Complex Systems Informatics and Modeling Quarterly
CSIMQ, Issue 7, June/July 2016, Pages 25–50
Published online by RTU Press, https://csimq-journals.rtu.lv
http://dx.doi.org/10.7250/csimq.2016-7.02
ISSN: 2255-9922 online

Static and Dynamic Aspects of Application Domains: An Inductively Defined Modeling Technique That Allows Decomposition

Theo P. van der Weide, Fiona P. Tulinayo and Patrick van Bommel

Institute of Computing and Information Sciences, Radboud University, Comeniuslaan 4, 6525 HP Nijmegen, Netherlands

{tvdw, pvb}@cs.ru.nl, ftulina@gmail.com

Abstract. Modeling is one of the most important parts of requirements engineering. Most modeling techniques focus primarily on their pragmatics and pay less attention to their syntax and semantics. Different modeling techniques describe different aspects, for example, Object-Role Modeling (ORM) describes underlying concepts and their relations while System Dynamics (SD) focuses on the dynamic behavior of relevant objects in the underlying application domain. In this paper we provide an inductive definition for a generic modeling technique. Not only do we describe the underlying data structure, we also show how states can be introduced for relevant concepts and how their life cycles can be modeled in terms of System Dynamics. We also show how decomposition can be applied to master complex application domains. As a result we get an integrated modeling technique covering both static and dynamic aspects of application domains. The inductive definition of this integrated modeling technique will facilitate the implementation of supporting tools for practitioners.

Keywords: Object-Role Modeling, static aspects, dynamic aspects, inductive definition, decomposition.

1 Introduction

Requirements can be seen as high-level abstractions of services with precisely stated constraints that are provided by a method under which they function [1], [2]. Vessey asserts that, humans have limited information processors, therefore, decomposing the problem under study into sub-problems is of great importance [2]. He further notes that, the key to successful problem decomposition is to structure the problem into sub-problem solutions which can be integrated to form a complete solution. Wieringa [3] admits that integration of methods is the act of unifying method elements together methodically. Wieringa also notes that the process of integrating methods requires defining relationships between the methods, precise understanding of the relations between the methods and understanding of the underlying ideas of the methods, which then lead to a more principled exposition of the methods. In [4], in the context of System Dynamics (SD), it is noted that integration could make computerization of system modeling and operating of complex systems easier. This paper therefore contributes towards improving SD model conceptualization by introducing an inductive definition and introducing states to a static modeling method called Object-Role Modeling (ORM).

ORM is applicable for conceptualizing and explicitly representing processes in a problem domain. The philosophy behind ORM is that it tries to describe a Universe of Discourse (UoD) by describing the communication between its members. An ORM scheme basically is a grammar

describing that communication. This grammar is also referred to as information grammar. The information grammar describes the elementary sentences that are valid in the associated UoD. From these sentences other sentences may be formed. Object Role Calculus (ORC) ([5]) and ORM2 ([6]) are examples of such generic systems for constructing sentences. These sentence will be referred to as information descriptors. ORC is the base for formal reasoning about the UoD. A major aspect during the design phase may be a (quantitative) simulation of the model in order to find bottlenecks for the implemented system. Such simulations are quite common in construction technology, for example, to prohibit a bridge from destruction by a too heavy load.

It is stressed here that we do not aim at introducing a completely new modelling technique. Rather, we build a generic bridge between existing techniques, such as techniques for modelling static aspects and techniques for modelling dynamic aspects. Building such a bridge will certainly not result in a decreased expressivity. For an overview on expressivity in conceptual data modeling, see [7].

This work is motivated by the fact that different modelling techniques have different communities of people who are using those techniques. The exchange of theoretical foundations between these communities is now getting more and more attention, but much more work needs to be done.

In order to generate research results which are acceptable and applicable to different communities, we have adopted a very general mechanism for the specification of models: inductive definitions. At this moment no inductive definitions for these modelling techniques are available, despite the inductive description mechanism is very powerful.

Inductive definitions use a (recursive) style in which basic elements are refined or extended in a stepwise manner. When reasoning about models and modelling techniques, the complete line of reasoning can be built using basic elements and more advanced elements. All advanced elements are ultimately defined in terms of basic elements, as in the well-known approaches of natural induction and structural induction.

The rest of this paper proceeds as follows. In Section 2 we present an inductive definition of ORM. In Section 3 we introduce the concept of states, and describe a decomposition mechanism to structure complex state structures. We also show how these states can be related to SD. In Section 4 we draw some conclusions.

2 Modeling Basics

In this section we will provide an inductive definition of information structures. This definition is compatible with most modern data modeling techniques. The advantage of an inductive definition is that it can follow the step-by-step construction of an information structure and thus can be the basis for building a supporting construction tool. Another advantage is that properties of information structure can be defined and proved inductively.

We start in Subsection 2.1 with some background and history of ORM. Then in Subsection 2.2 the inductive definition is described, followed in Subsection 2.3 with some examples. In Subsection 2.4 we describe and prove some properties, and, finally, in Subsection 2.5 the semantics are formally described by an inductive definition.

2.1 Background

ORM evolved from Natural-language Information Analysis Method (NIAM) [8]. It was developed in the early 70's as a fact-oriented approach for modeling information and querying the information content of a business domain at a conceptual level [9]. Originally, ORM was conceived for data modeling purposes and takes a static perspective on the domain in the sense that it aims at capturing the fact types and entity types that play potentially a role in the (dynamic) domain [10]. Fact-orientation means that it includes both, types and instances, in its models; types are

called ‘fact types’, instances ‘facts’. It supports n-ary relations, has an expressive and stable graphical notation. Its diagrams can be automatically verbalized into pseudo natural language sentences which makes it resistant to changes that cause attributes to be remodeled as object types or relationships [11]. Its formal specification and semantics are defined in [12], [13], [14]. Here we use ORM [10] and the Predicator Set Model (PSM) [7] to provide a conceptual description of object types and their relations. Objects are instances that behave according to this description. In ORM terminology, an object basically is the set of its properties. So objects with the same set of properties are assumed to be the same. The properties are used to identify the object during the communication with human beings (say). To simplify that identification, in data modeling techniques a fixed combination of properties enforces unique object identification. In the linguistically oriented ORM terminology, this is referred to as the standard name for that object.

In this paper we follow the approach originating in the Natural-language Information Analysis Method (NIAM), in which object types play roles in fact types, see [15], [16] and [17]. A NIAM model was the basis to construct the controlled semi-natural language Reference and IDea Language (RIDL, see [18] and [19]) for communication with the database. In [7] the term predicator (introduced in [20]) was used to describe the Predicator Set Model (PSM) as an extension of NIAM, and Object Role Calculus (ORC [5]) as an extension of RIDL. The ORM modeling technique [10] can be seen as a standardization of the modeling techniques sharing the NIAM ideas. The modeling process and the required competencies of its participants are described in [21].

The first proposal for adding a process-oriented perspective to the fact-based approach was the integration with activity diagrams, see for instance [22]. Many other proposal have been made, see, for example, PSM^2 ([23]) extending PSM with action types. In [24] object-role modeling (ORM) is integrated with System Dynamics (SD). As a central issue we see that states are added to object types. Actions then are seen as state-transformers.

In this paper we want to redefine the basic part of ORM techniques, the definition of the underlying meta-model for the information grammar. We base ourselves on PSM [7] because of its rigourous formal definition. The definition provides a set of structural rules required for valid models. A (concrete) model is an instantiation of these structural rules. Being a concrete model does not imply that the model is constructible in terms of elementary construction steps. In this paper we therefore focus on a constructive domain definition, also referred to as an inductive definition. An inductive definition describes the initial state and the construction operators to obtain other domains. Then in Section 3 we focus on extending object types with states. Conclusions are presented in Section 4.

2.2 Definition of ORM

An information structure *IS* based on an alphabet Σ and a set of domains *Doms* is introduced as a structure consisting of the following components:

$$IS = \langle \Sigma, Doms, Labels, Facts, Deriveds, Specs, Gens, Sets, Constraints, isTop, isSpec \rangle$$

where *Labels* are the most simple object types, *Facts* are the basic construction mechanism for new object types, *Deriveds* are derived object types to be used to form more complex constructs, *Sets* are object types that are instantiated by sets of objects, and *Constraints* are population constraints for the constructed information structures. Furthermore, there are special object types that are obtained either by specialization (*Specs*) or by generalization (*Gens*). This induces a hierarchy of object types. *Top* is the function assigning to each object type its top (also referred to as pater familias), and *isSpec* is a binary relation over object types describing the specialization relation.

We will refer to the set of all object types defined in information structure *IS* as $Objs(IS) = Labels \cup Facts \cup Deriveds \cup Specs \cup Gens \cup Sets$. As a consequence of our definition below each

object type X will have (i) a unique name, which is referred to as $Name(X)$, and (ii) a verbalizing expression referred to as $vRule(X)$. The verbalization rule is used as the base formulation for this object type in the conceptual modeling language. If N is the name of an object type, then $Obj(N)$ will refer to this object type. Name N is unused in IS if none of the object types has this name: $\forall_x [Name(x) \neq N]$.

As an abbreviation, we will use the following convention to address the individual components of information structure: $Doms(IS)$ is the set of domains in IS, $Labels(IS)$ is the set of label types, etc.

2.2.1 The Basic Construction Rules

Let Σ be some alphabet and $Doms$ be some set of domains (such as boolean, integer, real and string), then a basic information structure is inductively defined as follows.

Rule 1. Empty scheme $\langle \Sigma, Doms, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing \rangle$ is a basic information structure, referred to as the empty information structure, denoted $\varnothing_g(\Sigma, Doms)$, or simply $\varnothing_g$ when Σ and $Doms$ are obvious from the context.

Rule 2. Introducing the label type Let (i) IS be an information structure, (ii) N be an unused name, (iii) $D \in Doms(IS)$ be some domain and (iv) let $\sigma = \omega_1 D \omega_2$ with $\omega_1, \omega_2 \in \Sigma^*$ be a verbalization expression for label instances, then $Label(N, D, \sigma)$ is a (new) label type named N taking values from domain D and verbalized via σ. The information structure resulting after adding this label type to IS is denoted $IS + Label(N, D, \sigma)$. See Figure 1 for an example and a graphical representation of this rule.
In many cases ω_1 is used to qualify the value from D. For numeric values ω_2 may be used to add a physical dimension, but in many cases ω_2 will be the empty string. It may also happen that both ω_1 and ω_2 are empty strings.

Rule 3. Introducing the fact type Let (i) IS be an information structure, (ii) N be an unused name, (iii) $r_1, \ldots, r_k$ (for some $k > 0$) be so-called role names, (iv) the objects types $X_1, \ldots, X_k$ playing these roles be elements from $Objs(IS)$ and (v) let $\sigma = \omega_0 r_1 \omega_1 \ldots r_k \omega_k$ with $\omega_i \in \Sigma^*$ for $0 \leq i \leq k$ be a verbalization expression for fact instances, then the combination $Fact\left(N, \left\{r_i{:}X_i \;\middle|\; 1 \leq i \leq k\right\}, \sigma\right)$ is a fact type named N consisting of roles $\left\{\langle r_i, X_i \rangle \;\middle|\; 1 \leq i \leq k\right\}$. The information structure resulting after adding this fact type to IS is denoted $IS + Fact\left(N, \left\{r_i{:}X_i \;\middle|\; 1 \leq i \leq k\right\}, \sigma\right)$. See Figure 1 for an example and a graphical representation of this rule.
We will use $Preds(IS)$ to denote the set of all roles in all fact types in information structure IS. If $p \in Preds(IS)$, then $Fact(p)$ indicates the fact type in which the role has been introduced, and $Base(p)$ indicates the name of the object type associated with p.

Roles in different fact types may have the same role name. In such a case, the name of the corresponding fact type is used for disambiguation.

Introduction of basic object types (label types) and being able to make relations between object types is the basis for each data modeling technique. For example, the relational data model can be seen as a modeling technique that is restricted to basic construction rules. Furthermore, the relational model does not introduce (explicitly) verbalization rules. The correspondence is as follows: roles are the attributes, the fact types are the relational schemes.

Entity types are considered to be *objectifications of fact types* from *Facts*. This definition is in line, for example, with the approach taken in Fully Communication Oriented Information Modeling (FCO-IM) ([25]) and with the relational data model ([26]). Therefore we will not

explicitly add entity types to the definition of an information structure. For graphical conventions, see Figure 1.

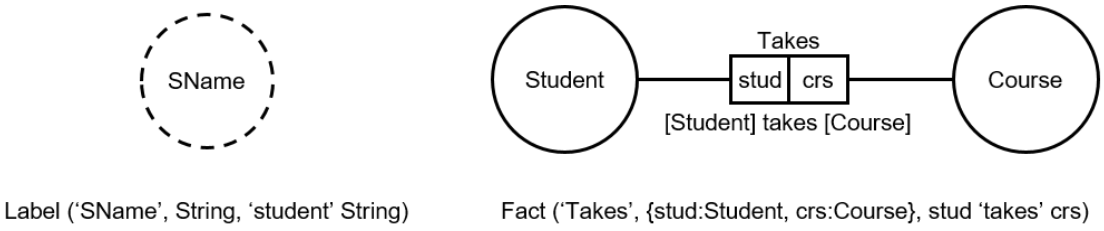

Figure 1. Graphical representation of label types and fact types

2.2.2 Introducing the Conceptual Language

The basic naming of information structure elements (object types and roles) is used to introduce a conceptual language (for more advanced verbalization strategies, see [27]). In this paper, we will use the ORC approach, see [5], but other modeling techniques (such as UML) have their own variants to introduce a similar conceptual language. Let $\pounds(IS)$ be the conceptual language that is derived from information structure *IS* following [5]. We assume that the conceptual language defines the following syntactic categories:

1. *Fact formulation*: this syntactic category is used to describe elementary or compound facts, and is used to inform an information system about new information.
2. *Information descriptor*: this syntactic category is used to describe information subsets.
3. *Condition*: this syntactic category is used to describe conditions on information descriptors.

We will assume the following special constructions for information descriptors:

1. In this paper we will use the constructor OR OTHERWISE (see [5]) to combine information descriptors by taking the union of their results. We will use the shorthand notation $\oplus$ for this operator.
2. We furthermore use the constructor AND ALSO as an operator to intersect information descriptor results. We will use $\otimes$ as a shorthand notation for this operator.
3. We also use the constructor BUT NOT as the set difference operator for information descriptors. We will use $\ominus$ as a shorthand notation for this operator.
4. Finally, we assume the check operator EXISTS that takes a fact formulation as its argument and returns *true* if that fact formulation, interpreted as an information descriptor, returns a non-empty result. We will use $\exists$ as a shorthand notation for this operator.

2.2.3 Adding Derived Object Types

In some cases it will be convenient to define special sets of instances in order to introduce special relations. Information descriptors from the conceptual language ORC are used to describe such special sets of instances. A derived object type is the objectification of an information descriptor:

Rule 4. Introducing the derived type Let *IS* be an information structure, let *N* be an unused name, and let σ be an information descriptor taken from the conceptual language $\pounds(IS)$, then $Derived(N,\sigma)$ is a derived type, that is instantiated by the instances resulting from evaluating σ. The information structure resulting after adding this derived type to *IS* is denoted $IS + Derived(N,\sigma)$.

In the remaining part of this paper we will use derived object types to introduce some special constructs.

2.2.4 Adding Constraints

The next mechanism that we introduce is the constraint, a rule that describes conditions that are required to hold for any instantiation of the information structure.

Rule 5. Introducing the constraint Let IS be an information structure, let N be a unique name and C a condition from the conceptual language $\pounds(IS)$, then $Constr(N,C)$ is a constraint imposed on IS. The information structure resulting after adding this constraint IS is denoted $IS + Constr(N,C)$.

In Section 2.5 we will consider the semantics of constraints in more detail.

2.2.4.1 Identity Related Constraints

In various modeling techniques, frequently occurring constraint types have associated an easily usable graphical representation. A powerful system of constraint types helps modelers to more precisely and adequately describe the semantics of the UoD underlying their modeling activity. All modeling techniques have special notations to enforce that roles are required to be played (total role or mandatory role) and that the instance can play the role at most once (unique role). These constraints are essential to handle object identification since they can enforce that each instance of some object type is related to some unique instance from the application domain, and thus serve as a mechanism for identification. We will give a more generic description, and add the extensional uniqueness constraint that is to be used when instances are identified by sets of instances (as is the case for mathematical sets, see also [28]):

1. *Total role or mandatory role*. Let R be a set of roles with a common base, then the constraint $total(R)$ requires all instances of the common base to play at least one of the roles from R. See Figure 2 for the ORM style to denote this type of constraint. A single-role mandatory constraint is represented by marking the role with a small dot. In case of multi-role mandatory constraint, a small circle (as a representation for a constraint) with a dot embedded is used.
2. *Unique role*. Let R be a set of roles, then the constraint $unique(R)$ requires instances of the (derived) relation $UniQuest(R)$ to be unique in the combination of roles in R. When all roles in R belong to the same fact type f (or $\forall_{r \in R}[Fact(r) = f]$), then $UniQuest(R) = f$. When R contains roles from different fact types, then $UniQuest(R)$ is a special derived fact type resulting from joining fact types from roles in R. A detailed description of the derived relation $UniQuest(R)$ can be found in [14].
   See Figure 2 for the ORM style to denote this type of constraint. A single-role unique role constraint is represented by marking the role with a small arrow. For a multi-role unique role constraint a small circle with the letter u embedded is used. For the constraint $unique(\{p_1\})$ we have, $UniQuest(\{p_1\}) = Fact(p_1)$, while $UniQuest(\{p_1, q_1\}) = Fact(p_1) \bowtie_{p_2=q_2} Fact(q_1)$.
3. *Extensional uniqueness*. Let R be a set of roles, then the constraint $extuniq(R)$ specifies that R-instances are identified via the derived relation $UniQuest(R)$ in a set-based fashion.
   See Figure 2 for the ORM style to denote this type of constraint.

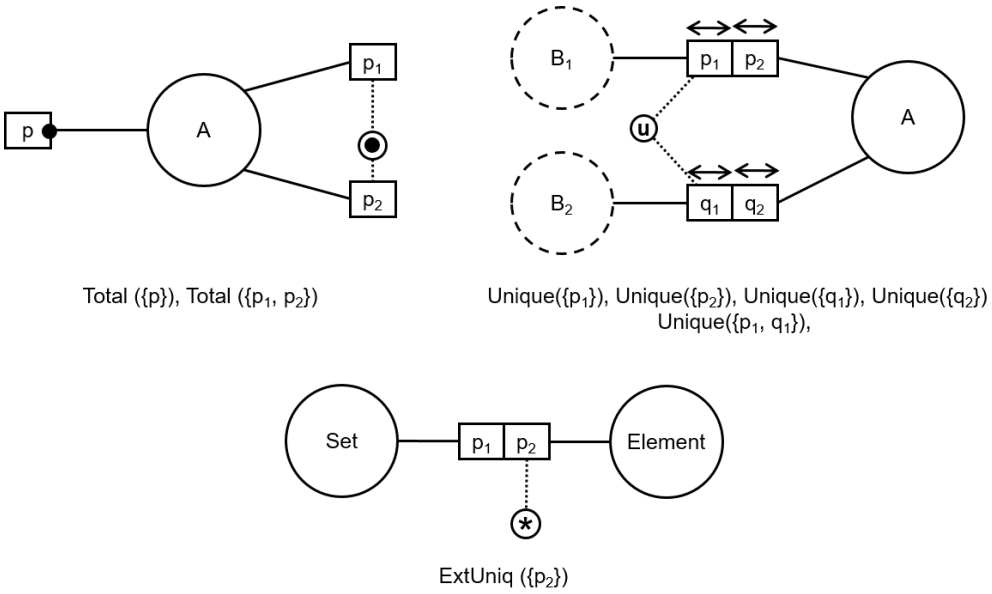

Figure 2. The identity related constraints

2.2.4.2 Role Participation Constraint

Another special constraint type is the role participation constraint. Let r be a role and $\rho \in \mathcal{L}(IS)$ be an information descriptor, then role participation constraint $Part(r,\rho)$ states that only instances that result from evaluating ρ are entitled to play role r. Constraint $mPart(r,\rho)$ makes this role playing mandatory. In Figure 3 we see the two manifestations of the role participation constraint. In both cases the affected role is r, being constrained by information descriptor ρ. For convenience we also show r's associated base $Base(r) = A$. In the left-hand situation the participation rule ρ restricts the possible candidates for participation in role r to only those instances from A that result from the evaluation of ρ. In the right-hand situation it also makes participation mandatory for all such instances.

Figure 3. The role participation constraint

2.2.5 Specialization

Specialization allows the modeler to introduce specialized versions of object types. In more advanced situations we may have multiple inheritance. For example, we may specify *Supervisors* and *Non-smokers* as two specializations of the object type *Employee*. Then *Senior non-smoking supervisors* are a specialization of both *Supervisors* and *Non-smokers*. Using specialization, the modeler can specify that some role may only be played by particular, well-defined instances of the supertype(s). As a consequence, specialization can be seen as the intended automatic addition of a participation constraint to subsequent roles. But, rather than adding these participation constraints, the roles are attached to the (derived) object type that is instantiated by the instances that are allowed to participate. Therefore, the specialization constructor may be seen as the *objectification of an information descriptor* from $\mathcal{L}(IS)$, describing a subset of some given object type.

Rule 6. Introducing the specialization Let IS be an information structure, let N be a unique name, let P be a set of object types from IS, all with the same top element A, and let ρ be an information descriptor from the conceptual language $\pounds(IS)$, then $S = Spec(N, P, \sigma)$ introduces an object type named N as the specialization of the object types from P according to the specialization rule ρ, as follows:

$$Derived\left(N, \sigma \otimes \bigotimes_{Y \in P} Name(Y)\right)$$

Furthermore, also S has top element A, or, $Top(S) = A$, and the specialization relation is extended for the newly introduced object type: $isSpec(Obj(N), Y) \Leftrightarrow Y \in P \vee \exists_{Z \in P}[isSpec(Z, Y)]$. See Figure 4 for a graphical representation of the specialization.
The information structure resulting after adding this specialization to IS is denoted $IS + Spec(N, X, \rho)$.

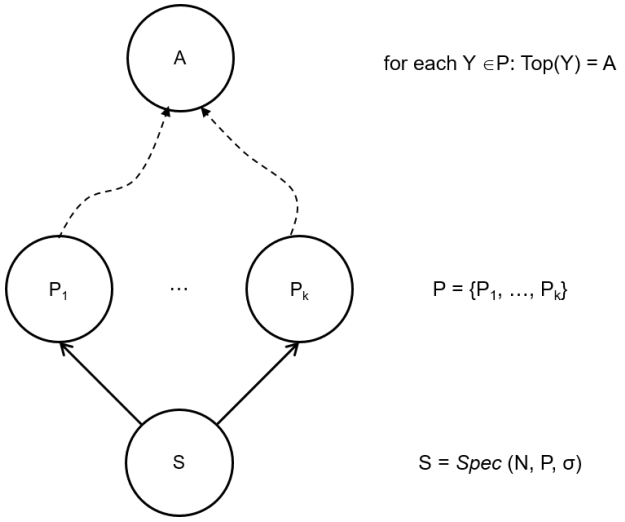

Figure 4. The definition of specialization $S = Spec(N, P, \sigma)$

2.2.6 Generalization

Generalization combines different object types and is used with the purpose of assigning shared properties to each of the specific underlying objects types. The generalization constructor may be seen as the *objectification of the special information descriptor* that is obtained as a disjoint union of the object types being generalized.

We will introduce generalization via 2 constructors. The reason is that this will allow the introduction of recursive information structures.

Rule 7. Introducing the generalization Let (1) IS be an information structure, (2) N a unique name, and let (3) X be an object type, then $Gen(N, X)$ is a generalization type named N, generalizing instances of object type X. The information structure resulting after adding this generalization to IS is denoted $IS + Gen(N, X)$.

Rule 8. Extension generalization Let (1) *IS* be an information structure, (2) $Gen(N,X)$ be a generalization from the *Gens* introduced so far in *IS*, and (3) let Y be an object type, then $GenExt(N,Y)$ extends the generalization named N with the object type Y. The information structure resulting after adding this generalization extension to *IS* is denoted $IS + GenExt(N,Y)$.

The graphical representation of the generalization is shown in Figure 5.

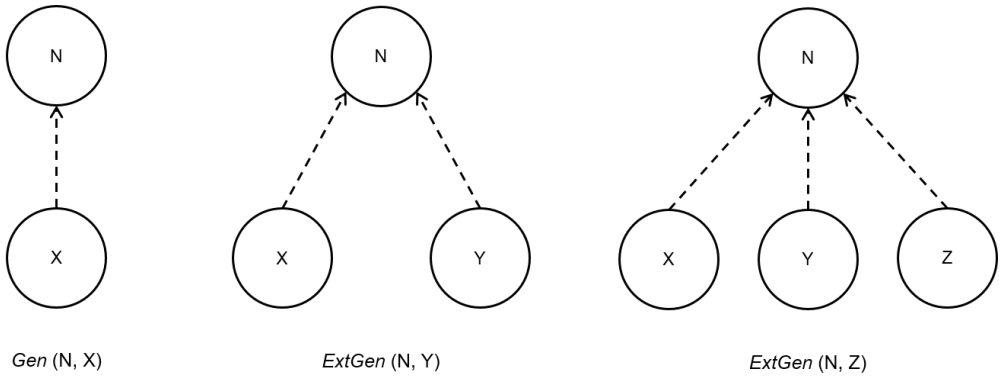

Figure 5. Three generalization steps

2.2.7 Set Type

It is possible to introduce an object type that is instantiated by sets taken from its underlying base object type.

Rule 9. Introducing the set Let (1) *IS* be an information structure, (2) N be a unique name, and (3) let X be some object type, then $S = Set(N,X)$ is a set type named N instantiated by sets of base type X. The set type may be graphically represented as a circle encircling its base type (see Figure 6). An alternative is to use a special connection line between the base type and the set type.
With this object type the following implicit fact type (see Figure 6, see also Figure 1)

$$Fact\left(\_S\_X, \left\{\_sN{:}S, \_eX{:}X\right\}, \varepsilon\right)$$

where $\_N\_X$, $\_sN$ and $\_eX$ are new names, is automatically added to the information structure, and also the extensional uniqueness constraint

$$Constr\left(Nrule, extuniq\left\{\_eX\right\}\right)$$

to guarantee S to be identifiable.
The resulting information structure is denoted $IS + Set(N,X)$.

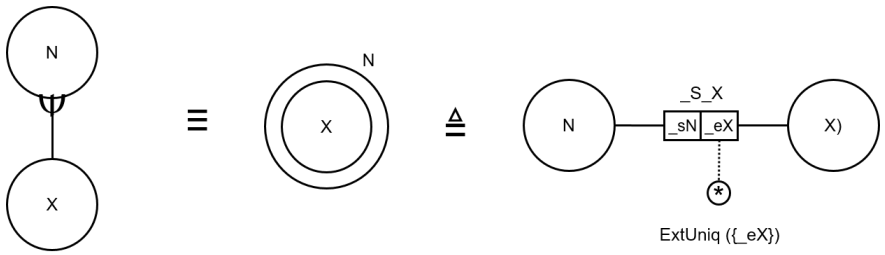

Figure 6. The definition of set type $S = Set(N, X)$

2.2.8 Closure

The final step of the inductive definition of information structures is the closure rule:

Rule 10. Closure Only what can be formed by these information structure composition rules is an information structure.

The consequence of the inductive definition is that properties of information structures can be proven by structural induction. Let Φ be a property for information structures, then the validity of $\Phi(IS)$ for all information structures can be proven by showing that the property holds for the empty information structure $\varnothing_g$, and is invariant under each construction operator. This is described by the following scheme of structural induction:

If we can prove:
1. (base) $\Phi(\varnothing_g)$
2. (step) $\forall_{IS,X}\left[\Phi(IS) \wedge IS + X \textit{ is an information structure} \Rightarrow \Phi(IS + X)\right]$

then we can conclude $\forall_{IS}\left[\Phi(IS)\right]$.

2.3 Some Examples

In the following examples, we assume the following set of underlying domains: $Doms = \{\text{Int}, \text{String}\}$. Let Σ be a standard alphabet.

1. The relational datamodel assumes a set $\mathcal{A}$ of attributes. Each attribute A has associated a domain $Dom(A)$ of possible values. Following conventions from relational algebra [26], the term *relational scheme* is used to refer to a set of attributes. An example is the relational scheme
   Takes (Student, Course)
   where the attributes *Student* and *Course* both have associated domain String (Figure 7). This scheme is constructed by:

$$\begin{aligned}&\varnothing_g + Label(\text{Student}, \text{String}, \text{`student' String})\\ &\quad + Label(\text{Course}, \text{String}, \text{`course' String})\\ &\quad + Fact\left(\text{Takes}, \{\text{stud:Student}, \text{crs:Course}\}, \text{stud `takes' crs}\right)\end{aligned}$$

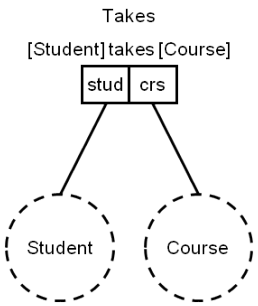

Figure 7. Students taking courses

The associated information grammar has the following format:

(a) terminal symbols: Σ

(b) nonterminal symbols: Student, Course, Takes, Takes.stud, Takes.crs, String

(c) The grammar rules are:

Student ::= 'student' String
Course ::= 'course' String
Takes ::= Takes.stud 'takes' Takes.crs
Takes.stud ::= Student
Takes.crs ::= Course
String ::= c — for each $c \in \Sigma$

2. The nested relational datamodel is an extension of the relational datamodel, allowing nested tables. In comparison with the relational model, an attribute may have a tabular value. Or, the value of an attribute can be a set of tuples from a table. An example, taken from [29], is the following NF^2 scheme:

Cities (zipcode, has_comp (name, president, produces (id, color)))

This nested table contains for cities (identified by their zip code) the set of companies enclosed. A company is recorded by its name, president and the set of produced products. A product is identified as a combination of product id and color. The reason for storing the information in this nested fashion is to avoid redundancies.
In Figure 8 this scheme is graphically displayed.
The construction of this scheme is as follows:

$$
\begin{aligned}
\varnothing_g &+ \mathit{Label}(\text{Id}, \text{String}, \text{'product sort' String}) \\
&+ \mathit{Label}(\text{Color}, \text{String}, \text{'color' String}) \\
&+ \mathit{Fact}\big(\text{Produces}, \{\text{prId:Id}, \text{prCol:Color}\}, \text{prId 'is produced in' prCol}\big) \\
&+ \mathit{Set}(\text{Product}, \text{Produces}) \\
&+ \mathit{Label}(\text{Name}, \text{String}, \text{'company' String}) \\
&+ \mathit{Label}(\text{President}, \text{String}, \text{'president' String}) \\
&+ \mathit{Fact}(\text{Commit}, \{\text{cNm:Name}, \text{cPre:President}, \text{cPro:Product}\}, \\
&\quad \text{cNm 'with' cPre 'produces product set' cPro}) \\
&+ \mathit{Set}(\text{Company}, \text{Commit}) \\
&+ \mathit{Label}(\text{ZipCode}, \text{String}, \text{'zipcode' String}) \\
&+ \mathit{Fact}\big(\text{Cities}, \{\text{zip:Zipcode}, \text{cmp:Companies}\}, \text{ 'at' zip 'are located' cmp}\big)
\end{aligned}
$$

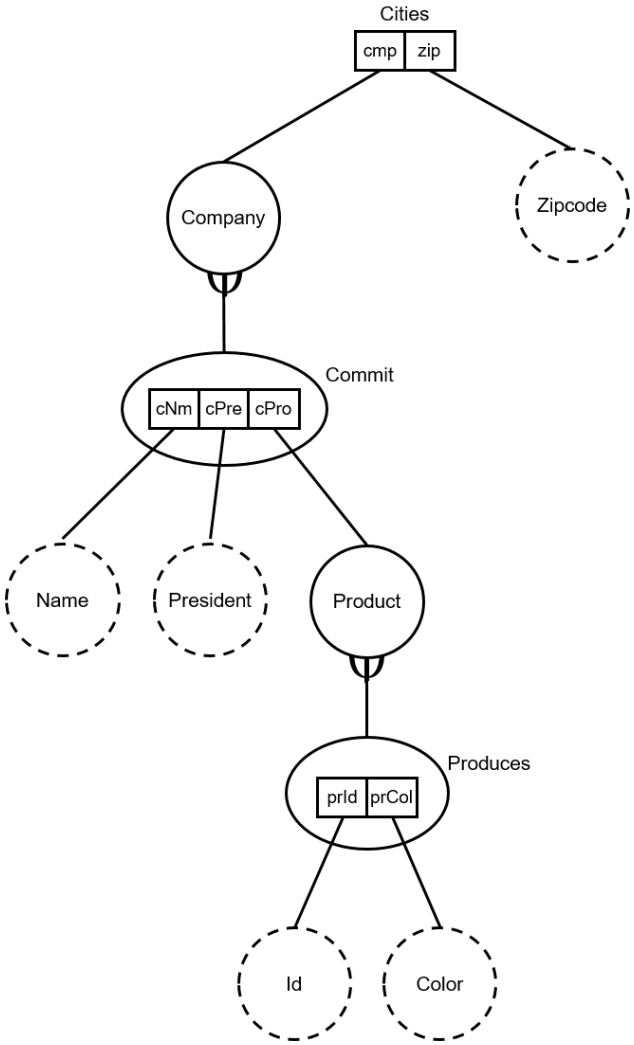

Figure 8. Companies and their products

3. Consider the conceptual scheme from Figure 9 describing students, courses and students taking courses. This scheme is similar to the scheme from Figure 7, but uses the style of FCO-IM ([25]). The scheme from Figure 9 is constructed as follows:

$$
\begin{aligned}
\varnothing_g &+ \mathit{Label}(\text{SName}, \text{String}, \text{String}) \\
&+ \mathit{Label}(\text{CName}, \text{String}, \text{String}) \\
&+ \mathit{Fact}\left(\text{Student}, \{\text{has:Sname}\}, \text{`student '} \ \text{SName}\right) \\
&+ \mathit{Fact}\left(\text{Course}, \{\text{has:Cname}\}, \text{`course '} \ \text{CName}\right) \\
&+ \mathit{Fact}\left(\text{Takes}, \{\text{stud:Student}, \text{crs:Course}\}, \text{std `takes' crs}\right)
\end{aligned}
$$

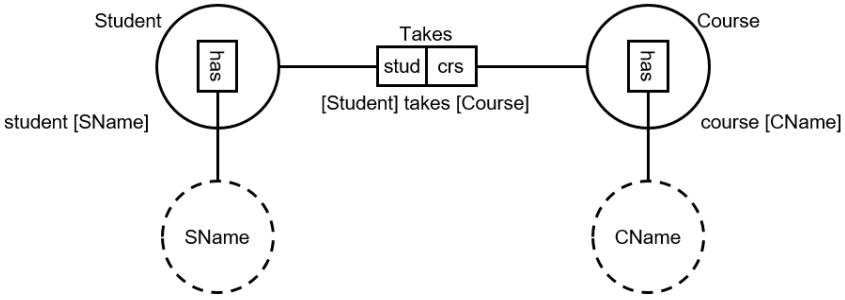

Figure 9. Students and courses revisited

4. The scheme from Figure 10 models a typical graph database as introduced in [30]. A node in this graph basically is a set of properties, where each property is a key-value pair. However, if a property set contains a property with key 'Label', then the set is an edge. Edges have starting and ending nodes. This resulting scheme is constructed as follows:

$$
\begin{aligned}
&\varnothing_g + \mathit{Label}(\mathrm{KName}, \mathrm{String}, \mathrm{String}) \\
&\quad + \mathit{Label}(\mathrm{CVal}, \mathrm{String}, \mathrm{String}) \\
&\quad + \mathit{Fact}\left(\mathrm{Key}, \{\mathrm{with{:}KName}\}, \text{'key' KName}\right) \\
&\quad + \mathit{Fact}\left(\mathrm{Value}, \{\mathrm{with{:}CVal}\}, \text{'value' CVal}\right) \\
&\quad + \mathit{Fact}\left(\mathrm{Property}, \{\mathrm{key{:}Key}, \mathrm{value{:}Value}\}, \text{'[' Key ':' Value ']'}\right) \\
&\quad + \mathit{Set}(\mathrm{PropertySet}, \mathrm{Property}) \\
&\quad + \mathit{Spec}(\mathrm{Edge}, \{\mathit{PropertySet}\}, \\
&\qquad \text{PropertySet CONTAINING Property having Key with Kname "Label"}) \\
&\quad + \mathit{Spec}\left(\mathrm{Node}, \{\mathit{PropertySet}\}, \text{ProperySet BUT NOT Edge}\right) \\
&\quad + \mathit{Fact}\left(\mathrm{Start}, \{\mathrm{node{:}Node}, \mathrm{edge{:}Edge}\}, \text{Edge ' starts in' Node'}\right) \\
&\quad + \mathit{Fact}\left(\mathrm{End}, \{\mathrm{node{:}Node}, \mathrm{edge{:}Edge}\}, \text{Edge ' ends in' Node'}\right)
\end{aligned}
$$

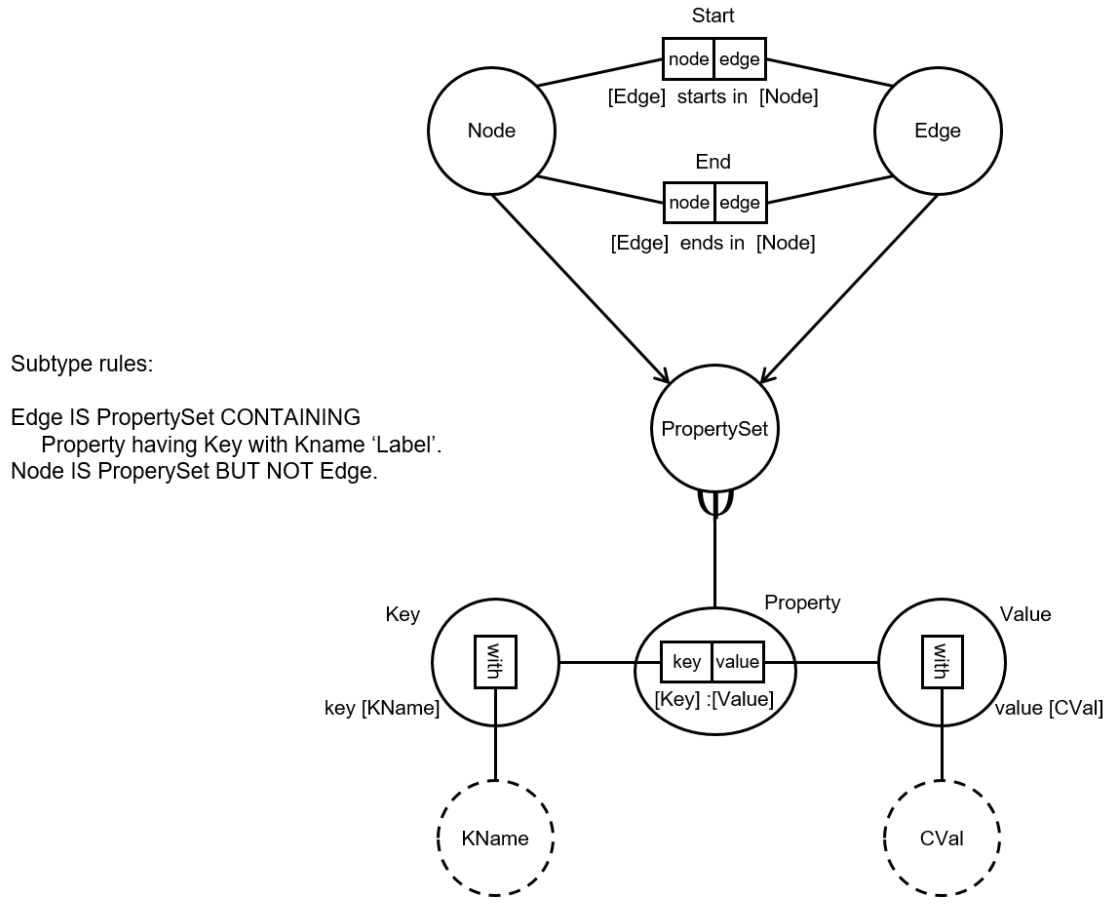

Figure 10. Meta scheme for graphical databases

The specialization rule is constructed according to the rules from ORC [5].

5. In Figure 11 we see a scheme describing the structure of simple expressions, using a set of variables and a single composition operator F.
This example shows how the inductive definition of a recursive datatype is described in terms of a generalization in ORM. In this example, expressions are defined either as a single variable or as the function F applied on a variable and an expression. The generalization is build in two

generalization steps, the first corresponding to the base of the inductive definition, the second corresponding to the construction step of the inductive definition.

$$\begin{aligned}&\varnothing_g + \mathit{Label}(\text{VName}, \text{String}, \text{String})\\ &\quad + \mathit{Fact}\left(\text{Variable}, \{\text{with:VName}\}, \text{‘variable’ VName}\right)\\ &\quad + \mathit{Gen}(\text{Expression}, \text{Variable})\\ &\quad + \mathit{Fact}\left(\text{F}, \{\text{arg1:VName}, \text{arg2:Expression}\}, \text{‘F (’ Variable ’, ’ Expression ’)’}\right)\\ &\quad + \mathit{GenExt}(\text{Expression}, \text{F})\end{aligned}$$

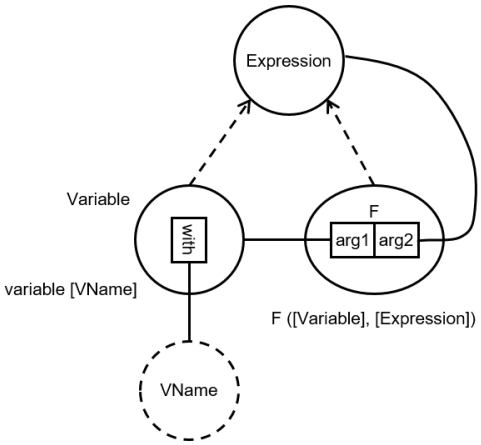

Figure 11. Construction of simple expressions

6. The scheme from Figure 12 introduces a set of (elementary) states. Each elementary state is a state, but also a grouping of states can act as a (compound) state. Then the scheme from Figure 12 is constructed as follows:

$$\begin{aligned}&\varnothing_g + \mathit{Label}(\text{SName}, \text{String}, \text{String})\\ &\quad + \mathit{Fact}\left(\text{ElemState}, \{\text{with:Sname}\}, \text{’state’ SName}\right)\\ &\quad + \mathit{Gen}(\text{State}, \text{ElemState})\\ &\quad + \mathit{Set}(\text{StateGroup}, \text{State})\\ &\quad + \mathit{GenExt}(\text{State}, \text{StateGroup})\end{aligned}$$

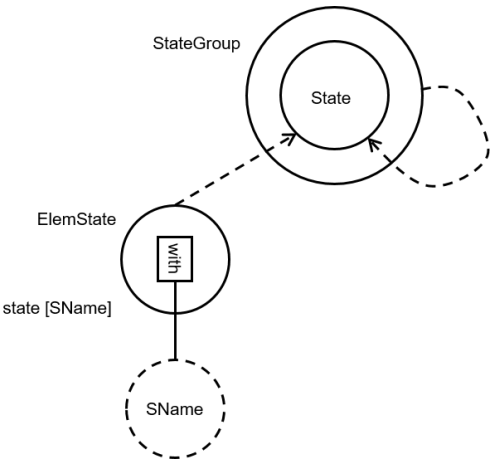

Figure 12. State Decomposition Scheme

7. In Figure 13 we see a simple administration for tracking book inventories. A book is identified by either its title, or, in case of a series, the combination of the title and index number. This scheme is constructed as follows:

$$
\begin{aligned}
&\varnothing_g \\
&\quad + \mathit{Label}(\text{Title}, \text{String}, \text{String}) \\
&\quad + \mathit{Label}(\text{Part}, \text{Natno}, \text{Natno}) \\
&\quad + \mathit{Fact}\big(\text{Compound}, \{\text{tl:Title}, \text{pt:Part}\}, \text{Title 'part' Part}\big) \\
&\quad + \mathit{Spec}\big(\text{Single}, \{\mathit{Title}\}, \text{Title BUT NOT having Part}\big) \\
&\quad + \mathit{Gen}(\text{Product}, \text{Single}) \\
&\quad + \mathit{GenExt}(\text{Product}, \text{Compound}) \\
&\quad + \mathit{Label}(\text{Amount}, \text{Natno}, \text{Natno 'copies'}) \\
&\quad + \mathit{Fact}\big(\text{Stock}, \{\text{prd:Product}, \text{amnt:Amount}\}, \text{Product 'has stock of' Amount}\big) \\
&\quad + \mathit{Constr}\big(C1, \mathit{unique}(\{\mathit{prd}\})\big) \\
&\quad + \mathit{Constr}\big(C2, \mathit{total}(\{\mathit{prd}\})\big)
\end{aligned}
$$

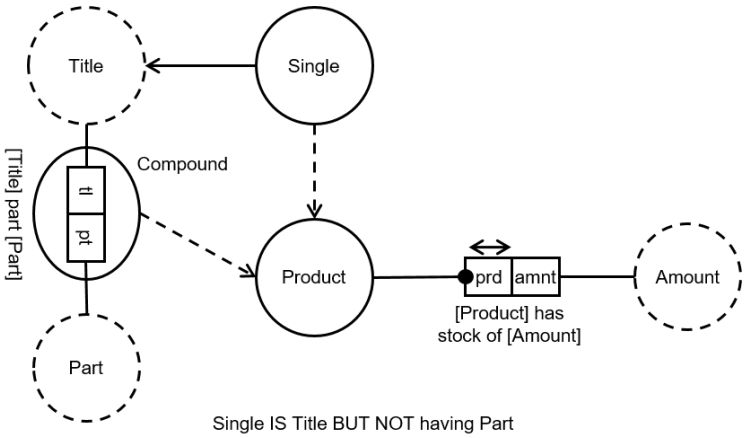

Figure 13. Book inventory scheme

2.4 Information Structure Properties

Structural induction is a powerful mechanism to prove (structural) properties for all information structures. In this section we will introduce a number of predicates over information structures, and then use structural induction to prove their properties. We start with the predicate that states whether an object (name) is being used in an information structure. This predicate is inductively defined as follows:

Definition 1.

$$
\begin{aligned}
&\mathit{Free}(a, \varnothing_g) \\
&\mathit{Free}(a, \text{IS} + X) = \text{Name}(X) \neq a \wedge \mathit{Free}(a, \text{IS})
\end{aligned}
$$

Lemma 1. $\mathit{Free}(a, \text{IS}) \wedge \neg \mathit{Free}(a, \text{IS} + X) \Rightarrow \text{Name}(X) = a$

Proof. Obvious! ◇

Except for the extension of a generalization, each construction step introduces a new object type name:

Lemma 2. *If* $\text{IS} + X$ *is an information structure then* $\mathit{Free}(\text{Name}(X), \text{IS})$ *or* $X = \mathit{GenExt}(\text{Name}(X), G)$ *for some G.*

Proof. Structural induction. In each construction step X of an information structure it is required that $Name(X)$ is a fresh name, except when $X = GenExt(a, G)$ in which case $Name(X)$ should be a known name: $\neg Free(Name(X), IS)$. ◇

Next we will focus on some rules for information structures, and show that their validness is guaranteed when the information structure is introduced stepwise as described by the inductive definition of the information grammar.

2.4.1 Specialization

First we consider relations associated with the specialization hierarchy. The relation $isSpec(IS, a, b)$ states that object type a is a specialization of b in information structure IS. The *isSpec*-relations is defined inductively as follows:

Definition 2.

$$\neg isSpec(\varnothing_g, a, b)$$

$$isSpec(\mathrm{IS} + X, a, b) = \begin{cases} b \in P \vee \exists_{c \in P}[isSpec(\mathrm{IS}, c, b)] & \text{if } X = Spec(N, P, \rho) \text{ and } a = X \\ isSpec(\mathrm{IS}, a, b) & \text{otherwise} \end{cases}$$

An immediate consequence is:

Lemma 3. $isSpec(\mathrm{IS}, a, b) \Rightarrow isSpec(\mathrm{IS} + X, a, b)$

Proof. Structural induction. The induction base is obvious. Next assume $isSpec(IS, a, b) \Rightarrow isSpec(IS + Y, a, b)$ for information structure IS and each Y such that $IS + Y$ is an information structure. Suppose $IS + X$ is an information structure, and let $isSpec(IS + X, a, b)$ for some object types a and b. We have to show that also $isSpec(IS + X + Y, a, b)$ for each extension Y of $IS + X$. From $isSpec(IS + X, a, b)$ we conclude that a is an object type from information structure $IS + X$, and thus $a \neq Y$. Applying the definition of *isSpec* yields the confirmatum. ◇

The function $Top(IS) : Objs(IS) \to Objs(IS)$ assigns to each object type a from IS its top element in the specialization hierarchy.

Definition 3.

$$Top(\mathrm{IS} + X, a) = \begin{cases} a & \text{if } X \text{ is not a specialization and } a = X \\ Top(\mathrm{IS}, b) & \text{if } X = Spec(N, P, \rho) \text{ and } a = X \text{ and } b \in P \\ Top(\mathrm{IS}, a) & \text{otherwise} \end{cases}$$

The predicate *isSpec* is the irreflexive transitive closure of the specialization relation that is built by the introduction of specializations in the information structure. In [7] the specialization hierarchy is defined by the following set of axioms:

PSM6 (asymmetry) $isSpec(IS, a, b) \Rightarrow \neg isSpec(IS, b, a)$
PSM7 (transitivity): $isSpec(IS, a, b) \wedge isSpec(IS, b, c) \Rightarrow isSpec(IS, a, c)$
PSM8 (cohesion): $isSpec(IS, a, b) \Rightarrow isTop(IS, a, c) = isTop(IS, b, c)$
PSM9 (ancestor): $a \neq Top(IS, a) \Rightarrow isSpec(IG, a, Top(IG, a))$

From the inductive definition these axioms can be proven. As example, we only show the proof of **PSM6**:

Lemma 4. ***PSM6*** *(asymmetry):* $isSpec(\mathrm{IS}, a, b) \Rightarrow \neg isSpec(\mathrm{IS}, b, a)$

Proof. Structural induction.
The base step is obvious since $\neg isSpec(\varnothing_g, a, b)$.
Next assume the statement has been proven for IS, consider information structure $IS + X$ for some X. The predicate *isSpec* is the same in IS and $IS + X$ unless $X = Spec(N, P, \rho)$ for some N, P and ρ,

in which case we have the following possibilities for $isSpec(IS+X,a,b)$:

1. a is introduced by X, or, $a = X$. Since a is not an object type in IS, the relation $isSpec(IS,b,a)$ does not hold, and also is not introduced by X.
2. b is introduced by X, or, $b = X$. Then the *isSpec*-relation is not extended for b by X, and thus is also valid in IS. This is a contradiction since b is not an object type in IS.
3. Neither a nor b are introduced by X. In this case, the relation $isSpec(IS,a,b)$, and thus from the induction hypothesis we conclude $\neg\, isSpec(IS,b,a)$. Since in this case the *isSpec*-relation is not extended by X for either a or b, the confirmatum follows. ◇

2.5 Information Structure Semantics

An information structure gets associated history that not only covers its construction process but also the construction of instances. As a consequence, at each moment during its history we can derive the population of each object type.

A population of information structure IS traditionally is defined an assignment of a set of instances to each of its object types such that all construction rules and constraints of IS are satisfied. An instance of an object type may also be seen as an elementary fact generated from the verbalization rule of that object type. The set of instances associated with object type A after history H is denoted $\mathcal{P}(H)(A)$.

Populations are constructed in the information structure history by (stepwise) adding (elementary or compound) fact formulations, starting from the empty population. The effect of a fact formulation is that it requires, if possible, a minimal modification of the population such that the fact formulation becomes a valid sentence. Note that strictly spoken, this may be an indeterministic operation.

Consequently, we may define the history of an information structure as a sequence of (history) addition operators, where each addition either modifies the underlying information structure, or modifies its population. We will also register the points of time when the history addition operator was proclaimed. We assume some underlying ordered domain of points of time starting at point 0.

Inductively, we define an information history based on alphabet Σ and set of domains *Doms* and its semantics as follows, where (1) $\mathcal{R}(H)$ represents the information structure that results after history H, (2) $\mathcal{P}(H)$ — the resulting population and (3) $\mathcal{T}(H)$ — the time of the most recent modification.

1. $\varnothing_h$ is the empty information history. This information history defines the information structure $\varnothing_g$, or, $\mathcal{R}(\varnothing_h) = \varnothing_g$ and the associated empty population $\mathcal{P}(\varnothing_h) = \varnothing_p$. The time of the last addition is set to time 0, or, $\mathcal{T}(\varnothing_h) = 0$.
2. Let H be an information history, let $\mathcal{R}(H)+X$ be a valid information structure, and let $t > \mathcal{T}(H)$ be a new point of time, then $H + \langle t, X\rangle$ is n information history provided population $\mathcal{P}(H)$ satisfies constraint C in case $X = Constr(N,C)$ for some N and C. Furthermore:
   - the associated information structure is: $\mathcal{R}(H + \langle t,X\rangle) = \mathcal{R}(H)+X$;
   - the population $\mathcal{P}(H + \langle t,X\rangle)$ is the extension of $\mathcal{P}(H)$ to the newly introduced object type X if X introduces a new object type, and $\mathcal{P}(H)$ otherwise;
   - $\mathcal{T}(H + \langle t,X\rangle) = t$.
3. Let H be an information history, let ϕ be a valid fact expression from $\mathcal{L}(\mathcal{R}(H))$, and let $t > \mathcal{T}(H)$ be a new point of time, then $H + \langle t, Add(\phi)\rangle$ is an information history. Furthermore:
   - this construct does not change the information structure: $\mathcal{R}(H + \langle t, Add(\phi)\rangle) = \mathcal{R}(H)$;
   - processing $Add(\phi)$ consists of making a minimal modification to $\mathcal{P}(H)$ such that ϕ is a valid fact in $H + Add(\phi)$;
   - $\mathcal{T}(H + \langle t, Add(\phi)\rangle) = t$.

From $\mathcal{P}(H)$ we can define the populations at any point in time as follows:

Definition 4.

$$Pop_{\varnothing_h,t}(A) = \varnothing_p$$
$$Pop_{H+\langle s,X\rangle,t}(A) = \begin{cases} \mathcal{P}(H+\langle s,X\rangle)(A) & \text{if } t \geq s \\ Pop_{H,t}(A) & \text{otherwise} \end{cases}$$

We extend the definition of populations to also cover all information descriptors from $\mathcal{L}(\mathcal{R}(H))$ (for details, see [5]). The result of the evaluation of information descriptor D thus may be denoted $Pop_{H,t}(D)$. When no confusion is likely to occur, we will simply write $Pop_t(D)$ rather than $Pop_{H,t}(D)$.

3 States

We assume that instances of an object type can be in various states. We are looking for a mechanism to introduce states in a natural way in the modeling technique. We assume that a state of an object type can be described by a specific combination of properties for that object type. We will use ORC as our mechanism to describe (combinations of) properties.

3.1 Elementary States

In this section we introduce a conceptual base as a vehicle to describe a complete set of states for an object type. Let $\mathcal{D}$ be a set of information descriptors, and let $\mathcal{X}$ be a set of object types.

> We call $\mathcal{D}$ a conceptual base for $\mathcal{X}$, denoted $cb(\mathcal{D},\mathcal{X})$, if:
>
> **C-1. (non-trivial)** $D \in \mathcal{D} \Rightarrow D$ not structurally empty.
> **C-2. (independent)** $D,E \in \mathcal{D} \wedge D \neq E \Rightarrow D$ and E structurally disjoint
> **C-3. (complete)** $\bigcup_{D\in\mathcal{D}} Pop_t(D) = \bigcup_{X\in\mathcal{X}} Pop_t(X)$ at each moment t.

An information descriptor is structurally empty if at each moment its population is empty. Information descriptors are structurally independent if at no moment they can share an instance in their population. From this definition we conclude that, at each moment, all instances of object types from $\mathcal{X}$ are partitioned by the information descriptors in $\mathcal{D}$. Each conceptual basis is maximal in the following sense:

Lemma 5. $\text{cb}(\mathcal{D},\mathcal{X}) \wedge D \in \mathcal{D} \Rightarrow \neg\text{cb}(\mathcal{D}-D,\mathcal{X})$

Proof. Direct consequence from **C-2** and **C-3**. ◇

A conceptual basis also is maximal in the following sense:

Corollary 1. $\text{cb}(\mathcal{D},\mathcal{X}) \wedge D_0 \notin \mathcal{D} \Rightarrow \neg\text{cb}(\mathcal{D}+D_0,\mathcal{X})$

Conceptual bases can be reduced into smaller conceptual bases in the following way:

Lemma 6. $\text{cb}(\{D_1,D_2,D3,\ldots\},\mathcal{X}) \Rightarrow \text{cb}(\{D_1 \oplus D_2,D_3,\ldots\},\mathcal{X})$

Proof. Suppose $cb(\{D_1,D_2,D3,\ldots\},\mathcal{X})$, then in order to prove $cb(\{D_1 \oplus D_2,D_3,\ldots\},\mathcal{X})$, we first remark that requirements **C-1** and **C-3** are obvious from the definition of the $\oplus$-operator. Considering **C-2**, let $D,E \in \{D_1 \oplus D_2,D_3,\ldots\}$. If both D and E are different from $D_1 \oplus D_2$, the structural independence from D and E follows from $cb(\{D_1,D_2,D3,\ldots\},\mathcal{X})$. So let $D = D_1 \oplus D_2$ and $E \in \{D_3,\ldots\}$. Suppose $Pop_t(D_1 \oplus D_2)$ and $Pop_t(E)$ share an instance at some moment t. Then $Pop_t(E)$ shares an instance with $Pop_t(D_1)$ or $Pop_t(D_2)$, which, in both cases would contradict the assumption $cb(\{D_1,D_2,D3,\ldots\},\mathcal{X})$. We conclude that at no moment $D_1 \oplus D_2$ and E can have a

common instance, or, $D_1 \oplus D_2$ and E are structurally independent. ◇

A set $\mathcal{X}$ of object types may have several conceptual bases, depending on (required) granularity and (modeling) point of view. Some examples are:

1. In Figure 14 there is a partitioning constraint between roles r and q stating that each instance of A should play exactly either role of r and p. Consequently we have the following conceptual bases for object type A:
   (a) $cb(\{Name(A)\},\{A\})$. In this conceptual base there is only one state for each A-instance.
   (b) $cb(\{r,p\},\{A\})$. In this conceptual base A-instances can be in any of two roles, depending on whether they play role r or role p.

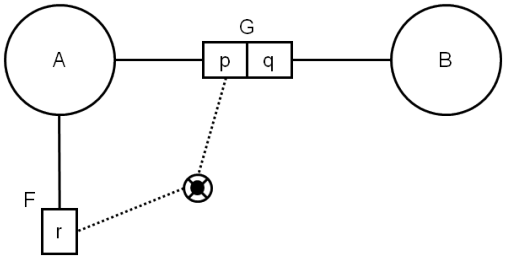

Figure 14. Example 1

2. In Figure 15 the partitioning constraint is not present. Consequently we have the following conceptual bases for A:
   (a) $cb(\{Name(A)\},\{A\})$.
   (b) $cb(\{p \ominus r, r \ominus p, Name(A) \ominus p \ominus r\},\{A\})$. In this case, depending on roles r and p, instances of object type A can be in 3 states.

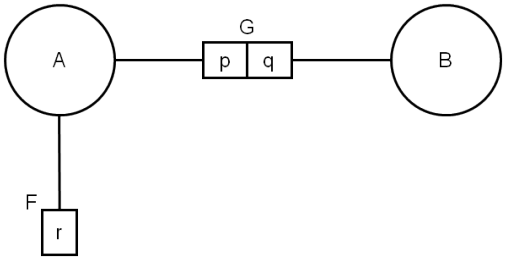

Figure 15. Example 2

If $\mathcal{D}$ is a conceptual base for object type X, then at each moment t each instance x has associated a unique state.

Lemma 7. $\text{cb}(\mathcal{D},\{X\}) \Rightarrow \forall_t \forall_{x \in Pop_t(X)} \exists!_{D \in \mathcal{D}} [x \in Pop_t(D)]$

Proof. Let $x \in Pop_t(X)$ at some moment t. Because of $Pop_t(X) = \bigcup_{D \in \mathcal{D}} Pop_t(D)$ we conclude $x \in Pop_t(D)$ for some $D \in \mathcal{D}$. Because of the partitioning behavior of the descriptors in $\mathcal{D}$ as expressed in **C-2**, we conclude D to be unique. ◇

We will alternatively use $state_t(x) = D$ in place of $x \in Pop_t(D)$, suggesting $state()$ to be a partial function Time $\times$ Instances $\mapsto \mathcal{D}$.

3.2 State Decomposition

We call $\mathcal{D}$ a set of elementary states for object type X if $cb(\mathcal{D}, \{X\})$. At each moment, each instance of X is in precisely one of the elementary states in $\mathcal{D}$. Besides elementary states, we also introduce compound states. Compound states will enable us to focus on transitions in the information structure at various levels of abstraction (granularity). Basically, a compound state is a grouping of (elementary or compound) states, acting as a higher level state. This leads to a hierarchy of states. In this section we describe the requirements for building a composition hierarchy based on the set $\mathcal{D}$ of elementary states.

Let $\mathcal{S}$ be the set of all (elementary and compound) states. The first requirement states that the composition hierarchy is based on the elementary states:

S-1. $\mathcal{D} \subseteq \mathcal{S}$

The hierarchy of states is represented by the relation *SubState*, where $SubState(x,y)$ is representing the fact that x is a substate of y. The hierarchical structure induced by *SubState* is enforced by the following requirements:

S-2. $\neg\, SubState(x,x)$
S-3. $SubState(x,y) \wedge SubState(y,z) \Rightarrow SubState(x,z)$

States without decomposition are elementary. We will use $Elem(x)$ to denote state x being elementary, abbreviating $\neg\, \exists_y [SubState(y,x)]$. The only elementary states come from $\mathcal{D}$:

S-4. $Elem(x) \Leftrightarrow x \in \mathcal{D}$

To express that x is a direct substate of y we use $SubState_1(x,y)$ which is an abbreviation for $SubState(x,y) \wedge \neg\, \exists_z [SubState(x,z) \wedge SubState(z,y)]$. States can have at most one direct superstate:

S-5. $SubState_1(x,y) \wedge SubState_1(x,z) \Rightarrow y = z$

We will refer to $\langle \mathcal{D}, \mathcal{S}, SubState \rangle$ as an S-hierarchy for $\mathcal{D}$ when all requirements **S-1** to **S-5** are fulfilled. Then, with each state $s \in \mathcal{S}$ we associate its underlying set of information descriptors as follows:

$$\delta(s) = \begin{cases} \{D\} & \text{if } s \in \mathcal{D} \\ \{d \in \mathcal{D} \mid SubState(d,s)\} & \text{otherwise} \end{cases}$$

Lemma 8. *$\delta(s)$ is a conceptual basis for $\bigoplus_{D \in \delta(s)} D$, or:* $\text{cb}\left(\delta(s), \bigoplus_{D \in \delta(s)} D\right)$.

Proof. Obvious! ◇

Consequently, the set $\mathcal{S}$ of states induces a partitioning subtype hierarchy on the object type X. The subtype rule for state $s \in \mathcal{S}$ interpreted as a subtype object is $\bigoplus_{D \in \delta(s)} D$ and population $Pop_t(s) = \bigcup_{D \in \delta(s)} (Pop_t(D))$.

Lemma 9. $SubState(x,y) \Rightarrow \delta(x) \subseteq \delta(y)$

Proof. Suppose $SubState(x,y)$, then $y \notin \mathcal{D}$ because of **S-4**. There are 2 possibilities:

1. $x \in \mathcal{D}$: From the definition of the δ-function we conclude $x \in \delta(y)$, and consequently $\delta(x) = \{x\} \subseteq \delta(y)$.
2. $x \notin \mathcal{D}$: Let $x \in \delta(x)$ then we have $x \in \mathcal{D} \wedge SubState(d,x)$ because of $SubState(x,y)$ and **S-3**. So also $x \in \mathcal{D} \wedge SubState(d,y)$ and thus $x \in \delta(y)$. Conclusion: $\delta(x) \subseteq \delta(y)$. ◇

Using the decomposition structure $\mathcal{S}$ of descriptors $\mathcal{D}$, we may associate to each instance the following set of states:

$$states_t(x) = \{ s \in \mathcal{S} \mid SubState(state_t(x), s) \} \cup \{ state_t(x) \}$$

3.3 State Transitions

First we introduce two helpful predicates as follows. Let Φ be a predicate over points of time, then $LE(t,\Phi)$ expresses the fact that in some period before t property Φ holds: $LE(t,\Phi) \triangleq \exists_{s<t} \forall_{s<r<t} [\Phi(r)]$. The expression $RE(t,\Phi)$ states that Φ holds some period after t, or: $RE(t,\Phi) \triangleq \exists_{t<s} \forall_{t \leq r<s} [\Phi(r)]$.

In this section, let $\mathcal{D}$ be a semantic base for object type X, and $\langle \mathcal{D}, \mathcal{S}, SubState \rangle$ be an S-hierarchy. Looking at the history of an information structure we observe the birth of object x at time t in state s as follows:

$$state_t(x) = s \wedge LE(t, x \notin Pop_{t'}(X))$$

The death of instance x in state s at time t is observed as:

$$state_t(x) = s \wedge RE(t, x \notin Pop_{t'}(X))$$

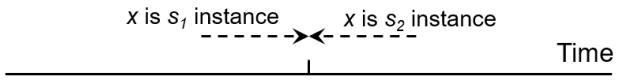

Figure 16. Observing a transition

Looking at the history of an information system we observe a transition from state s_1 to state s_2 at time t, denoted $s_1 \rightsquigarrow s_2$, if (see Figure 16):

$$s_1 \rightsquigarrow s_2 \triangleq \exists_x [LE(t, x \in Pop_{t'}(s_1)) \wedge RE(t, x \in Pop_{t'}(s_2))]$$

Transitions are carried over to superstates (see Figure 17):

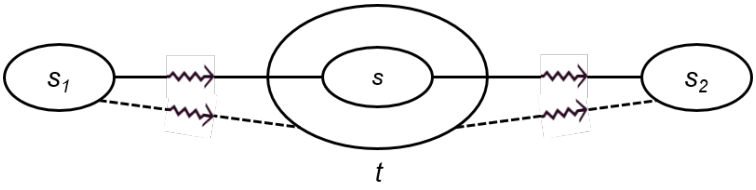

Figure 17. Transitions and superstates

Lemma 10.

1. $SubState(s,t) \wedge s_1 \rightsquigarrow s \Rightarrow s_1 \rightsquigarrow t$
2. $SubState(s,t) \wedge s \rightsquigarrow s_2 \Rightarrow t \rightsquigarrow t_2$

Proof. The main argument is: $x \in Pop_t(s) \wedge SubState(s,t) \Rightarrow x \in Pop_t(t)$. ◇

However, transitions can only originate from elementary states:

Lemma 11.

1. $s_1 \rightsquigarrow s \wedge \neg Elem(s) \Rightarrow \exists_t [SubState(t,s) \wedge s_1 \rightsquigarrow t]$
2. $s \rightsquigarrow s_2 \wedge \neg Elem(s) \Rightarrow \exists_t [SubState(t,s) \wedge t \rightsquigarrow s_2]$

Proof. Obvious, since instances of compound states originate from instances of underlying elementary states. ◇

Using the states $\mathcal{S}$, we further extend the ORM scheme by adding state transitions as a binary relation $\dashrightarrow$ on the states $\mathcal{S}$. The expression $S_1 \dashrightarrow S_2$ is interpreted as: an object in state S_1 may move to state S_2. Furthermore we assume a set $\mathcal{D}_{\mathsf{in}}$ of initial (elementary) states of object type X, and a set $\mathcal{D}_{\mathsf{out}}$ of final (elementary) states. Compound states have (at least) one initial and (at least) one final state: The rules for initial and final states are:

6. $\neg Elem(x) \Rightarrow \exists_y [SubState_1(y,x) \wedge y \in \mathcal{S}_{\mathsf{in}}]$
7. $\neg Elem(x) \Rightarrow \exists_y [SubState_1(y,x) \wedge y \in \mathcal{S}_{\mathsf{out}}]$

Each object type will have an initial state. The instances of an object type may be persistent, in which case the object type may not have a final state. Within a compound state, an initial state has no predecessor, and a final state has no successor:

8. $SubState_1(x,y) \wedge z \dashrightarrow x \wedge \neg SubState_1(z,y) \Rightarrow x \in \mathcal{S}_{\mathsf{in}}$
9. $SubState_1(x,y) \wedge x \dashrightarrow z \wedge \neg SubState_1(z,y) \Rightarrow x \in \mathcal{S}_{\mathsf{out}}$

If a compound state has a successor then this is effectuated by any of its final states. The same holds for the case of a predecessor. This is formalized by the following rules:

10. $u \dashrightarrow v \wedge \neg Elem(u) \wedge \neg Elem(v) \Rightarrow \exists_{x,y} [SubState_1(x,u) \wedge SubState_1(y,v) \wedge x \dashrightarrow y]$
11. $u \dashrightarrow v \wedge \neg Elem(u) \wedge Elem(v) \Rightarrow \exists_x [SubState_1(x,u) \wedge x \dashrightarrow v]$
12. $u \dashrightarrow v \wedge Elem(u) \wedge \neg Elem(v) \Rightarrow \exists_y [SubState_1(y,v) \wedge u \dashrightarrow y]$

Finally, transitions are inherited upward in the hierarchy:

13. $SubState_1(x,u) \wedge SubState_1(y,v) \wedge u \neq v \wedge x \dashrightarrow y \Rightarrow u \dashrightarrow v$

We call a (compound) state x a *start state* of compound state y if x is an initial state and also a direct substate of y:

$$StartState(x,y) \triangleq x \in \mathcal{D}_{\mathsf{in}} \wedge SubState_1(x,y)$$

We call x a birth state if it is an initial state but not the start state of another state:

$$BirthState(x) \triangleq x \in \mathcal{D}_{\mathsf{in}} \wedge \neg \exists_y [SubState_1(x,y)]$$

If x is a birth state, then we also write:

$$\omega \dashrightarrow x$$

where ω is virtual (source) state. If x is a death state, then we also write:

$$x \dashrightarrow \Omega$$

where Ω also is a virtual (sink) state. Analogously we introduce a final state and a death state:

$$FinalState(x,y) \triangleq x \in \mathcal{D}_{\mathsf{out}} \wedge SubState_1(x,y)$$
$$DeathState(x) \triangleq x \in \mathcal{D}_{\mathsf{out}} \wedge \neg \exists_y [SubState_1(x,y)]$$

We call the structure $\mathcal{B}(X) = \langle \mathcal{S}, SubState_1, \mathcal{D}, \dashrightarrow, \mathcal{D}_{\mathsf{in}}, \mathcal{D}_{\mathsf{out}} \rangle$ a behavioral description of object type X. Note that an object type may have more than one behavioral description, but this will be generally not the case in practice.

3.4 Example

We discuss an example taken from [23]. In this example the life of a pop band is described. According to some basic fact types, we may distinguish the elementary states $D_1, \ldots, D_6$. After setting up the band (D_1) the band gets a name (D_2). Then the band repeatedly performs actions like writing a song (D_3) followed by recording that song (D_4), or producing a recording (D_5). Then the band may stop its activities (D_6). The resulting extended ORM scheme is shown in Figure 18, focusing at the life model only. We see two compound states, S_1 that comprises the band foundation process, and S_2 covering the active life of the band. Formally, we have:

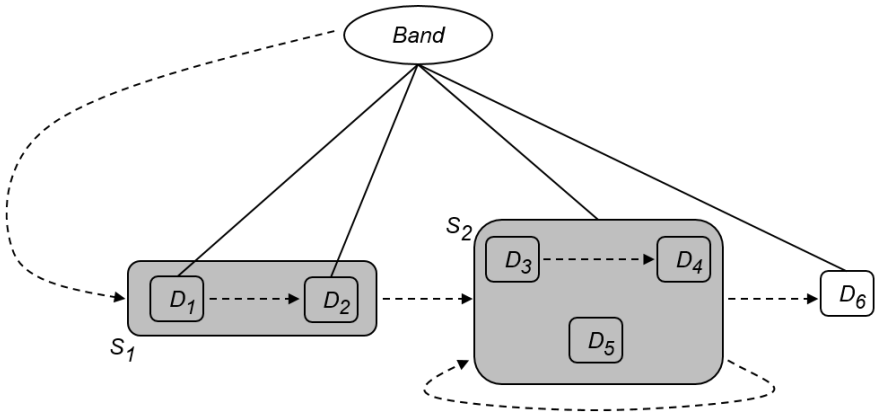

Figure 18. The object life model of a band. Source: [23]

- $\mathcal{D} = \{D_1, \ldots, D_6\}$
- $\mathcal{S} = \{D_1, \ldots, D_6, S_1, S_2\}$
- $SubState_1 = \{(D_1,S_1),(D_2,S_1),(D_3,S_2),(D_4,S_2),(D_5,S_2)\}$
- $\dashrightarrow = \{(D_1,D_2),(D_2,D_3),(D_2,D_5),(S_1,S_2),(D_3,D_4),(D_4,D_3),(D_4,D_5),(D_5,D_3),$ $(D_5,D_5),(S_2,S_2),(D_4,D_6),(D_5,D_6),(S_2,D_6)\}$
- $\mathcal{D}_{\mathsf{in}} = \{D_1, D_3, D_5\}$
- $\mathcal{D}_{\mathsf{out}} = \{D_2, D_4, D_5, D_6\}$

Using the decomposition mechanism, it is possible to see the life of a band at various levels of abstraction. For example, in Figure 19 we have a high level view on the band's life.

4 Conclusion

Modern information and communication technology is having a big impact on all aspects of daily life. For organizations, a consequence may be that their information systems have to adapt to new situations, leading to rapid changes in the underlying information structure. Besides, the organizational paradigms are intertwined with the structure of the organization's information systems [31]. New developments also necessitate the making of new relationships with data from other companies to address various aspects of data science. Consequently, the question of proper

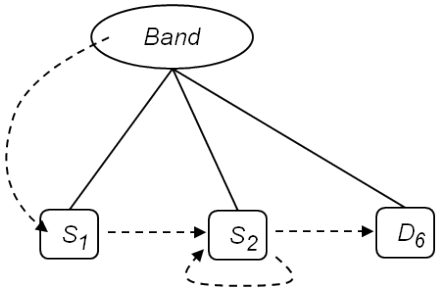

Figure 19. The simplified object life model of a band. Source: [23]

data (information) description representing relevant phenomena in real life becomes more and more essential. An important issue is to be able to describe both static and dynamic aspects, preferably within a single formalism.

This paper has therefore presented a mechanism to use the information structure for describing information structure states and their relationships in time. In so doing, an integrated modeling technique covering both static and dynamic aspects of application domains is obtained. The inductive definition that we advocated in this paper may be used for more in-depth studies on properties of data models. As a result, we propose an extension of ORM with the behavioral description of object types to allow decomposition to master schema complexity.

Secondly, we have provided a foundation for improving system dynamics conceptualization by introducing the inductive definition for data modeling techniques in general and for ORM in particular. In the context of System Dynamics state decompositions may be used to develop more complex applications. A further point of research is to further explore this foundation both theoretically and pragmatically.

Acknowledgements

We thank the anonymous reviewers for their careful reading of our manuscript and their many insightful comments and suggestions.

References

[1] D. Gupta and N. Prakash, “Engineering Methods from Method Requirements Specifications,” Requirements Engineering, vol. 6, no. 3, pp. 135–160, 2001. [Online]. Available: http://dx.doi.org/10.1007/s007660170001

[2] I. Vessey and S. A. Conger, “Requirements Specification: Learning Object, Process, and Data Methodologies,” Commun. ACM, vol. 37, no. 5, pp. 102–113, 1994. [Online]. Available: http://dx.doi.org/10.1145/175290.175305

[3] R. Wieringa, “Combining Static and Dynamic Modelling Methods: A Comparison of Four Methods,” The Computer Journal, vol. 38, no. 1, pp. 17–30, November 1995. [Online]. Available: http://dx.doi.org/10.1093/comjnl/38.1.17

[4] Y. Fu, Q. Zhang, and C. Wang, “Utilizing UML and System Dynamics to Optimize the Eco-Environmental Impacts Evaluation and Prediction Models.” Information and Automation, pp. 238–243, June 2008. [Online]. Available: http://dx.doi.org/10.1109/ICINFA.2008.4608003

[5] A. t. Hofstede, H. Proper, and T. v. d. Weide, “Formal Definition of a Conceptual Language for the Description and Manipulation of Information Models,” Information Systems, vol. 18, no. 7, pp. 489–523, October 1993. [Online]. Available: http://dx.doi.org/10.1016/0306-4379(93)90004-K

[6] T. Halpin and M. Curland, “Automated Verbalization for ORM 2,” in On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops. Heidelberg: Springer LNCS, 2006, vol. 4278, pp. 1181–1190. [Online]. Available: http://dx.doi.org/10.1007/11915072_21

[7] A. t. Hofstede and T. v. d. Weide, “Expressiveness in Conceptual Data Modelling,” Data & Knowledge Engineering, vol. 10, no. 1, pp. 65–100, February 1993. [Online]. Available: http://dx.doi.org/10.1016/0169-023X(93)90020-P

[8] J. Wintraecken, NIAM Information Analysis Method: Theory and Practice. Norwell, MA, USA: Kluwer Academic Publishers, 1990. [Online]. Available: http://dx.doi.org/10.1007/978-94-009-0451-4

[9] T. Halpin and G. Wagner, “Modeling Reactive Behavior in ORM,” in Conceptual Modeling, Proc. of the 22nd ER2003 Conference. Chicago: Springer LNCS, October 2003, vol. 2813, pp. 567–569. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-39648-2_45

[10] T. Halpin, ‘Object-Role Modeling (ORM/NIAM)’, Handbook on Architectures of Information Systems. Springer Berlin Heidelberg, 1998. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-03526-9_4

[11] M. Jarrar, “Towards Automated Reasoning on ORM Schemes-Mapping ORM into DLR-idf description logic,” in Proceedings of the 26th international conference on Conceptual modeling. Auckland, New Zealand: Springer-Verlag Berlin / Heidelberg, November 2007, vol. 4801, pp. 181–197. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-75563-0_14

[12] T. Halpin, A Logical Analysis of Information Systems: Static Aspects of the Data-Oriented Perspective. University of Queensland: PhD dissertation, 1989.

[13] P. v. Bommel, A. t. Hofstede, and T. v. d. Weide, “Semantics and Verification of Object-Role Models,” Information Systems, vol. 16, no. 5, pp. 471–495, 1991. [Online]. Available: http://dx.doi.org/10.1016/0306-4379(91)90037-A

[14] T. v. d. Weide, A. t. Hofstede, and P. v. Bommel, “Uniquest: Determining the Semantics of Complex Uniqueness Constraints,” The Computer Journal, vol. 35, no. 2, pp. 148–156, Apr. 1992. [Online]. Available: http://dx.doi.org/10.1093/comjnl/35.2.148

[15] G. Verheijen and J. v. Bekkum, “NIAM: an Information Analysis Method,” in Information Systems Design Methodologies: A Comparative Review, T. Olle, H. Sol, and A. Verrijn-Stuart, Eds. Amsterdam, The Netherlands: North-Holland/IFIP, 1982, pp. 537–590.

[16] C. Leung and G. Nijssen, “From a NIAM Conceptual Schema into the Optimal SQL Relational Database Schema,” The Australian Computer Journal, vol. 19, no. 2, pp. 69–75, May 1987.

[17] E. Falkenberg, A. Verrijn–Stuart, K. Voss, W. Hesse, P. Lindgreen, B. Nilsson, J. Oei, C. Rolland, and R. a. Stamper, Eds., A Framework of Information Systems Concepts. Laxenburg, Austria, EU: IFIP WG 8.1 Task Group FRISCO, IFIP, 1998.

[18] O. De Troyer, “RIDL*: A Tool for the Computer-Assisted Engineering of Large Databases in the Presence of Integrity Constraints,” in Proceedings of the ACM SIGMOD International Conference on Management of Data, Portland, Oregon, 1989, pp. 418–429. [Online]. Available: http://dx.doi.org/10.1145/67544.66965

[19] O. De Troyer, R. Meersman, and P. Verlinden, “RIDL* on the CRIS Case: A Workbench for NIAM,” in Computerized Assistance during the Information Systems Life Cycle, T. Olle, A. Verrijn-Stuart, and L. Bhabuta, Eds. Amsterdam, The Netherlands: North-Holland/IFIP, 1988, pp. 375–459.

[20] E. Falkenberg and T. v. d. Weide, “The TOP Model: A Metamodel for Conservative Data Bases,” Department of Information Systems, University of Nijmegen, The Netherlands, Research Report, 1988.

[21] P. Frederiks and T. v. d. Weide, “Information Modeling: the Process and the Required Competencies of its Participants,” Data & Knowledge Engineering, pp. 123–134, 2004. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-27779-8_11

[22] M. Hanani and P. Shoval, “A Combined Methodology for Information Systems Analysis and Design Based on ISAC and NIAM,” Information Systems, vol. 11, no. 3, pp. 245–253, 1986. [Online]. Available: http://dx.doi.org/10.1016/0306-4379(86)90032-3

[23] P. Frederiks and T. v. d. Weide, “Deriving and Paraphrasing Information Grammars Using Object-Oriented Analysis Models,” Acta Informatica, vol. 38, pp. 437–488, June 2002. [Online]. Available: http://dx.doi.org/10.1007/s002360200083

[24] F. Tulinayo, S. Hoppenbrouwers, and H. Proper, “Integrating System Dynamics with Object-Role Modeling,” in IFIP Working Conference on The Practice of Enterprise Modeling. Springer, 2008, pp. 77–85. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89218-2_6

[25] G. Bakema, J. Zwart, and H. v. d. Lek, “Fully Communication Oriented NIAM,” NIAM-ISDM working conference 1994, 1994.

[26] D. Mayer, Theory of Relational Databases. Computer Science Press, 1990.

[27] A. t. Hofstede, H. E. Proper, and T. v. d. Weide, “Exploiting Fact Verbalisation in Conceptual Information Modelling,” Information Systems, vol. 22, no. 6/7, pp. 349–385, September 1997. [Online]. Available: http://dx.doi.org/10.1016/S0306-4379(97)00022-7

[28] A. t. Hofstede and T. v. d. Weide, “Deriving Identity from Extensionality,” International Journal of Software Engineering and Knowledge Engineering, vol. 8, no. 2, pp. 189–221, 1997. [Online]. Available: http://dx.doi.org/10.1142/S0218194098000121

[29] H.-J. Schek and M. H. Scholl, “The Relational Model with Relation-Valued Attributes,” Information systems, vol. 11, no. 2, pp. 137–147, 1986. [Online]. Available: http://dx.doi.org/10.1016/0306-4379(86)90003-7

[30] I. Robinson, J. Webber, J. Webber, and E. Eifrem, Graph Databases. O’Reilly Media, 2013. [Online]. Available: http://books.google.nl/books?id=RTvAmQEACAAJ

[31] E. Brynjolfsson and H. Mendelson, “Information Systems and the Organization of Modern Enterprise,” Journal of Organizational Computing and Electronic Commerce, vol. 3, no. 3, pp. 245–255, 1993. [Online]. Available: http://dx.doi.org/10.1080/10919399309540203