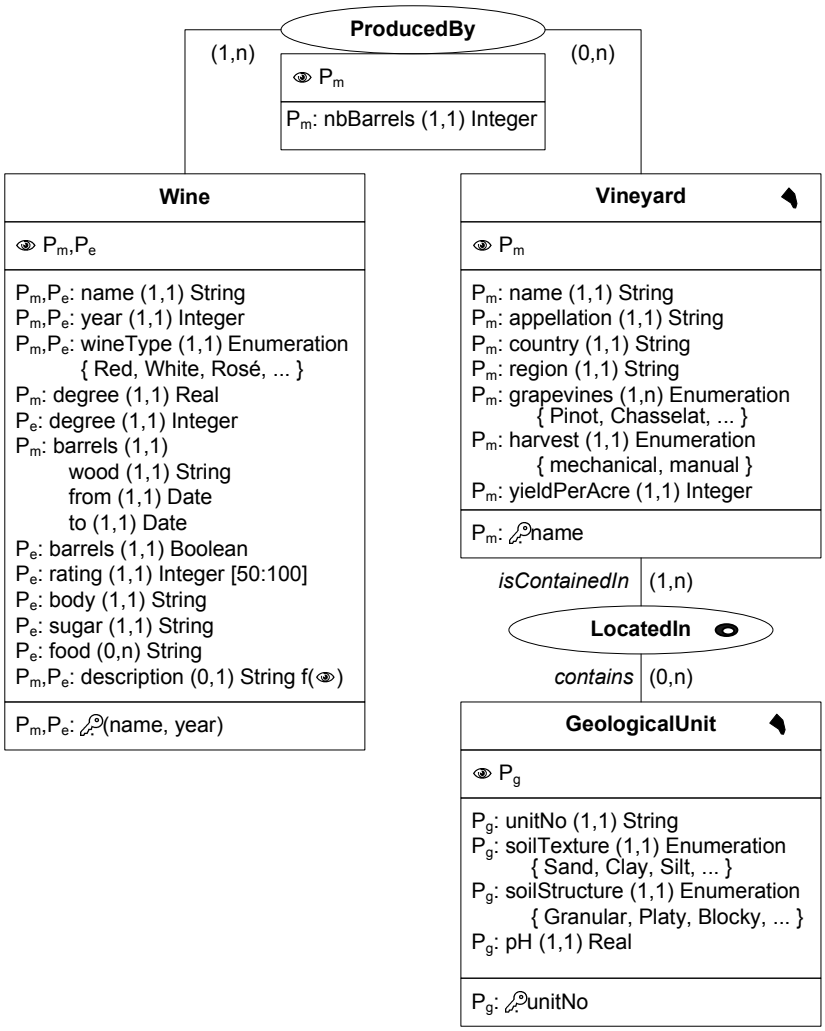

Fig. 5.9. A detailed version of the schema of Fig. 5.3.

whether to create for each multi-perception object (and relationship) type a unique relational table containing all the attributes from the various perceptions, or several tables, one table for each perception that contains only the attributes defined for this perception. The first solution boils down to translating the multiperception schemas as presented with all perceptions merged, while the second solution to translating the multiperception schemas as presented with each perception on its own. The first solution generates tuples with NULL values each time that an object does not belong to all the perceptions defined for its object type. Here, we present the second solution, one table per object type and per perception. The translation algorithm consists in 1) applying to each perception the classic translation algorithm from

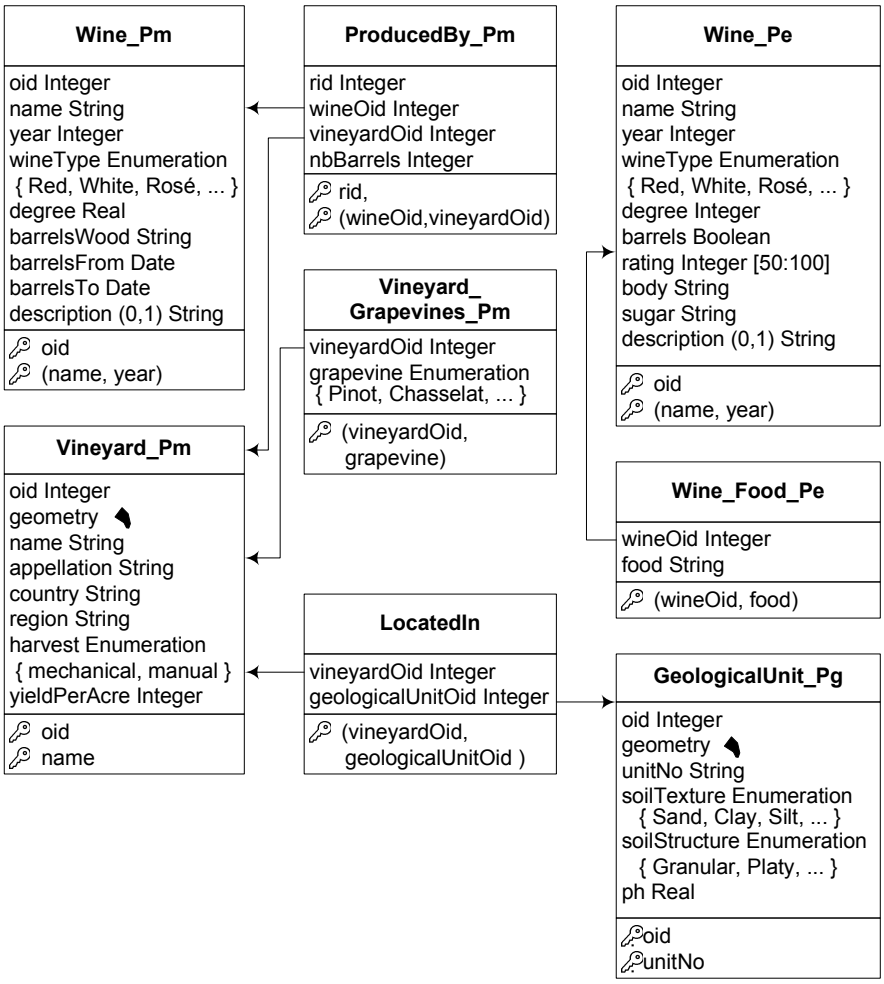

Fig. 5.10. Relational implementation of the schema of Fig. 5.9.

the Entity Relationship model without perception to the relational model, and 2) implementing each interperception relationship type by a relational table.

The result of the translation of the schema of Fig. 5.9 into a relational schema is shown Fig. 5.10. The first rule we used is: For each perception and for each of its object type, generate one primary table per perception[7]. In the example, as Wine belongs to two perceptions, its translation generates the two relational tables Wine_Pm and Wine_Pe, each one holding the monovalued attributes of perceptions P_m and P_e, respectively. The second rule states that composite attributes, such as barrels in P_m are replaced by their component attributes. This rule leads to a semantic loss (the composite attribute itself is lost), but the loss is in the label, the

[7] The primary table is the one holding all monovalued attributes of the object type.

attribute values are preserved. The third rule is the traditional one that translates multivalued attributes by generating an additional table. In our running example, for perception P_e, the translation of the multivalued attribute food generates the table Wine_Food_Pe and the translation of the multivalued attribute grapevines for perception P_m generates the table Vineyard_Grapevines_Pm.

The relational representation of Wine does not make any difference between the attributes that are identical to both perceptions, such as name, year, and wineType, and the attributes whose value is perception dependent, such as description. Yet, in the conceptual specification, the values of the former attributes is shared by the two perceptions (i.e., the value is always the same in the two perceptions), while the values of the latter, description, are independent one from the other in the two perceptions. To prevent this semantic loss, the translation generates triggers (to be loaded into the target DBMS) to ensure that when users update an instance of, e.g., Wine_Pm, the updated values of name, year, and wineType (but not description) are propagated to the corresponding instance of Wine_Pe. Translation of local relationship types follows the same rule: one primary relational table per perception and per relationship type.

Like in traditional databases, roles of relationship types are translated into external keys. Lastly, each interperception relationship type, e.g LocatedIn, is translated into a relational table, exactly like for any classic Entity Relationship model.

Identifiers (oids and rids) simplify the translation of is-a and overlapping links (whether local or interperception). Consider again Fig. 5.7 where there is an interperception link between Wine in perception P_e and the three disjoint object types RedWine, WhiteWine, and RoseWine in perception P_e. In this case, the relational representation will include the tables Wine_Pm, RedWine_Pe, WhiteWine_Pe, and RoseWine_Pe, all of them with an attribute oid. The is-a relationship will be implemented by referential integrity contraints between each of the three tables in perception P_e and the table in perception P_m.

The identifiers help also to link several representations of the same instance. For example, if in Fig. 5.10 a wine has both representations P_m and P_e, the same oid value will be found in tables Wine_Pm and Wine_Pe. For example the following query retrieves the P_m representation of the wine "Zifandel Clos Marie" 2000:

```
SELECT * FROM Wine_Pm
WHERE name="Zifandel Clos Marie" AND year=2000
```

Similarly, the following query retrieves all representations of the same wine:

```
SELECT * FROM Wine_Pm FULL OUTER JOIN Wine_Pe
    ON Wine_Pm.oid=Wine_Pe.oid
WHERE name="Zifandel Clos Marie" AND year=2000
```

The translation of the Mads multiperception schema is completed by describing in the data dictionary of the relational database the set of simple perceptions of the schema and for each simple perception the set of tables that belong to that perception. This information can be organized as one table:

SimplePerceptionTables (perceptionId, tableName)

Another table is required to store the definition of interperception relationship types and the associated object types:

InterPerceptionRelationships (relationshipTable, objectTable)
objectTable REFERENCES SimplePerceptionTables

The content of these two tables for the database of Fig. 5.10 is given in Fig. 5.11. These tables are used by the system when a user begins a working session by opening the multiperception database with either a simple or a composite perception. For example, a user opens the Mads database of Fig. 5.9 – let us call WineDB this database – by issuing the following command:

openDatabase (WineDB, Pm)

The system, after looking at the SimplePerceptionTables table, will give to the user the access rights to the Wine_Pm, Vineyard_Pm, and ProducedBy_Pm tables. On the other hand, if a user issues:

openDatabase (WineDB, (Pm+Pg))

the system will look for the tables to which the user will be given access rights by searching in the SimplePerceptionTables and InterPerceptionRelationships tables. The resulting list of tables will be: Wine_Pm, Vineyard_Pm, ProducedBy_Pm, GeologicalUnit_Pg, and LocatedIn.

SimplePerceptionTables

perceptionId	tableName
Pm	Wine_Pm
Pm	Vineyard_Pm
Pm	ProducedBy_Pm
Pe	Wine_Pe
Pg	GeologicalUnit_Pg

InterPerceptionRelationships

relationshipTable	objectTable
LocatedIn	Wine_Pm
LocatedIn	GeologicalUnit_Pg

Fig. 5.11. The data dictionary of the perceptions of the database in Fig. 5.10.

5.9 Related Work

This section compares the perception mechanism in Mads to other approaches that similarly aim at supporting multiple perspectives on the same information repository, database or ontology. As the concept of perspective is subject to a variety of interpretations, a variety of mechanisms have been developed, first in the database domain, to meaningfully partition an information system into subsets defined for different purposes or characterized by different properties. Views and versions are available in commercial DBMS. Distributed data solutions (e.g., federated databases, multidatabases) have been defined to support modules and are therefore related to the modularization topic of this book, but have had an impact only in the research community, except for the simplest category (distributed databases) that only implies managing multiple storage systems.

The view mechanism is the most widespread and its use is routine work for database administrators. It has also been defined for ontologies, as shown in Part

II of this book. A view is an on-demand personalized data structure (at the logical level) built from the underlying data structures implemented in the database. As discussed in detail in the introductory section of this chapter, 5.1, views provide poorer functionality than our perception concept. Basically, the scope of a view is to provide an application-specific perspective on an object type, while the scope of a perception is to provide an application-specific (conceptual) perspective on the whole database. Moreover, views are mostly intended for data retrieval. Updates can be performed onto a view only if the view derivation process satisfies some quite restrictive rules. For example, one rule states that the columns being modified in the view must directly reference the underlying data in the table columns, and thus, e.g., cannot be derived through an aggregate function, cannot be computed from an expression that uses other columns, or cannot be formed by using set operators such as union, difference or intersection. The reason for these restrictive rules is that the system must be able to unambiguously translate modifications in the view into modifications in the base tables from which the view is derived. In case of ambiguity, an update of a view element can nevertheless be allowed if INSTEAD OF triggers have been manually and explicitly defined by the database administrator to state how any given modification to the data in the view is to be translated into modifications to the underlying base tables. Perceptions, instead, are meant to fully support application-specific data management, not just retrieval. They are therefore updatable, unless specific application constraints are defined to restrict updatability.

Versioning, as the name says, is a mechanism specifically designed to support change management. It allows managing an ordered graph of versions of the same element (document, object, database, ...). Its main functionality is to enable backtracking to previous versions of an element and to retrieve a consistent set of versions of parts of a composite element (typically sections in a document) when these parts have evolved in a-synchronized way in a collaborative environment. The perspective provided by a version is alike a temporal perspective, but instead of looking at the state of affairs at a certain instant in time it looks at the state of affairs at a certain moment of an evolution path. Although tagging with a version identifier can be seen, at least to some extent, as similar to tagging with a perception identifier, the two approaches rely on fundamentally different paradigms. Versions offer successive images of an evolving element, while perceptions offer complementary images of an element taken at the same moment in time. It would not be wise to confuse users (and the system) by offering versioning concepts to support perceptions.

Closer to the Mads perception concept is the contextual module concept proposed by Mylopoulos and Motschnig [MMP95, MP00]. The authors propose a generic abstract model, independent of any specific information model, which supports modules, called contexts. They specify basic rules for defining an information model with modules. An example is the rule stating that elements belonging to several modules should be allowed to have a specific name local to each module. Another rule states that whenever two modules share some elements, they should agree on the propagation of their updates. The Mads approach was defined independently of that work, but its principles are very much in line with the work and its results confirm and refine the ideas in [MMP95, MP00]. However, Mads has been designed and implemented as a data model specifically targeted to support spatio-temporal databases. This emphasized the quest for orthogonality between the structure, space, time, and

perception modeling dimensions, which certainly influenced the way the perception mechanism has been defined.

Let us now turn our attention to ontologies and compare Mads multi-perception databases with modular ontologies. The closest to Mads approach in the ontology world is Cyc's microtheories concept and mechanism. The comparison between the two has been discussed in Chapter 1 of this book, and is not repeated here. The two have almost identical goals. The overall goal of Mads is the creation of a new database composed of modules – the Mads perceptions. The goal of Cyc in this respect is to create an ontology composed of microtheories. These goals are quite different from the goals of the other approaches to modular ontologies that have been investigated and whose major representatives are described in Parts II and III of this book. These other approaches are: ontology partitioning, module extraction, interconnection of existing ontologies. The latter looks at reusing a set of existing ontologies as modules of a broader ontology that is built by inter-connecting the existing ones. The other two approaches propose different ways to create modules from an existing ontology. Partitioning is meant to split an ontology into several modules according to some splitting criteria, while extraction targets creating a module by extracting information from the ontology (similarly as in view materialization).

The creation of a new module (perception) from a running database is also possible in Mads. The definition of an additional simple perception can be done anytime through a schema modification process. First, an identifier has to be specified for the new perception, and second, the definition of the database schema has to be revisited to add the new perception identifier to the set of perceptions associated to the elements (schema and instances) the database administrator wants to see in the new perception. This extensional process has to be validated by checking the consistency rules that enforce a perception to obey the modeling constraints of a normal database. The other mechanism that dynamically creates new perceptions is the composition of existing perceptions into a composite perception. This intensional process is prompted anytime a transaction uses the openDatabase command with a composite perception. The new composite perception remains a virtual one. It is not materialized. Nevertheless, the database administrator can anytime decide to materialize a composite perception, if required.

Let us now compare the constructs supported by the Mads data model for perceptions to the ones supported by approaches that connect existing ontologies. Mads support three kinds of links between perceptions:

1. An implicit link defined by the fact that the different perceptions are integrated into a multiperception definition of an object (or relationship) type with several representations. These representations are, by definition, related to each other, whatever their dissimilarities. The descriptions (attributes, keys) and the populations for each perception may be the same, different, or even disjoint.
2. Two different object types belonging to two different perceptions, but representing at least partially the same real-world entities, may be linked by an interperception multi-instantiation link (is-a or overlap link).
3. Two different object types belonging to two different perceptions may be linked by an interperception relationship type.

The first and second kinds of interperception links, in the case where the populations of the linked types are one included in or equal to the other, are similar to the bridge rules of C-OWL, which allow relating two concepts that, in any interpretation, describe two sets of entities that are linked by an inclusion [BGv+04]. A

difference between the Mads links and the bridge rules is that Mads' first mechanism works even if the populations are disjoint, and Mads' second mechanism works with included or overlapping populations. On the other hand, bridge rules are intended for two concepts related by an inclusion (or an equality). Mads' third mechanism is similar to the link property of E-connections that allows relating two classes from disjoint modules (i.e., modules that describe disjoint parts of the world) by an inter-module role, called link property [CPS08].

The main difference between Mads and approaches that connect existing ontologies is that Mads allows representing the same real-world phenomenon with representations that are quite dissimilar from each other. Two representations may have disjoint populations and still be two representations of the same object type. For instance, a perception of the Wine object type may describe only European wines while another perception may describe only American wines. As another example, in Fig. 5.4 the attribute barrels has two representations that are quite dissimilar, and still in Mads these two representations are related: Users querying the barrels attribute with the composite perception ($P_m + P_e$) get for each wine two values, one for each perception. This possibility of stating that several object types, several relationship types, or several attributes describe the same phenomenon, even if they are totally different, is – as far as we know – peculiar to Mads.

5.10 Conclusion

This chapter has described an approach to database modularization in terms of supporting multiple perceptions over a database and multiple representations of its elements. The new concepts and rules that form the approach are presented as embedded in the Mads conceptual data model. Perception features can thus be applied to the thematic as well to the spatio-temporal characteristics of a database. The chapter focused on discussing perceptions. A detailed description of other Mads features can be found elsewhere [PSZ06a], namely including its concepts, how to use them in database modeling, and the operations to work with these concepts to create and maintain a multiperception database.

Defining data corresponding to a specific perception is equivalent to defining a module in a modular database. To this extent, the perception and module concepts are synonyms. This explains that the Mads approach and solution share many commonalities with the Cyc approach to modular ontologies. However, differences between the goals of Mads and Cyc induce different solutions. Cyc is a huge and still growing ontology where reuse is important. Therefore, the organization of Cyc modules is an inheritance hierarchy, while Mads modules are organized along a composition graph. Mads' goal is to provide different groups of users with different perceptions of the same database. Consequently, all Mads modules share the same interpretation domain, while each Cyc microtheory has its own. Moreover, in Mads the system is aware – and manages – the fact that the same real-world phenomenon is described by several representations. In Cyc two microtheories may contain different representations of the same phenomenon but Cyc ignores it. Users interested in modularizing a knowledge repository from its creation onwards should carefully analyze which goal they are trying to achieve to choose the most suitable solution.

The Mads model has been used in many real-world applications. For example, in a cartographic application at the French Mapping Agency (IGN) the multirep-

resentation features of the model were used for describing the representations of geographic objects at different levels of detail (i.e., resolution). This cartographic application also needed the interperception links to compare the different representations of real-world objects for validation purposes. In another application realized at Cemagref, a research center on risk management, perceptions were used to define different user profiles to obtain customized information from the same database. For example, information about natural risks, such as avalanches or landslides, is delivered to users depending on their profile: the general public obtains validated and less technical information with respect to risk experts.

Future work for the Mads model includes the extension of interperception links between constructs of different kinds, e.g., when the same phenomenon is represented as an object type in one perception and as an attribute or as a relationship type in another perception. In view of targeting semantic Web applications a formalization of the Mads model according to latest W3C standards would be needed. Unfortunately, spatio-temporal semantics is not supported by standard description logics. For this reason we are exploring a complementary approach consisting in using database technology (which somehow knows how to manage spatio-temporal data) to handle ontological data and services. As a first step, a prototype, called OntoMinD, has been developed to store large DL ontologies in an extended object-relational database system. The OntoMinD extension relies on the specification of a set of stored procedures that perform ontological reasoning on the TBox and ABox [AJPS08].

References

AJPS08. L. Al-Jadir, C. Parent, and S. Spaccapietra. OntoMind: Reasoning with large DL ontologies stored in relational databases. In preparation, 2008.

APS07. A. Artale, C. Parent, and S. Spaccapietra. Evolving objects in temporal information systems. *Annals of Mathematics and Artificial Intelligence*, 50(1–2):5–38, 2007.

BGv+04. P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. Contextualizing ontologies. *Journal of Web Semantics*, 1(4):325–343, 2004.

CPS08. B. Cuenca Grau, B. Parsia, and E. Sirin. Ontology integration using E-connections. Chapter 12 in this book, 2008.

Cyc06. Cycorp. What is a context? `http://www.cyc.com/cycdoc/course/what-is-a-context.html`, 2006.

MMP95. J. Mylopoulos and R. Motschnig-Pitrig. Partitioning information bases with contexts. In *Proceedings of the 3rd International Conference on Cooperative Information Systems, CoopIs'95*, pages 44–54, 1995.

MP00. R. Motschnig-Pitrig. A generic framework for the modeling of contexts and its applications. *Data and Knowledge Engineering*, 32(2):145–180, 2000.

Ope06. Open Geospatial Consortium Inc. OpenGIS Implementation Specification for Geographic information – Simple feature access – Part 2: SQL option. OGC 06-104r3, Version 1.2.0, 2006.

PSZ06a. C. Parent, S. Spaccapietra, and E. Zimányi. *Conceptual Modeling for Traditional and Spatio-Temporal Applications: The MADS Approach*. Springer-Verlag, 2006.

PSZ06b. C. Parent, S. Spaccapietra, and E. Zimányi. The MurMur project: Modeling and querying multi-represented spatio-temporal databases. *Information Systems*, 31(8):733–769, 2006.

SPZ07. S. Spaccapietra, C. Parent, and E. Zimányi. Spatio-temporal and multi-representation modeling for supporting active conceptual modeling of learning, ACM-L. In *Proc. of the 1st International Workshop on Active Conceptual Modeling of Learning*, LNCS 4512, pages 194–205. Springer-Verlag, 2007.

Index