Antony Bryant and Lesley Semmens (Eds)

Methods Integration

Proceedings of the Methods Integration Workshop, Leeds, 25-26 March 1996

Paper:

Data Modelling in ZIM
V. Kasurinen and K. Sere

Published in collaboration with the British Computer Society

©Copyright in this paper belongs to the author(s)
Data Modelling in ZIM

Virpi Kasurinen          Kaisa Sere
Department of Computer Science and Applied Mathematics, University of Kuopio
Kuopio, Finland,

Abstract

Data modelling is an important part of the software development process. In this paper we propose the ZIM (Z In data Modelling) method to do data modelling. ZIM is an approach where structured or object-oriented requirements specifications given as entity-relationship diagrams are transformed to formal Z specifications. Techniques from the world of relational data modelling are used to guarantee that the generated Z specification is well structured and in a certain sense minimal. The ZIM method is exemplified via a case study.

1 Introduction

In this paper we present a new approach, ZIM, to do data modelling. In ZIM, structured, object-oriented, and relational methods are used when creating formal Z specifications. The ZIM method uses an entity-relationship diagram as the starting point. The main goal in ZIM is to transform such a diagram to a Z specification that fulfills the following:

- Unnormalized data is removed from the objects (i.e. entities and relationships).
- Redundant data representations are removed.
- The specification reflects the structure of each object.
- Every object is explicitly associated with all its constraints in the specification.

Furthermore, the transformation from an entity-relationship diagram to a Z specification should be such that it can be performed mechanically within a CASE-tool.

Previously similar integrations have been reported between formal methods and structured methods, see Josephs et al. [7, 8], Semmens et al. [14, 15, 16], Gimbayashi [4], Polack et al. [12, 13]. The main difference in our approach compared to these is that in order to meet the criteria above we use theory from relational data modelling to remove redundancy and to normalize data objects. Furthermore, we pay special attention to the structure of every object by keeping tightly together an object and all its associated constraints.

There also exists integrations between formal methods and relational methods for database design [1, 10]. These are, however, mainly used to specify database applications. This can also be done following the ZIM approach.

Overview. We proceed as follows. In section 2, we will shortly present the semantics of entity-relationship diagrams. The formal language Z [18] that is used in ZIM is assumed to be familiar. The ZIM method is put forward in section 3. In section 4 we give an extension to our method where some additional features of relational data modelling are taken into account. We show the practical applicability of our method via a case study in section 5. We end in section 6 with some concluding remarks.
2 Entity relationship models

The entity-relationship model (ER model) is a semantic data model that is used to model data as entities and relationships between these. The result of entity-relationship modelling is called an entity-relationship diagram (ER diagram) first introduced by Chen [2]. Later some extensions were made to Chen’s ER model for example by Engels et al. [3] and Yourdon [19]. The ER model has its origins in structured methods. However, the ER model also forms a basis for object-oriented and relational data modelling methods.

Let us start by briefly describing the basic notions of ER model, (i.e. entities and relationships). In Figure 1, a general form of an ER diagram is shown.

![ER Diagram](image)

**Figure 1:** An ER diagram.

**Entities** An entity is something material or nonmaterial that exists in the real world. There are two types of entities, strong entities and weak entities. Strong entities, also known as regular entities, exist independently of other entities. Every weak entity depends on some strong entity. Entities have attributes that describe their properties. It is assumed that there is a distinct attribute list, or the attributes are presented in the ER diagram. Attribute values are grouped into domains. At least one attribute (known as a key) should uniquely identify instances of every entity. Instances form an entity set.

Each weak entity can inherit all of the properties of its corresponding strong entity. Furthermore, each weak entity may add its own properties. However, an instance of a weak entity is identified by the key of the corresponding strong entity in combination possibly with its own attributes.

**Relationships** Relationships represent the interaction between two or more entities. The participating entities can be paired using one-to-one, many-to-one, or many-to-many mapping constraints. If the constraint is many-to-many we talk about relational relationships, otherwise the relationship is functional. Each entity may participate or has to participate in a relationship, i.e. an entity is optional or compulsory in a relationship. Cardinality aspect is formalized by giving it as an ordered pair \((x, y)\) next to the appropriate entity in a diagram. Here \(x \in \{0, 1\}\) with \(x = 0\) indicating that each instance of the entity may participate
in the relationship (i.e. the entity is optional in the relationship) and \(x = 1\) indicating that each instance of the entity has to participate in the relationship (i.e. the entity is compulsory in the relationship). Moreover, \(y \in \{1, N\}\) where \(y = 1\) denotes that an entity participates in at most one relationship and \(y = N\) denotes that an entity participates in one relationship or in many relationships.

Functional and relational relationships are called *standard relationships*. There can be a standard relationship about which some information is maintained. This information is carried by an entity that is called an *assigner* [17]. Standard relationships are identified by the identifiers of the connecting entities.

ISA- and ID-relationships are two functional relationships called *existential relationships*. Both relationships are used for representing hierarchical structures between strong entities and weak entities. An ISA-relationship is used for modelling the partitioning of a set of strong entities (a superclass) into subsets of weak entities (subclasses) [16] particularly familiar from the world of object-oriented design.

An ID-relationship (known as a part-of relationship) is always between exactly two entities, a strong entity and a weak entity. The weak entity is identified by an unique identifier, which is a concatenation of the key of corresponding strong entity and its own attributes. More on these topics can be found elsewhere [9].

**Removing redundancy** Often ER diagrams contain information that is redundant. When such diagrams are transformed into \(Z\), this redundancy is reflected in the \(Z\) specification. Therefore, it is advisable to first remove the redundant information.

If an entity has an attribute whose value can be derived by a relationship from an attribute of another entity, this derivable attribute is removed as shown in Figure 2. Let us assume that in Figure 2 a) the entity A has an attribute A2 that is derivable via the relationship R2. In Figure 2 b) this attribute is removed.

Another kind of redundancy can be found from relationships, whenever one relationship is created from another by composition. This is also illustrated in Figure 2 where the relationship R3 in a) is a composition of R1 and R2 \((R3 = R1 \circ R2)\), and hence can be removed.

Figure 2: Structural redundancy elimination.
3 The ZIM method

The ZIM method takes an ER diagram, where redundancy has been removed as described above, and produces a Z specification from it. The method has three phases as follows:

**Entity type normalization** The attributes have domains, that should be defined before an entity is formed. All non-key attributes should be dependent on the key attribute of an entity. If there are functional dependencies between non-key attributes they are removed.

**Object set generation** Schemas that illustrate sets of instances of objects (i.e. entities and relationships) are generated. Constraints associated with the set are considered. The definition of a set links each object to a unique key.

**Forming complete state** Finally, all classes are gathered into one complete state.

Let us look at these phases.

3.1 Entities and entity sets in ZIM

The entities are transformed into Z schemas. First, we look at the dependencies between the different attributes within an entity. If such dependencies are found, they are first removed. Then the domains for the attributes are defined. The entity sets are defined in two steps. First, we generate a specification for the non-key attributes of an entity, and secondly the entity set itself is defined by identifying the possible instances of the entity.

**Entity type normalization** In ZIM, the interdependencies between attributes are removed using entity type normalization. This technique, which is here described only through an example, is used in relational modelling when removing functional dependencies between attributes leaving only the dependencies on the key attribute [3].

The entity type normalization is illustrated in Figure 3. There the entity A has four attributes, A1, A2, A3, A4. The attribute A1 is the key attribute, and the attribute A2 is dependent on it. Moreover, attribute A4 is dependent on A3. This dependency is removed by splitting the entity A into two entities A and A' (Figure 3b). The attribute A3 becomes the key of the entity A'. Cardinality of A' is (1,N) and of A it is (1,1), i.e. both entities in the functional relationship are compulsory, and A is related by several instances of A'.

![Diagram](a) unnormalized entity type  
![Diagram](b) normalized entity type

**Figure 3**: Entity type normalization.
The domains of attributes Each entity has a set of attributes. Every attribute has an associated type, a domain name. These will directly correspond to Z types:

\[\text{DOMAIN}\]
\[\text{DOMAIN} :: value_1 | value_2 | \ldots | value_n\]

Domains can also be specified using some other domain, or domains can have constraints:

\[\text{DOMAIN} == \text{ANOTHER\_DOMAIN}\]
\[\text{DOMAIN} == \{ \text{dom} : \text{ANOTHER\_DOMAIN} | \text{constraints} \}\]

In the relational data model, the attributes are atomic, single-valued, and mandatory or optional [5]. If a value of an attribute is structured, or it has several values, the attribute should be changed to an entity [9]. However, in ZIM an attribute is allowed to be structured, as well. Therefore, the domain of a structured attribute is specified as a schema type:

\[\text{DOMAIN} \triangleq [\text{attribute}_1 : \text{DOMAIN}_1; \ldots; \text{attribute}_N : \text{DOMAIN}_N]\]

Moreover, a non-key attribute is allowed to have several (or null) values. In this case, its type is defined using the power set construction \( P \text{DOMAIN} \).

Non-key attributes of an entity Let us now consider an entity with \( N \) attributes, \( \text{attribute}_1, \ldots, \text{attribute}_N \), plus the key attribute. The non-key attributes of an entity are combined into a record schema as follows:

\[\text{Entity} \triangleq [\text{attribute}_1 : \text{DOMAIN}_1; \ldots; \text{attribute}_N : \text{DOMAIN}_N]\]

A similar definition also identifies the assigner in an associative relationship. Each strong entity that participates in an ISA-relationship is formalized by the partitioned weak entities. This is shown later in section 3.3.

Entity set Let us now consider a set of instances of the entity, i.e. the entity set. Let \( \text{ENTITY\_ID} \) be the domain of the key attribute. The possible instances of every strong entity are now formalized in Z as the schema

\[
\begin{align*}
\text{EntityDS} & \\
\text{Entities} : \text{ENTITY\_ID} \mapsto \text{Entity} \\
\text{constraints}
\end{align*}
\]

where the partial function \((\mapsto)\) guarantees that each entity has its own identifier but they can have similar non-key attributes. The predicate specifies the constraints over the attributes of existing instances.

### 3.2 Standard relationships in ZIM

Let us now consider the standard relationships, relational, and functional.

A relational relationship A relational relationship between two entities is defined as a Z relation \((\mapsto)\) between the domains of the key attributes of the two entities:

\[
\begin{align*}
\text{RelationshipDS} & \\
\text{DomainEntityDS} \cap \text{RangeEntityDS} \\
\text{Relationships} : \text{DOMAIN\_ID} \mapsto \text{RANGE\_ID} \\
\text{dom Relationships} & \subseteq \text{dom DomainEntities} \\
\text{ran Relationships} & \subseteq \text{dom RangeEntities}
\end{align*}
\]

The participating entity set schemas (\( \text{DomainEntityDS}, \text{RangeEntityDS} \)) are included in the definition part of the schema. The known instances of both entities are used to constrain the defined relationship. Both entity sets are defined as \( \text{ENTITYDS} \) in the previous section. As it is now, the subset sign \((\subseteq)\) illustrates that both entities (\( \text{DomainEntities} \) and \( \text{RangeEntities} \)) are optional in the relationship. In case an entity is compulsory in the relationship, the corresponding subset sign is replaced with equality \((=)\).
A functional relationship  A functional relationship is defined as the relational relationship; the only difference is in the definition part. There are two cases:

- many-to-one, illustrated by the partial function symbol ($\rightarrow$) as follows
  
  \[
  \text{Relationships} : \text{DOMAIN\_ID} \rightarrow \text{RANGE\_ID}
  \]

- one-to-one, illustrated by the injective partial function symbol ($\rightarrow$) as follows
  
  \[
  \text{Relationships} : \text{DOMAIN\_ID} \rightarrow \text{RANGE\_ID}
  \]

A standard relationship with an assigner  A relationship between two entities is identified by the key domains of these entities, $\text{DOMAIN\_ID}$ and $\text{RANGE\_ID}$. If some information is carried by the relationship, this information is modelled by an assigner [6]. This is captured in the following Z schema:

\[
\begin{array}{c}
\text{AssociativeDS} \\
\text{DomainEntityDS; RangeEntityDS} \\
\text{Associatives} : (\text{DOMAIN\_ID} \rightarrow \text{RANGE\_ID}) \rightarrow \text{Entity} \\
\text{dom}(\bigcup(\text{dom Associatives})) \subseteq \text{dom DomainEntities} \\
\text{ran}(\bigcup(\text{dom Associatives})) \subseteq \text{dom RangeEntities}
\end{array}
\]

where $\text{Entity}$ refers to an assigner that is related to the relationship. If the relationship between entities is functional, the mapping between entities is either:

- many-to-one $(\text{DOMAIN\_ID} \rightarrow \text{RANGE\_ID})$ or
- one-to-one $(\text{DOMAIN\_ID} \rightarrow \text{RANGE\_ID})$

### 3.3 Existential relationships in ZIM

Next, the existential relationships, ISA and ID, are considered.

An ISA-relationship schema  An ISA-relationship partitions the instances of a strong object into several weak objects. However, one instance of the strong object corresponds to one instance of the weak object. Every strong object participating in this relationship is identified by the key of the strong object. Let $\text{STRONG\_ID}$ be the key domain of the strong object. Then an ISA-relationship corresponds to the Z schema:

\[
\begin{array}{c}
\text{ISA\_RelationshipDS} \\
\text{WeakObjects}_1 : \text{STRONG\_ID} \rightarrow \text{WeakObject}_1 \\
\vdots \\
\text{WeakObjects}_n : \text{STRONG\_ID} \rightarrow \text{WeakObject}_n \\
\text{StrongObjects} : \text{STRONG\_ID} \rightarrow \text{StrongObject} \\
\langle \text{dom WeakObjects}_1, \ldots, \text{dom WeakObjects}_n \rangle \\
\text{partition} \text{dom StrongObjects}
\end{array}
\]

where the key attribute identifies exactly one strong object instance.
**An ID-relationship schema** In the ID-relationship we have one strong entity identified by its key attribute `STRONG_ID`, and a weak entity. The domain of a weak entity, `W_ENTITY`, is modelled by a non-key `Entity`. It is identified by the key attribute inherited from the strong entity in combination with an own attribute, `WEAK_ID`.

\[
\begin{align*}
&\text{ID\_RelationshipDS} \\
&\quad \text{StrongEntityDS} \\
&\quad \text{WeakEntities : (WEAK\_ID \leftrightarrow STRONG\_ID) \leftrightarrow W\_ENTITY} \\
&\quad \text{ran(\{\text{dom (WeakEntities)}\}) \subseteq \text{dom (StrongEntities)}}
\end{align*}
\]

### 3.4 Complete state

The complete state schema combines the defined entities and relationships that are included in the definition part of it. In the object-oriented approach there is a concept of a *metaclass*, i.e., a class whose instances are object sets (entity sets or relationship sets). The complete state corresponds to a metaclass. The possible constraints are applicable to the set of instances as a whole [11], and these are added to the predicate part of the complete state schema.

### 4 Foreign keys

Relationships are represented in the relational models by *foreign keys* [1]. Constraints on foreign keys are used instead of constraints on whether an entity is compulsory or optional in a relationship. In ZIM, either the domain or the range entity has to be compulsory in the relationship when the foreign key representation is used.

Let us illustrate the foreign key representation. A functional relationship where the participation of the domain entity is compulsory can be removed by adding the key attribute of the range entity into the attributes of the domain entity. The added attribute is called the foreign key. Hence, the value of a foreign key is always equal to the value of the key of the range entity in the removed relationship. This is illustrated in Figure 4.

#### a) functional relationship

![Functional Relationship Diagram](image)

#### b) removed relationship by the foreign key

![Removed Relationship Diagram](image)

**Figure 4**: Transforming a relationship into a foreign key.

Let the entities A and B be defined as follows:

**Methods Integration Workshop, 1996**
Let the relationship \( R \) have the following definition:

\[
\begin{align*}
  R &: A_{\text{DS}}; B_{\text{DS}} \\
  \text{Relationships} &: B_{\text{ID}} \rightarrow A_{\text{ID}} \\
  \text{dom Relationships} &= \text{dom } B_{\text{Entities}} \\
  \text{ran Relationships} &\subseteq \text{dom } A_{\text{Entities}}
\end{align*}
\]

where the relationship is functional, and furthermore, \( B \) is a compulsory participant. Instead of this relationship the definition of the \( B \) entities is changed by mapping each \( B \) entity to the key of the \( A \) entity that is called a foreign key. After redefinition the set \( B_{\text{DS}} \) is as follows:

\[
\begin{align*}
  B_{\text{DS}} \\
  A_{\text{DS}} \\
  B_{\text{Entities}} &: (B_{\text{ID}} \rightarrow B) \rightarrow A_{\text{ID}} \\
  \text{ran } B_{\text{Entities}} &\subseteq \text{dom } A_{\text{Entities}}
\end{align*}
\]

Observe that the injective partial function \( (\rightarrow) \) is used when one-to-one relationship is removed. If the mapping constraint between the domain and the range entity is many-to-one, the general form of a schema that incorporates a foreign key is:

\[
\begin{align*}
  \text{DomainEntity}_{\text{FK}_{-}\text{DS}} \\
  \text{RangeEntity}_{\text{DS}} \\
  \text{DomainEntities} &: (\text{DOMAIN}_{\text{ID}} \rightarrow \text{DomainEntity}) \rightarrow \text{RANGE}_{\text{ID}} \\
  \text{ran DomainEntities} &\subseteq \text{dom } \text{RangeEntities}
\end{align*}
\]

where partial function \( (\rightarrow) \) is used to mapping the domain of a foreign key \( \text{RANGE}_{\text{ID}} \).

If the range entity is compulsory in the relationship, and the mapping constraint between the range and the domain entity is many-to-one, the relationship is removed by following foreign key construction:

\[
\begin{align*}
  \text{RangeEntity}_{\text{FK}_{-}\text{DS}} \\
  \text{DomainEntity}_{\text{DS}} \\
  \text{RangeEntities} &: (\text{RANGE}_{\text{ID}} \rightarrow \text{RangeEntity}) \rightarrow P_1 \text{DOMAIN}_{\text{ID}} \\
  \bigcup (\text{ran RangeEntities}) &\subseteq \text{dom } \text{DomainEntities}
\end{align*}
\]

If both entities are optional in the relationship, null values should be considered when foreign keys are used to replace functional or relational relationships between entities. There is no universally accepted approach to null (missing attribute) values. The null values can be explicitly included via the foreign key approach as done by Barros [1]. He has used his FOR_KEY operators, and he has specified one null constant for each attribute domain.
5 An example: Access Security System

In secure computer systems access to information, user/password validation at login, the creation of new data entities and the maintenance of security data need to be controlled. An ER diagram and the associated Z specification for this problem following the method of Semmens et al. is given in [15]. The ER diagram for the system is given in Figure 5.

The attribute list for assigners and entities is as follows:

- USER=@$Id$:USER_ID; Pw:PASSWORD; Clearance:SECURITYLEVEL
- JOB=@$Id$:JOB_ID; Clearance:SECURITYLEVEL
- OBJECTTYPE=@$Id$:OBJECT_ID
- DATAOBJECT=@$Id$:DATA_ID; Classification:SECURITYLEVEL
- JOBPERFORMANCE=Status:STATUS
- ALLOWEDACCESS=AccessMode:ACCESSMODE

where @ denotes the key attributes.

There is no need for redundancy elimination, nor entity type normalization. Hence, we can directly give a Z formalization of the ER diagram in Figure 5 following the ZIM method.

The attribute domains The domains of Access Security Systems do not have any constraints [15]. The domains of attributes are derived from the attribute list above:

\[
[\text{USER\_ID, PASSWORD, SECURITYLEVEL, JOB\_ID, OBJECT\_ID, DATA\_ID, ACCESSMODE}]
\]

\[
\text{STATUS} ::= \text{LoggedIn} \mid \text{LoggedOut}
\]

The entities as Z types There are only strong entities in our system. Hence, it is straightforward to give the formalization of the non-key attributes:

\[
\text{USER} \cong [\text{Pw} : \text{PASSWORD}; \text{Clearance} : \text{SECURITYLEVEL}]
\]

\[
\text{JOB} \cong [\text{Clearance} : \text{SECURITYLEVEL}]
\]

\[
\text{DATAOBJECT} \cong [\text{Classification} : \text{SECURITYLEVEL}]
\]
**Entity sets** The domains of the key attributes now identify the instances of the entities:

- \( UserDS \equiv [Users : USER_ID \leftrightarrow USER] \)
- \( JobDS \equiv [Jobs : JOB_ID \leftrightarrow JOB] \)
- \( ObjectTypeDS \equiv [ObjectTypes : OBJECT_ID \leftrightarrow OBJECTTYPE] \)
- \( DataObjectDS \equiv [DataObjects : DATA_ID \leftrightarrow DATAOBJECT] \)

Observe that OBJECTTYPE entity has no non-key attributes. To reflect this, we define instances of this entity using the power set construction.

**Instances of standard relationships** There is one relational relationship, CAN_CREATE, where the entity OBJECTTYPE is compulsory, and the entity JOB is optional:

\[
\text{CanCreateDS} \\
\text{ObjectTypeDS; JobDS} \\
\text{CanCreate} : \text{OBJECT_ID} \leftrightarrow \text{JOB_ID} \\
\text{ran CanCreate} \subseteq \text{dom Jobs} \\
\text{dom CanCreate} = \text{ObjectTypes}
\]

The relationship CONSISTS is functional. The entities are paired using many-to-one mapping constraint. Here the entity DATAOBJECT is compulsory, and the entity OBJECTTYPE is optional in the relationship CONSISTS:

\[
\text{ConsistsDS} \\
\text{DataObjectDS; ObjectTypeDS} \\
\text{Consists} : \text{DATA_ID} \leftrightarrow \text{OBJECT_ID} \\
\text{dom Consists} = \text{dom DataObjects} \\
\text{ran Consists} \subseteq \text{ObjectTypes}
\]

**The assigners** There are two standard relationships with the assigners that are JOBPERFORMANCE and ALLOWEDACCESS. They are defined as follows:

- \( JOBPERFORMANCE \equiv [Status : STATUS] \)
- \( ALLOWEDACCESS \equiv [AccessMode : ACCESSMODE] \)

Both relationships are relational, and they are now defined in the following way:

\[
\text{JobPerformanceDS} \\
\text{JobDS; UserDS} \\
\text{JobPerformances} : (JOB_ID \leftrightarrow USER_ID) \leftrightarrow JOBPERFORMANCE \\
\text{dom}(\bigcup(\text{dom JobPerformances})) = \text{dom Jobs} \\
\text{ran}(\bigcup(\text{dom JobPerformances})) = \text{dom Users}
\]

\[
\text{AllowedAccessDS} \\
\text{DataObjectDS; JobDS} \\
\text{Accesses} : (DATA_ID \leftrightarrow JOB_ID) \leftrightarrow ALLOWEDACCESS \\
\text{dom}(\bigcup(\text{dom Accesses})) = \text{dom DataObjects} \\
\text{ran}(\bigcup(\text{dom Accesses})) \subseteq \text{dom Jobs}
\]

In JOBPERFORMANCE both partners are compulsory. In ALLOWEDACCESS the entity DATAOBJECT is compulsory, and the entity JOB is optional.
The complete state  The complete state is now defined collecting all the specifications of the object sets.

| SecurityData | UserDS; JobDS; ObjectTypeDS; DataObjectDS; ConsistsDS |
| JobPerformanceDS; AllowedAccessDS; CanCreateDS |

Foreign key representation  It is possible to remove relational and functional relationships using the foreign key representation. Let us look at the CAN_CREATE relationship. Straightforward application of the given transformation rule gives:

\[
\text{ObjectType}_\text{FK}_{\text{DS}} \\
\text{JobDS} \\
\text{ObjectTypes} : \text{OBJECT}_\text{ID} \mapsto \text{P}_1 \text{JOB}_\text{ID} \\
\text{ran} \text{ObjectTypes} \subseteq \text{dom} \text{Jobs}
\]

Here \( \text{P}_1 \text{JOB}_\text{ID} \) is the foreign key.

Also the CONSISTS relationship is removed via the following transformation:

\[
\text{DataObject}_\text{FK}_{\text{DS}} \\
\text{ObjectType}_\text{FK}_{\text{DS}} \\
\text{DataObjects} : (\text{DATA}_\text{ID} \mapsto \text{DATAOBJECT}) \mapsto \text{OBJECT}_\text{ID} \\
\text{ran} \text{DataObjects} \subseteq \text{dom} \text{ObjectTypes}
\]

Here \( \text{OBJECT}_\text{ID} \) is the type of the foreign key.

As the definition of the instances of the entity DATAOBJECT has changed, the relationship AllowedAccessDS should be changed as follows:

\[
\text{AllowedAccessDS} \\
\text{DataObject}_\text{FK}_{\text{DS}}; \text{JobDS} \\
\text{Accesses} : (\text{DATA}_\text{ID} \mapsto \text{JOB}_\text{ID}) \mapsto \text{ALLOWEDACCESS} \\
\text{dom}(\bigcup(\text{dom Accesses})) = \text{dom}(\bigcup(\text{dom DataObjects})) \\
\text{ran}(\bigcup(\text{dom Accesses})) \subseteq \text{dom} \text{Jobs}
\]

Finally, the complete state is now even more compact than previously, and is given below:

| SecurityData | UserDS; JobDS; ObjectType_FK_DS; DataObject_FK_DS |
| JobPerformanceDS; AllowedAccessDS |

6 Conclusions

We have proposed a new method, ZIM, to do data modelling. Our main goal in designing the method was to be able to generate clear, readable, and well-structured Z specifications with as little redundancy as possible. Furthermore, ZIM is independent of used software development method that can be structured, object-oriented, or relational.

Compared to other reported integrations between the entity-relationship model and Z [4, 6, 7, 8, 12, 14, 15], our complete state specification is more concise. In ZIM, all the constraints belonging to an entity and a relationship are given when entities and relationships are presented. Moreover, we pay special attention to relationships between entities. In ZIM, the participating entities are not redefined when the relationships are specified. The participating entity set schemas (DomainDS, RangeDS) are used to constrain the defined relationship.
Data Modelling in ZIM

RelationshipDS
DomainEntityDS; RangeEntityDS
Relationships : DOMAIN_ID → RANGE_ID

dom Relationships = dom DomainEntities
ran Relationships = dom RangeEntities

However, in the related integrations relationship schemas have redundancy in the definition part where the set of existing instances (or known identifiers) of the participating entities (knownDomains, knownRanges) have to be redefined, even though there already exist definitions for the participating entities.

RelationshipDS

knownDomains : P DOMAIN_ID
knownRanges : P RANGE_ID
Relationships : DOMAIN_ID → RANGE_ID

knownDomains = dom Relationships
knownRanges = ran Relationships

Additionally, the reported integrations do not consider the different features of the relationships when the definition part of the relationship schemas are specified. Instead different relationships are specified by adding contraints to the predicate part of the schema. In ZIM, both standard and existential relationships have their own formalizations. The features of the relationships are declared in the definition part. Therefore, additional constraints are not needed in the predicate part of the relationship schemas.

Close to our method, in the sense of redundancy removal, is the SAZ method of Polack et al. [13], because in their work the key attribute is introduced only when defining the instances of both an entity and a relationship. Furthermore, in the relationship schemas the participating entity sets as types are used to constrain its domain and range sets:

\[
\forall \text{do : DomainEntityDS}; \text{ro : RangeEntityDS} \quad \text{do Relationships} \subseteq \text{do DomainEntities} \wedge \text{ran Relationships} \subseteq \text{ran ro RangeEntities}
\]

However, the quantification implies that the predicate part of the relationship schema is true only when both domain and range sets of the relationship are empty. If either domain or range entity is compulsory in the relationship the predicate is false. Polack et al. do not utilize the different functional features that can be expressed in Z functions directly as we do in ZIM when formalizing the relationships. Hence, in their relationship specifications some redundancy is present.

We are currently considering how state transitions and operations can be included in ZIM. Furthermore, we are considering the possibility to include our ideas into a CASE-tool.

Acknowledgement The first author wants to thank Seppo Lammi for discussions on relational data modelling. The comments on drafts of this paper by Emil Sekerinski, and the anonymous reviewers are gratefully acknowledged. This work was supported by the Technology Development Centre of Finland, Tekes.

References

[1] Barros R.S.M. On the Formal Specification and Derivation of Relational Database Applications. PhD Thesis, University of Glasgow, 1995.
[2] Chen P. The Entity-Relationship Model-Toward a Unified View of Data. ACM Transactions on Database Systems, vol. 1, no 1, March 1976.

[3] Engels G., Gogolla M., Hohenstein U. Hülsmann K., Löh-Richter P., Saake G., Ehrich H-D. Conceptual modelling of database applications using an extended ER model. Data & Knowledge Engineering, 9, 1992/1993.

[4] Ginbayashi J. Analysis of Business Processes: Specified in Z against an E-R Data Model. Technical Monograph PRG-103, Oxford University Computing Laboratory, 1992.

[5] Hainaut J. Database Reverse Engineering - Problems, Methods and Tools. Tutorial notes, CAISE-95, University of Jyväskylä, 1995.

[6] Hammond J. Producing Z Specifications From Object-Oriented Analysis. In: Bowen J.P., and Hall J.A. (eds.) Z User Workshop, Cambridge 1994. Springer-Verlag, 1994.

[7] Josephs M. B., Redmond-Pyle D. Entity-Relationship Models Expressed in Z: A Synthesis of Structured and Formal Methods. Technical Report PRG-TR-20-91, Oxford University Programming Research Group, 1991.

[8] Josephs M. B., Redmond-Pyle D. A Library of Z Schemas for use in Entity-Relationship Modelling. Technical Report PRG-TR-21-91, Oxford University Programming Research Group, 1991.

[9] Mannila H., Räihä K-J. The Design of Relational Databases. Addison-Wesley, Cambridge, 1992.

[10] Mišić V., Velašević D., Lazarević B. Formal Specification of a Data Dictionary for an Extended ER Data Model. The Computer Journal, vol. 35, no. 6, 1992.

[11] Navathe S., Pillalamarri M. OOER: Toward Making the E-R Approach Object-Oriented. In: Batini C. (ed) Entity-Relationship Approach. Elsevier Science Publishers B.V., North-Holland, 1989.

[12] Polack F., Whinston M., Hitchcock P. Structured Analysis - A Draft Method For Writing Z Specification. In: Nicholls (ed) Z User Workshop, York 1991. Springer-Verlag, 1992.

[13] Polack F., Mander K. Software Quality Assurance using the SAZ Method. In: Bowen J.P., and Hall J.A. (eds.) Z User Workshop, Cambridge 1994. Springer-Verlag, 1994.

[14] Semmens L., Allen P. Using Entity Relationship Models as a basis for Z Specifications. Technical Report, Leeds Metropolitan University, Faculty of Information & Engineering Systems, 1990.

[15] Semmens L., Allen P. Using Yourdon and Z: an Approach to Formal Specification. In: Nicholls J.E. (ed) Z User Workshop, Oxford 1990. Springer-Verlag, 1990.

[16] Semmens L., France R., Docker T. Integrated Structured Analysis and Formal Specification Techniques. The Computer Journal, vol. 35, no. 6, 1992.

[17] Shlaer S., Mellor S.J. Object Lifecycles- Modelling the World in States. Yourdon Press, Prentice Hall, 1992.

[18] Spivey J. The Z Notation: A Reference Manual. Prentice Hall, 2nd edition, 1992.

[19] Yourdon E. Modern Systems Analysis. Prentice Hall, Englewood Cliffs NJ, 1989.