Chapter extracted from André Filipe Arnedo Reis Master Thesis:
XIS-Reverse: A Model-Driven Reverse Engineering Approach for Legacy Information Systems
Instituto Superior Técnico - June 2017

Chapter 3

XIS-Reverse Approach

This chapter presents the proposed approach. The approach is composed by three main stages which are briefly introduced in section 3.1. Section 3.2 introduces the running example that is used to illustrate the transformations used in this approach. Sections 3.3, 3.4 and 3.5 describe in detail the stages of the approach. Finally, Section 3.6 still describes the technology behind the XIS-Reverse software tool.

3.1 Overview

XIS-Reverse is the MDRE approach proposed in this research. This approach focus in the first two stages of the broader reengineering process as discussed in Section 2.1.3 (see Figure 2.2), namely: Injection and Reverse Engineering stages. XIS-Reverse is represented in Figure 3.1 and starts with the extraction of the application database schema from an existent data source (through injection). Thereafter the Application data model is generated. Secondly, the user is able to define his own preferences by tweaking some heuristic's parameters, producing the XIS-Reverse configuration. The reverse engineering stage takes as input these two artefacts and generates XIS* models and RSLingo's RSL models. Then, the user can analyse the produced artefacts and introduce some refinements, such as changing automatically identified relationships into different ones in the Entities view, enhancing the Use-Cases views, etc.

3.2 Running Example

For better understanding and simplicity of the explanation, we introduce a simple but practical example: the "Tiny Social Security" (SS-App) legacy application.

The SS-App is a simple application that manages beneficiaries, dependents and tutors, associated documents, system's documents and user accounts.

This domain is modelled as 8 tables: *Beneficiary* and *Dependent* which primary keys are foreign keys to a *Person* table. *Dependents* table represents the relationship between a *Beneficiary* and a *Dependent* (through its foreign keys linked to each of those tables). A *Dependent* may have a *Tutor*, which are linked by a foreign key. Each *Beneficiary* has a set of documents (since a *Document* has a foreign key to a *Beneficiary* table) and a unique *UserAccount* (due to the unique constraint that *UserAccount* table has over its *BenIdNumber* foreign key). Furthermore, several documents not linked to any *Beneficiary* may exist as a *SystemDocument*. Each of those tables, columns, constraints and number of rows in the database are represented in Figure 3.2.

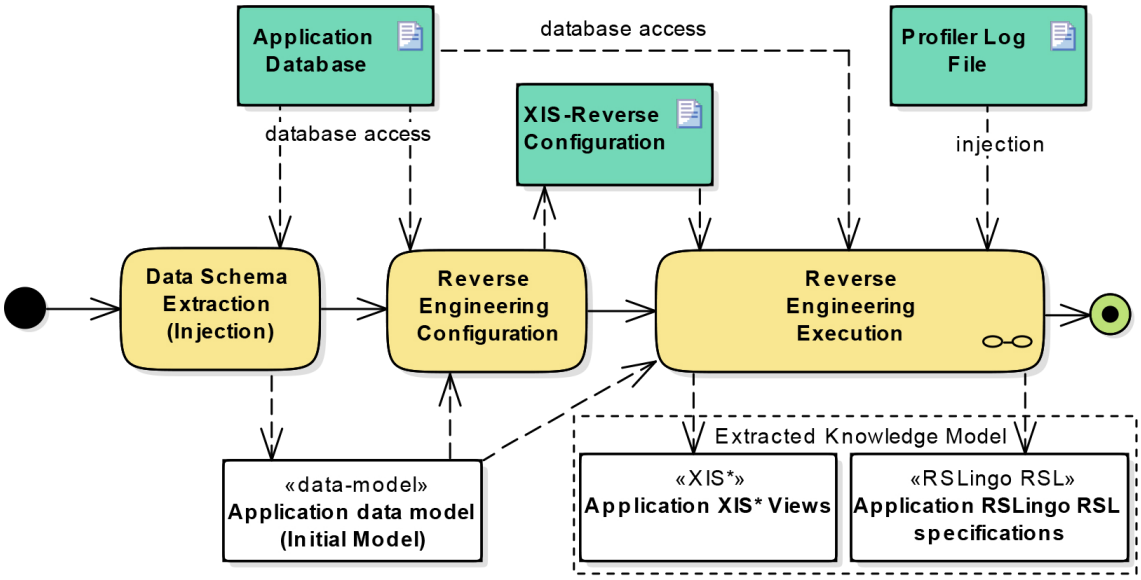


Figure 3.1: Overview of the XIS-Reverse approach.

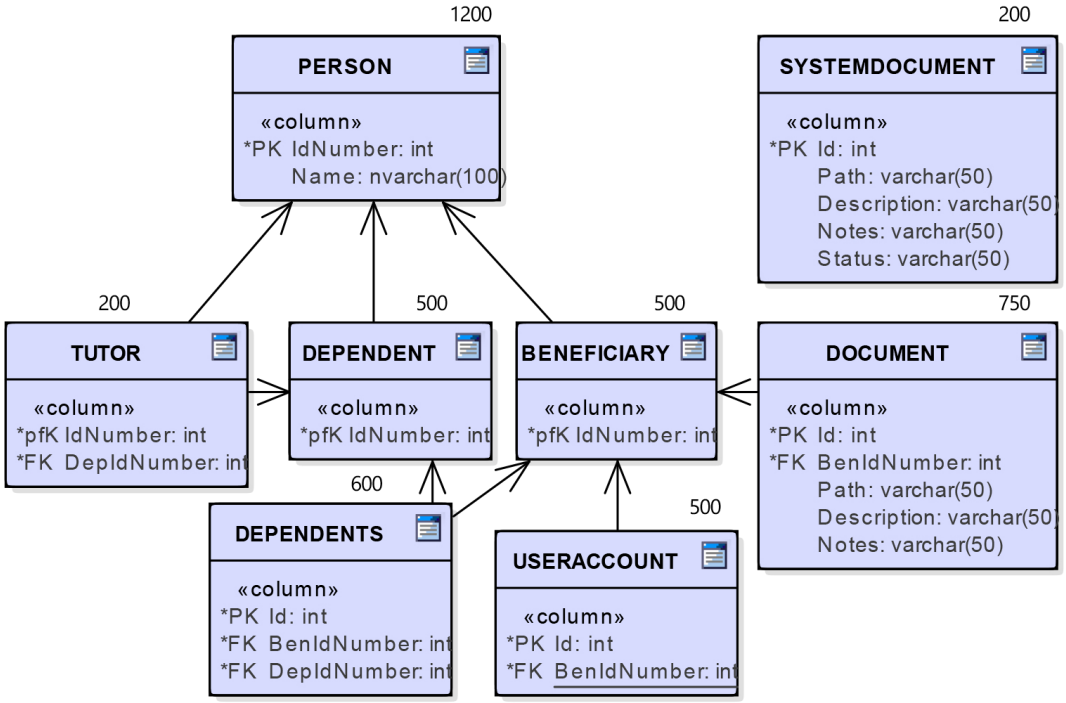


Figure 3.2: Running Example: SS-App data model.

3.3 Data Schema Extraction

Following the aforementioned process of Figure 3.1, during the Data Schema Extraction stage, in order to extract the application data model using our approach, a connection with a Database Management System (DBMS) is established and the database schema is extracted. For example, Figure 3.3 illustrates the SS-App database in Microsoft SQL Sever Studio which is extracted and represented as an application data model (illustrated in Figure 3.2)

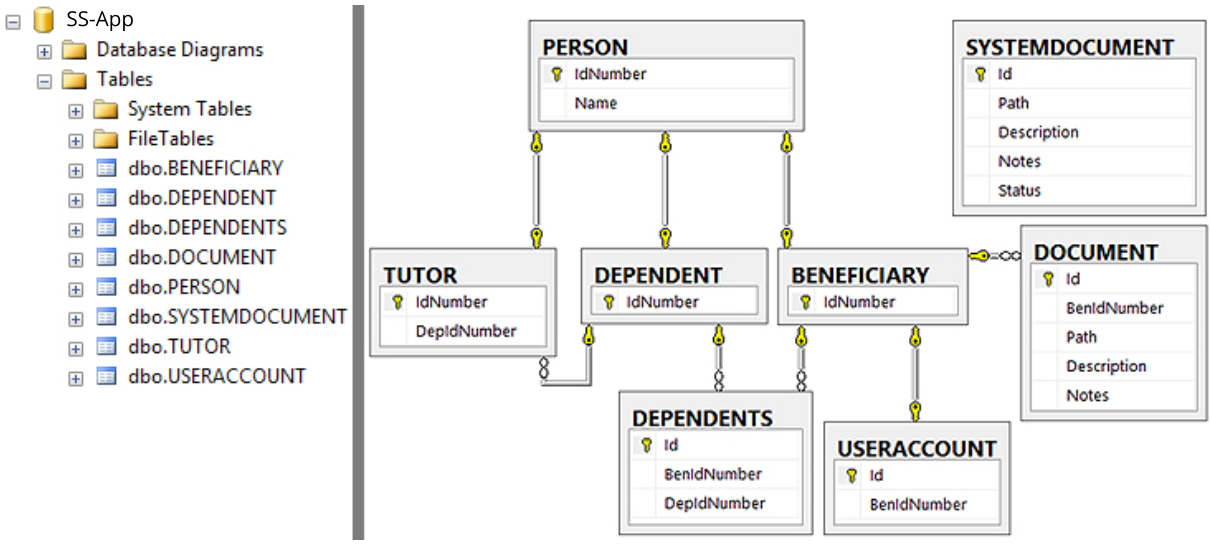


Figure 3.3: Running Example: SS-App database in Microsoft SQL Server Management Studio.

3.4 Reverse Engineering Configuration

A snapshot of the tool is illustrated in Figure 3.4, which is used to explain the main configuration features.

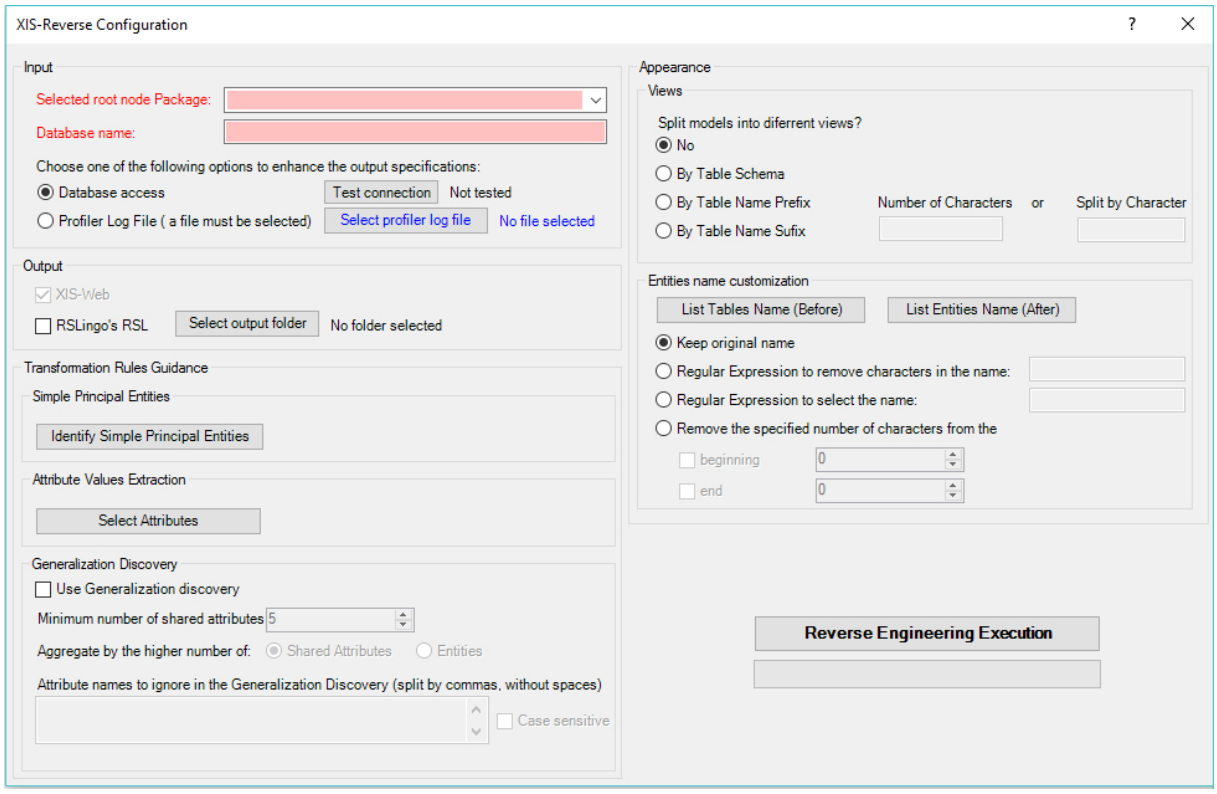


Figure 3.4: Main configuration panel of the tool.

The configuration panel of the tool has four main areas:

- **Input** - This area allows to specify the root node Package where the application data model was imported, to introduce the database name and to choose how to enhance the output specifications. Moreover, if database access is chosen to enhance specifications, it is also possible to test if it is

possible to establish connection with the database identified by the specified database name. On the other hand, if the Profiler Log File option is selected, a file must be selected in order to start the reverse engineering execution.

- **Output** - This area allows the user to select additional output representations, namely XIS-Web or RSLingo's RSL representations, and their output location (if applicable).

- **Transformation Rules Guidance** - This area allows the user to contribute with his knowledge to enhance the Reverse Engineering process. This configuration can be split into three areas: (i) Simple Principal Entities (ii) Attribute Values Extraction and (iii) Generalization discovery. The user can select which tables are Simple Principal Entities (which we consider as entities without many relationships, such as configuration entities or "kind of" entities). Moreover, the user can enhance the produced specifications by extracting values from a selected column of a certain table. Furthermore, in both cases the user can also filter the list of tables by specifying the maximum number of rows per table, which allows the user to find "kind of" tables more easily, since usually they have a small number of instances (rows). The selection window for both (i) and (ii) areas can be seen in Figure 3.5. Finally, the user can also activate generalization discovery and add some knowledge, such as a list of XisEntityAttribute names to ignore, the minimum number of shared XisEntityAttributes (e.g. 5 or 10), and to choose how to aggregate those entities, namely by the higher number of shared attributes or by the higher number of aggregated entities.

- **Appearance** - This area allows to improve readability of the produced specifications, by arranging models into smaller views (Views area) and changing the name of the entities to a clearer one (Entities name customization area). In regards to arranging models, it is possible to do it by table schema and also by the table name prefix/suffix, since it is also a recurrent pattern used to organize database tables. Using the last two options, the user must introduce the number of characters to take into account or specify a character which will split every table name (e.g. underscore). On the other hand, it is possible to clean the name of each entity by removing some characters from the corresponding table name, applying a regular expression or specifying the number of characters that will be removed from the beginning and/or the end of each name.

Moreover, during the Reverse Engineering Execution, additional information about the progress and the current state of the execution can be found in the title of the window allied with the progress bar below the Reverse Engineering Execution button.

3.5 Reverse Engineering Execution

Taking as input (i) the application data model, (ii) the XIS-Reverse configuration and (iii) the profiler log file or the database connection, the reverse engineering process is composed by a set of transformation tasks, that will be applied. Those include a chain of M2M transformations which are used to specify the application as a set of XIS* views and RSLingo's RSL specifications.

3.5.1 Initialization

Before M2M transformations take place, the profiler log file or the database connection, if provided, are used to extract more detailed information about the legacy application, that may enhance the produced specifications.

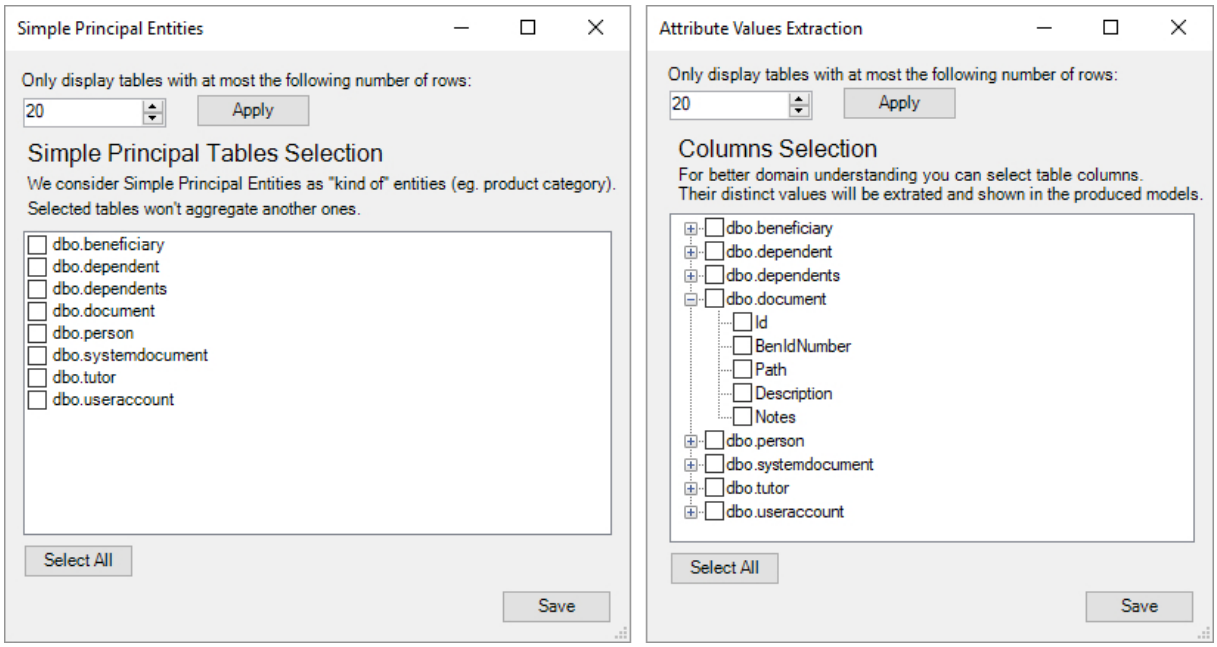


Figure 3.5: Configuration panel for Simple Principal Entities (left) and Attribute Values Extraction (right).

From the profiler log file it is possible to extract SQL statements (INSERT, SELECT, UPDATE and DELETE), and from those statements the set of used tables. Moreover, from that we generate 4 different lists, one for each type of statement, where each one of them holds for each table the number of occurrences for the corresponding kind of statement.

A database connection can be used to collect for each table in the initial application model, the number of rows in the specified application database.

3.5.2 Transformations to XIS models

The following transformations are broken down into the different set of input elements.

3.5.2.1 XIS Entities transformation rules

For each table of the extracted application data model, the following XIS Entities (XE-i) rules are applied:

(XE 1) If a table has exactly 2 foreign keys, each one to a distinct table, does not contain more attributes (excluding its foreign keys and primary keys), and one of the two things (1) the primary key is composed of exactly two primary keys which are also the foreign keys (two pfK) or (2) both foreign keys are not primary keys. Following that, this table will be translated into a XisEntityAssociation between the two referenced tables, using a many-to-many cardinality, ie. *:* cardinality (e.g. *Dependents* in the running example - illustrated in Figure 3.6).

(XE 2) Otherwise the table will map a XisEntity (e.g. *Beneficiary* in the running example - illustrated in Figure 3.7).

3.5.2.2 XIS Entity Attributes transformation rules

Regarding table columns, the following XIS Entity Attributes (XEA-j) rules are applied:

Let A, B be two tables related by a foreign key constraint from B (referencing table) to A (referenced table):

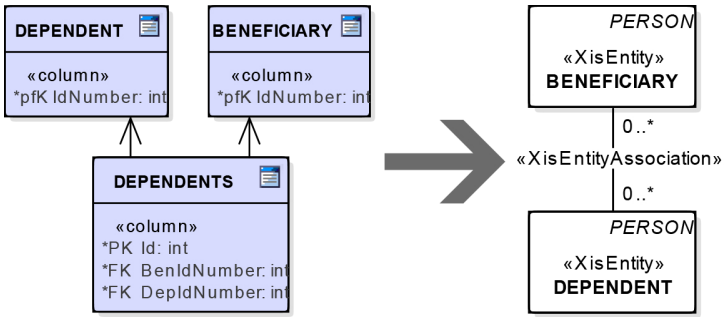


Figure 3.6: XE-1 transformation example.

Figure 3.7: XE-2 transformation example.

(XEA 1) For the complete Primary Key:

a) If all the foreign key columns in B are also its complete primary key, in other words B and A have a 1:1 relationship since B primary key references A primary key. Then B foreign keys will be translated into a XisEntityInheritance relationship, where the class equivalent to A will be the superclass and the equivalent class to B will be the subclass (e.g. *Person* (A) and *Beneficiary* (B) in the running example - illustrated in Figure 3.8).

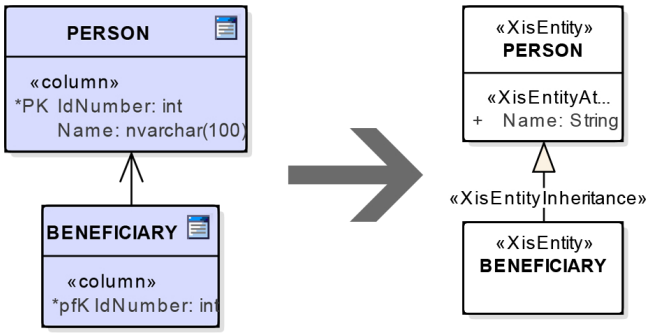


Figure 3.8: XEA-1-a transformation example.

b) Otherwise, the complete Primary Key will not be represented (e.g. *User Account* in the running example - illustrated in Figure 3.9). Due to XIS* Domain View similarity to a Class Diagram, each XisEntity can be seen as a Class, which can have objects, and each object is different by definition, so there is no need for a unique identifier for each XisEntity.

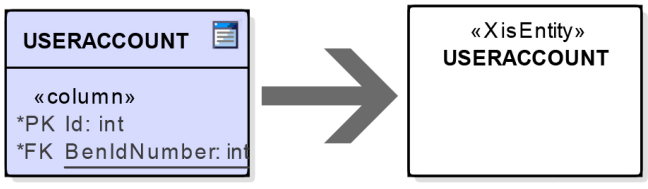


Figure 3.9: XEA-1-b transformation example.

(XEA 2) For the remaining columns with Foreign Key constraints:

a) If the foreign key in B constitute a Unique Index, the XisEntityAssociation will have a 1:1 cardinality, preserving the relationship direction (e.g. *BenIdNumber* of *UserAccount* in the running example (the underlined column name is used to illustrate the unique index property) - illustrated in Figure 3.10).

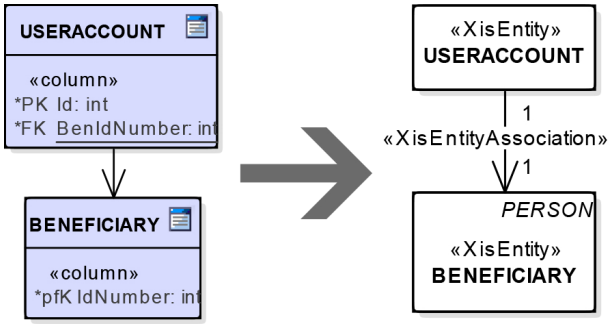


Figure 3.10: XEA-2-a transformation example.

b) If A was not selected as Simple Principal Entity by the user (configuration stage), and one of the following options is also true:

- The profiler log file was provided, both A and B tables were referenced in that file, and table B had the same or higher amount of INSERT operations as A did.
- The database connection was provided and table B had the same or higher amount of rows as A did (if B foreign key column allows NULL values, only rows with NOT NULL values on that column will be taken into account).

Then we assume that there is an aggregation relationship between entity A and B (A aggregates B), that is translated into a XisEntityAssociation, preserving the foreign key cardinality and represented with a bidirectional arrow (e.g. *Beneficiary* with 500 rows (A) and *Document* with 1000 rows (B) in the running example - illustrated in Figure 3.11).

This reasoning made sense for us since that, assuming that A and B are respectively strong and weak entities, A is the one which aggregates B, it is fair to assume that if that foreign key was made for that purpose, then in a system with good amount of data there must be at least the same amount of B instances created compared with A.

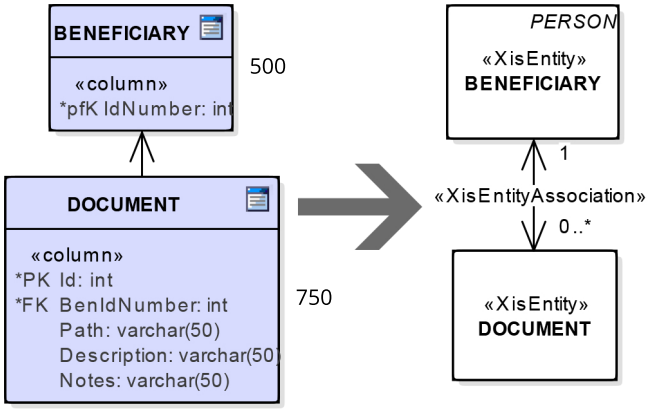


Figure 3.11: XEA-2-b transformation example.

c) Otherwise, the XisEntityAssociation will have a 1:* cardinality between table B and A, preserving the relationship direction (e.g. *Dependent* (A) and *Tutor* (B) in the running example - illustrated in Figure 3.12).

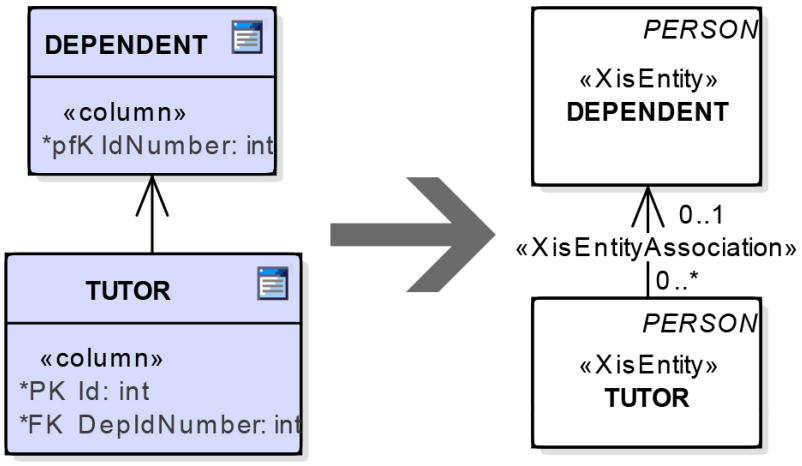


Figure 3.12: XEA-2-c transformation example..

(XEA 3) Otherwise the column will be translated into a XisEntityAttribute with a corresponding XIS* attribute type following a predefined mapping (e.g. *Status* from table *SystemDocument* in the running example (without annotation) - illustrated in Figure 3.13). Moreover, if previously configured, the user can explicitly select if values from a certain XisEntityAttribute from a particular XisEntity should be extracted and written as an annotation, which will be linked to the XisEntity (e.g. the same example as aforementioned, but now including the annotation with the values from that column - illustrated in Figure 3.13). This feature is a significant contribution since the produced specifications are enhanced with those values, allowing a better understanding of the XisEntities captured from the legacy domain.

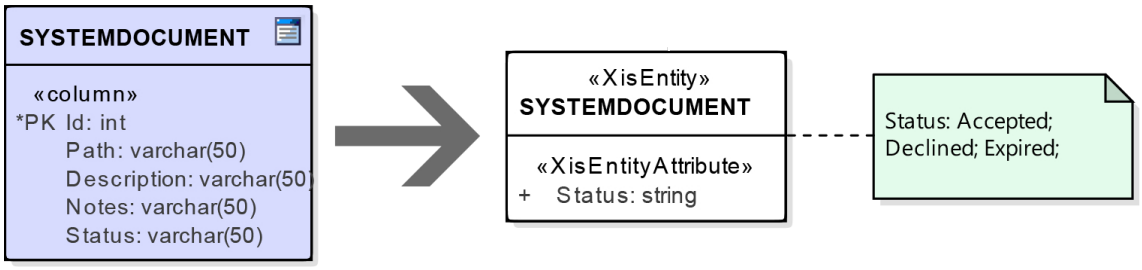


Figure 3.13: XEA-3 transformation example.

Note: in both b) and c) cases, from XEA-2, when the Not Null property is not set for that column (in other words, that column can be NULL), the XisEntityAssociation target's cardinality is the concatenation of 0.. with the original target cardinality. (e.g. 1 was the original target cardinality and the column allowed NULL values, then it becomes 0..1)

3.5.2.3 Implicit XIS Entity Inheritance transformation rule

For each XisEntity, after the previously mentioned XisEntities and XisEntityAttributes rules are applied and if the Generalization Discovery feature was configured, the following XIS Entity Inheritance (XEI) rule is applied:

(XEI) A XisEntityInheritance identification is performed comparing every XisEntity and their XisEntityAttributes with each other. We assume that two XisEntityAttributes are the same if they share the same name and type. During this comparison, if previously defined, some XisEntityAttributes can be excluded by their name. Moreover, we cluster 2 or more XisEntities if they share at least the same or higher configured number of shared XisEntityAttributes. Furthermore during this comparison only XisEntities without inheritance relationship are taken into account.

After this clustering procedure, there are two ways of finding inheritance, depending on the configuration, the list can be ordered by the descending number of entities or ordered by the descending number of shared XisEntityAttributes in each cluster. While this list has clusters of XisEntities, a Superclass is created for each cluster. This Superclass will have the identified XisEntityAttributes, each of the XisEntities in that cluster will not own those XisEntityAttributes and each XisEntity will be linked to the superclass using a XisEntityInheritance (e.g. *SystemDocument* and *Document* share 3 identical columns (Path, Description and Notes) in the running example - illustrated in Figure 3.14).

Although, the concept of generalization is based on shared characteristics between entities, that can be attributes, associations, or methods. From the available characteristics (attributes and associations) we did not take into account associations, since those may require deep knowledge about the system. For example, beneficiary and dependent entities could have an association with a country entity, however the relationship in the first case could be used to specify the residence country of the beneficiary, and in the second case it could be used to specify the country where the dependent was born. Moreover, the same entity can have multiple associations, with different meanings, to the same entity.

Furthermore, in order to implement the first phase of this algorithm, given the complexity that is comparing every XisEntity with each other taking into account all their XisEntityAttributes, in domains that can easily be compound by hundreds or thousands of elements, we based our approach in the MapReduce [37] programming model which allows to handle large data sets.

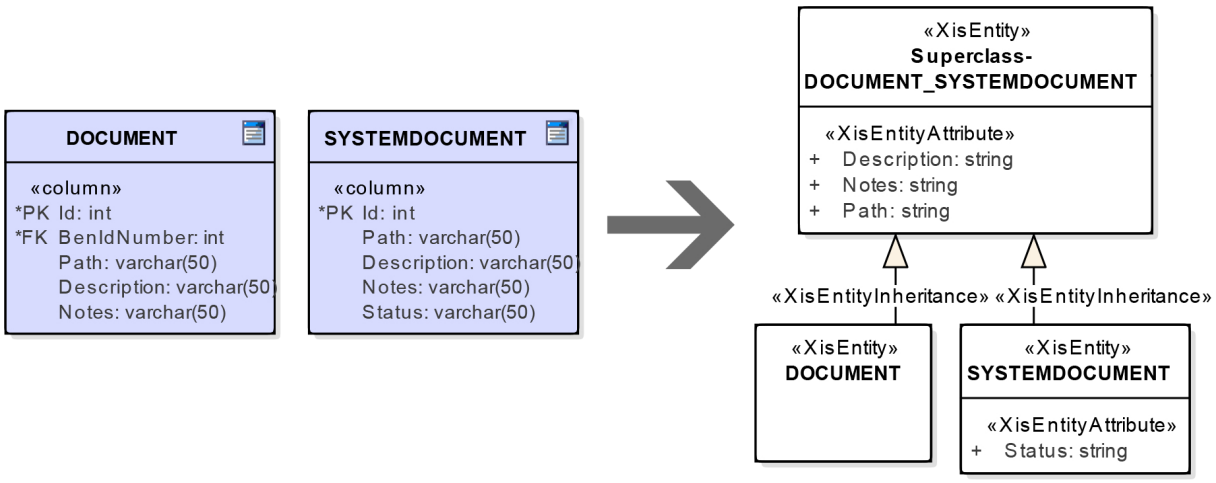


Figure 3.14: XEI transformation example.

3.5.2.4 XIS Business Entities transformation rules

For each XisEntity, the following XIS Business Entities (XBE-k) rules are applied:

(XBE 1) If the current XisEntity aggregates other XisEntities or is not aggregated by another XisEntity (i.e. we do not consider weak entities as Business Entities), then a corresponding XisBusinessEntity will be created, followed by the creation of a XisBE-EntityMasterAssociation that will link that XisBusinessEntity with its XisEntity from the Domain view (e.g. *Beneficiary* in the running example - illustrated in Figure 3.15).

And, for each XisEntityAssociation:

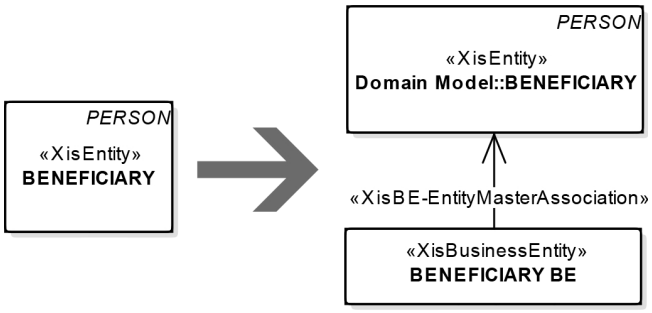


Figure 3.15: XBE-1 transformation example.

a) If the XisEntityAssociation is an aggregation (the current XisEntity aggregates another XisEntity) then it will be translated into a XisBE-EntityDetailAssociation between that XisBusinessEntity and the target XisEntity of the XisEntityAssociation (e.g. *Beneficiary* and *Document* in the running example - illustrated in Figure 3.16). Moreover, if the corresponding target XisBusinessEntity exists, there will be a XisBE-EntityReferenceAssociation between the target XisBusinessEntity and the source XisEntity.

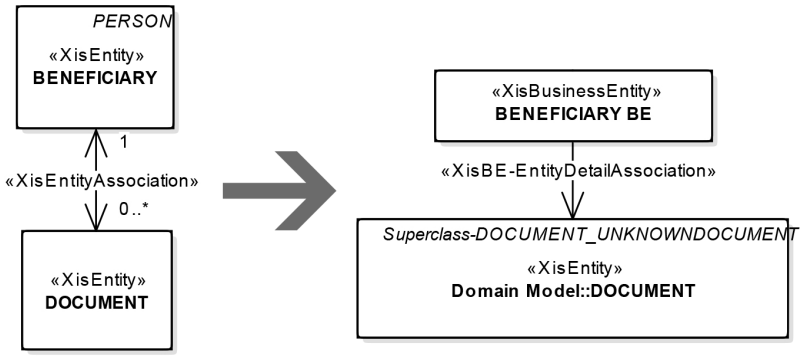


Figure 3.16: XBE-1-a transformation example.

b) Otherwise, if the current XisEntity has a 0..* cardinality in that XisEntityAssociation, it will be translated into a XisBE-EntityReferenceAssociation between that XisBusinessEntity and the other XisEntity of the XisEntityAssociation (e.g. *Tutor* and *Dependent* in the running example - illustrated in Figure 3.17). Therefore, if the XisEntityAssociation is a many-to-many association, both XisBusinessEntities will have a XisBE-EntityReferenceAssociation to the other XisEntity (e.g. *Beneficiary* and *Dependent* in the running example).

(XBE 2) Otherwise there is no mapping.

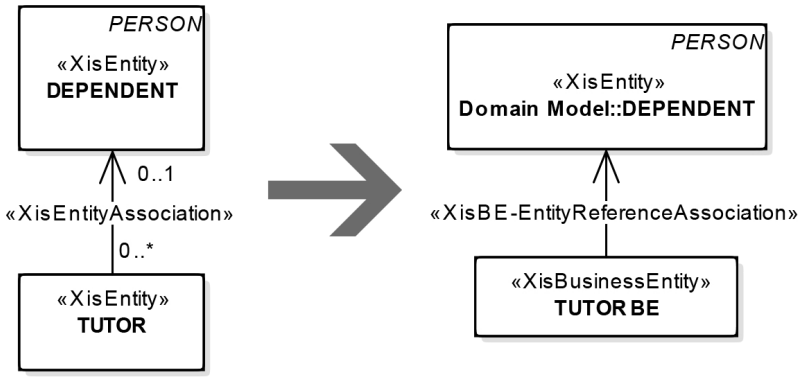


Figure 3.17: XBE-1-b transformation example.

3.5.2.5 XIS Entity Use Cases transformation rules

For each XisBusinessEntity, the following XIS Entity Use Case (XEUC) rule is applied:

(XEUC) There will be a corresponding XisEntityUseCase that is named “Manage” followed by the XisBusinessEntity master XisEntity name, associated to the XisBusinessEntity by a XisEntityUC-BEAssociation. And a default XisActor will be linked to the XisEntityUseCase using a XisActor-UCAssociation (e.g. *Beneficiary* in the running example - illustrated in Figure 3.18). Additionally, each XisEntityUseCase has boolean tagged values representing the CRUD operations for the master, detail and reference entities (e.g. CreateMaster, ReadDetail). And in our approach, all of them will be set to true.

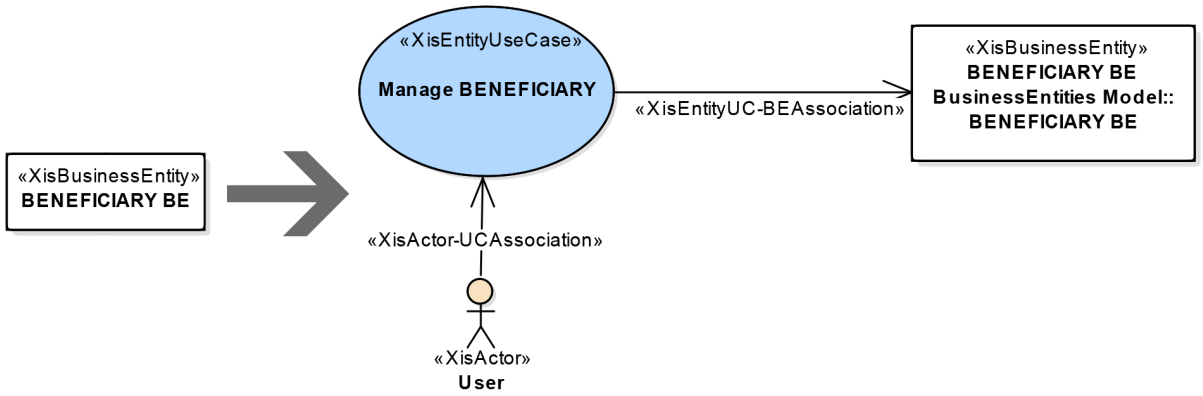


Figure 3.18: XEUC transformation example.

3.5.3 Transformations to RSLingo’s RSL models

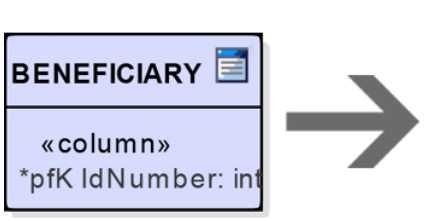
First of all, although RSLingo’s RSL specifications are textual artefacts, we consider the transformation chain which produces such artefacts a M2M transformation chain, since RSLingo’s RSL specifications conform to a well defined grammar, allowing to perform validations and then those specifications can also be forward engineered to produce several specifications conforming other formats, such as EXCEL files.

RSLingo’s RSL M2M transformations start together with the XIS M2M transformations, in order to collect all the needed information as RSLingo’s RSL specifications.

3.5.3.1 RSLingo's RSL Data Entities transformation rules

Since RSLingo's RSL uses a specification for Entities very similar to SQL, the following RSLingo's RSL Data Entities (RDE-i) rules are applied:

(RDE 1) Every table is mapped into a RSLingo's RSL dataEntity (e.g. *Beneficiary* in the running example - illustrated in Figure 3.19). Moreover, if the number of attributes of this dataEntity is higher than 5, its subtype will be Complex, otherwise will be Simple.



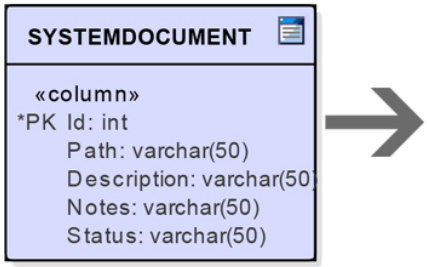

```
1 dataEntity e_BENEFICIARY : Principal : Simple [
2   name "Beneficiary"
3   isA e_PERSON
4   attribute IdNumber: Integer [name "Idnumber" NotNull]
5   primaryKey (IdNumber)
6   foreignKey e_PERSON(IdNumber)
7   description "has 500 records (by xis-reverse in 2017/05/04)"
8 ]
```

Figure 3.19: RDE-1 transformation example.

(RDE 2) Every column is mapped into a RSLingo's RSL attribute. Also taking into account their NOT NULL properties, and converting their type into an equivalent RSLingo's RSL attribute type (e.g. *IdNumber* column from *Beneficiary* in the running example - illustrated in Figure 3.19 line 4).

a) If a given column has a Primary Key or a Foreign Key constraint, the RSLingo's RSL dataEntity will represent that as well, including the referenced table for the second case (e.g. *IdNumber* column from *Beneficiary* in the running example - illustrated in Figure 3.19 line 5 and 6 respectively).

b) Likewise XIS Entity Attributes rule XEA-3, values from a certain attribute can be extracted (e.g. *Status* from table *SystemDocument* in the running example - illustrated in Figure 3.20 line 7).




```
1  dataEntity e_SYSTEMDOCUMENT : Principal : Simple [
2    name "Systemdocument"
3    isA e_Superclass_DOCUMENT_SYSTEMDOCUMENT
4    attribute Id: Integer [name "Id" NotNull]
5    attribute Status: String [
6      name "Status"
7      values "Accepted; Declined; Expired"
8    ]
9    primaryKey (Id)
10   description "has 200 records (by xis-reverse in 2017/05/04)"
11 ]
```

Figure 3.20: RDE-2-b transformation example.

(RDE 3) For each identified XisBusinessEntity, the corresponding RSLingo's RSL dataEntity will have Principal as dataEntity type (e.g. *Beneficiary* in the running example - illustrated in Figure 3.19 line 1). The left over RSLingo's RSL dataEntities are classified as Secondary (e.g. *Document* in the running example).

(RDE 4) Every RSLingo's RSL dataEntity that its corresponding XisEntity has a Superclass, will have an isA reference to the Superclass entity (e.g. *Person* is the Superclass of *Beneficiary* in the running example - illustrated in Figure 3.19 line 3). In the same way as aforementioned in 3.5.2.3,

implicit generalizations can be detected comparing every XisEntity generated. From that, found generalizations will also be added as RSLingo's RSL dataEntities and their subclasses will have an isA reference to the entity they inherit from (e.g. *SystemDocument* Superclass in the running example - illustrated in Figure 3.20 line 3).

3.5.3.2 RSLingo's RSL Data Entity Views transformation rules

For each XisBusinessEntity created, the following RSLingo's RSL Data Entity Views (RDEV) rules are applied:

(RBEV) There will be an equivalent RSLingo's RSL dataEntityView, with the same name, and same master, detail and reference associations to RSLingo's RSL dataEntities. (e.g. *Beneficiary* in the running example - illustrated in Figure 3.21).

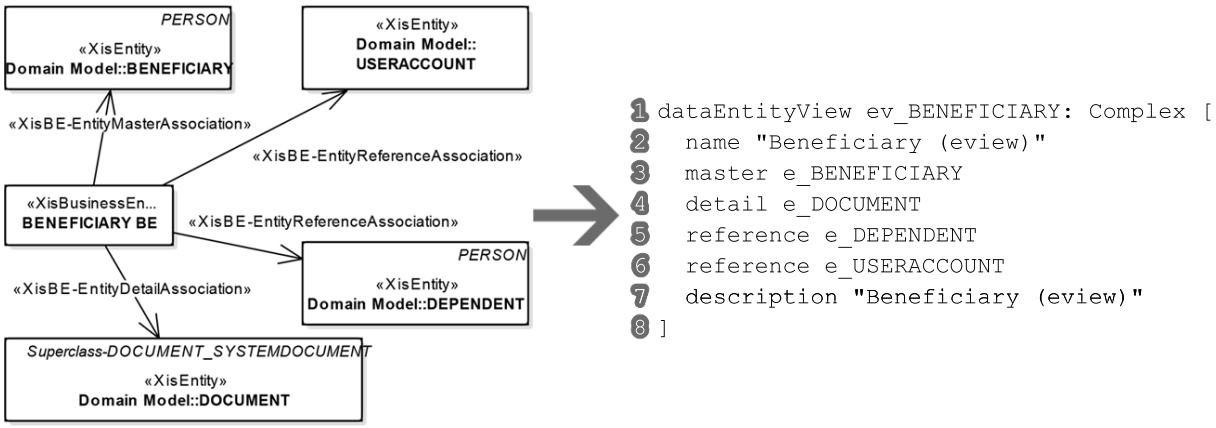


Figure 3.21: RBEV transformation example.

a) If the produced RSLingo's RSL dataEntityView has detail and reference associations or the sum of detail and reference associations is higher than 5 it is classified as Complex, otherwise, if it has detail or reference associations it is classified as VerySimple (e.g. *Beneficiary* (left), *Dependent* (middle) and *Person* (right) respectively in the running example - illustrated in Figure 3.22 line 1).

```
1 dataEntityView ev_BENEFICIARY: Complex [
2   name "Beneficiary (eview)"
3   master e_BENEFICIARY
4   detail e_DOCUMENT
5   reference e_DEPENDENT
6   reference e_USERACCOUNT
7   description "Beneficiary (eview)"
8 ]
```

```
1 dataEntityView ev_DEPENDENT: Simple [
2   name "Dependent (eview)"
3   master e_DEPENDENT
4   reference e_BENEFICIARY
5   description "Dependent (eview)"
6 ]
```

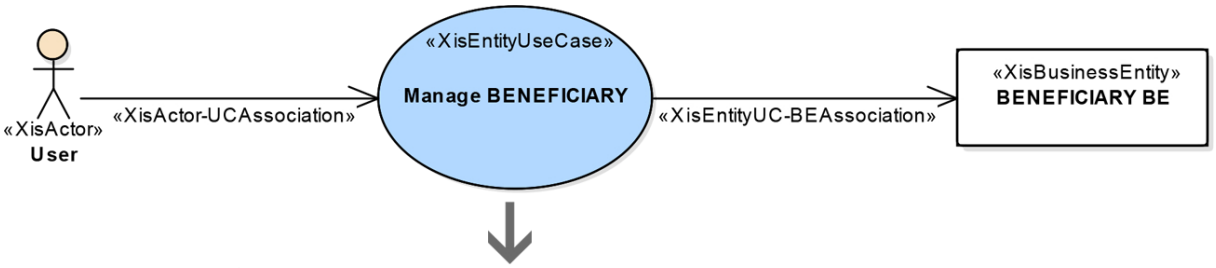
```
1 dataEntityView ev_PERSON: VerySimple [
2   name "Person (eview)"
3   master e_PERSON
4   description "Person (eview)"
5 ]
```

Figure 3.22: RBEV-a transformation example.

3.5.3.3 RSLingo's RSL Use Cases transformation rule

For each XisEntityUseCase created, the following RSLingo's RSL Use Cases (RUC) rule is applied:

(RUC) There will be an equivalent RSLingo's RSL useCase with, the same name, EntitiesManage as type, a default actor which initiates the use case, an equivalent RSLingo's RSL dataEntityView, and the following actions: Create, Read, Update, Delete, Search and Filter (e.g. *Beneficiary* in the running example - illustrated in Figure 3.23).



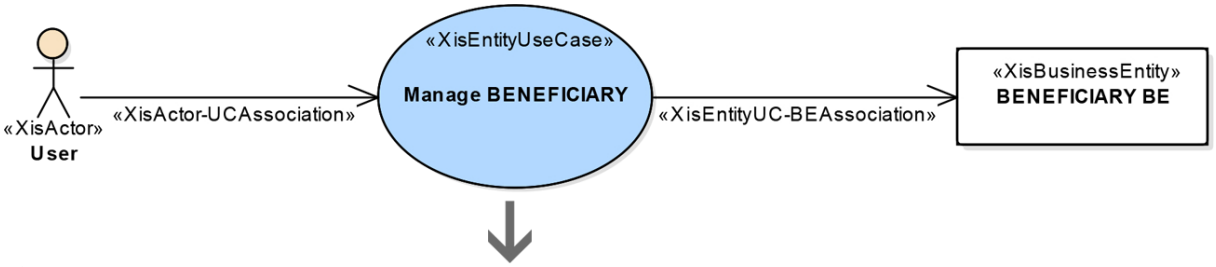

```
useCase uc_4: EntitiesManage [
  name "Manage Beneficiary"
  actorInitiates a_User
  dataEntityView ev_BENEFICIARY
  actions Create, Read, Update, Delete, Search, Filter
]
```

Figure 3.23: RUC transformation example.

3.5.3.4 RSLingo's RSL User Stories transformation rule

For each XisEntityUseCase created, the following RSLingo's RSL User Stories (RUS) rule is applied:

(RUS) There will be an equivalent RSLingo's RSL userStory with, UserStory as type, the same name, a default actor, and the following iWant property: "Manage ≪user story name≫ with Create, Read, Update, Delete, Search, and Filter features" (e.g. *Beneficiary* in the running example - illustrated in Figure 3.24).



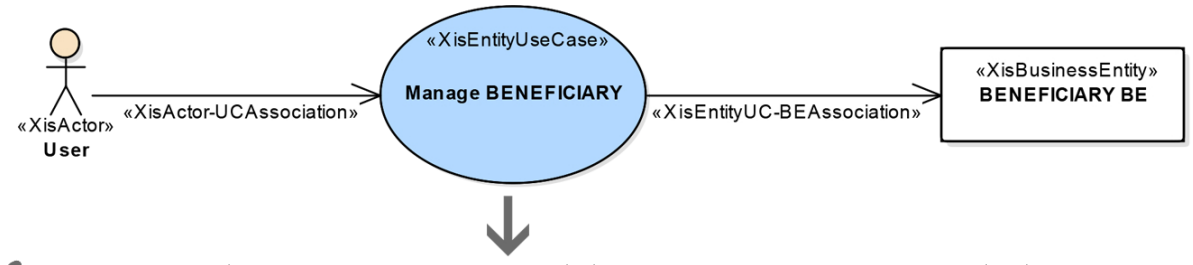

```
userStory us_4: UserStory [
  name "Manage Beneficiary"
  asA a_User
  iWant "to manage Beneficiary with Create, Read, Update, Delete, Search, and Filter
  soThat "some reason"
]
```

Figure 3.24: RUS transformation example.

3.5.3.5 RSLingo's RSL Functional Requirements transformation rule

For each XisEntityUseCase created, the following RSLingo's RSL Functional Requirements (RFR) rule is applied:

(RFR) There will be an equivalent RSLingo's RSL FR with, Functional as type, the same name, a default stakeholder, and Should as priority (e.g. *Beneficiary* in the running example - illustrated in Figure 3.25).



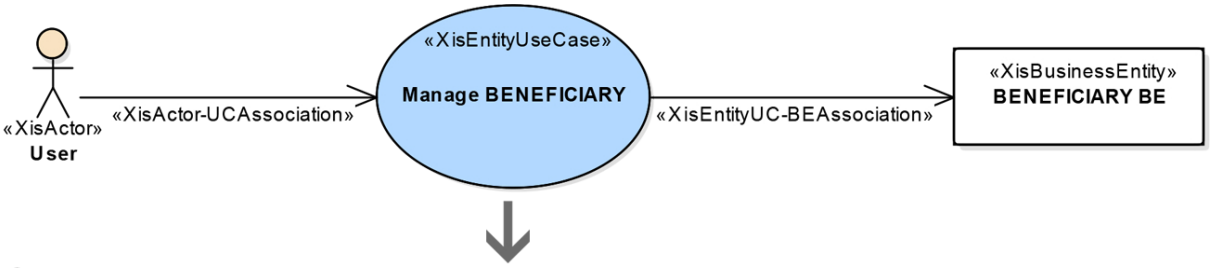

```
FR fr_1: Functional [name "Manage Beneficiary" stakeholder stk_user priority Should]
```

Figure 3.25: RFR transformation example.

3.5.3.6 RSLingo's RSL Goals transformation rule

For each XisEntityUseCase created, the following RSLingo's RSL Goals (RG) rule is applied:

(RG) There will be an equivalent RSLingo's RSL goal with, Concrete as type, the same name, a default stakeholder, and Should as priority (e.g. *Beneficiary* in the running example - illustrated in Figure 3.26).



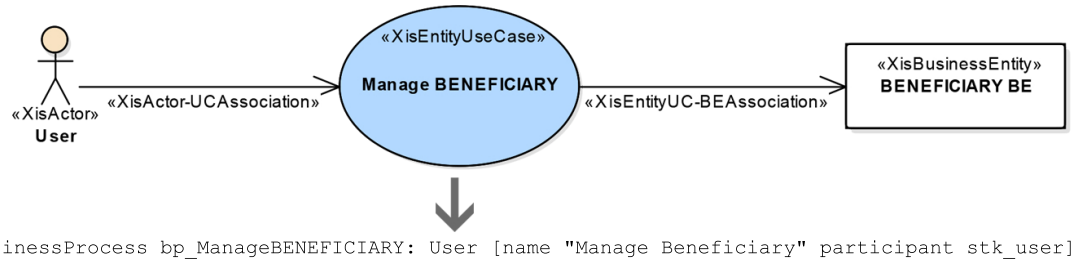

```
goal g_2: Concrete [name "Manage Beneficiary" stakeholder stk_user priority Should]
```

Figure 3.26: RG transformation example.

3.5.3.7 RSLingo's RSL Business Processes transformation rule

For each XisEntityUseCase created, the following RSLingo's RSL Business Processes (RBP) rule is applied:

(RBP) There will be an equivalent RSLingo's RSL businessProcess with, User as type, the same name, and a default participant (e.g. *Beneficiary* in the running example - illustrated in Figure 3.27).




```
businessProcess bp_ManageBENEFICIARY: User [name "Manage Beneficiary" participant stk_user]
```

Figure 3.27: RBP transformation example.

3.5.3.8 RSLingo's RSL Terms transformation rule

For each XisEntityUseCase created, the following RSLingo's RSL Terms (RT) rule is applied:

(RT) There will be an equivalent RSLingo's RSL term with, Noun as type, the same name, applicable to a DataEntity, and a default description (e.g. *Beneficiary* in the running example - illustrated in Figure 3.28).

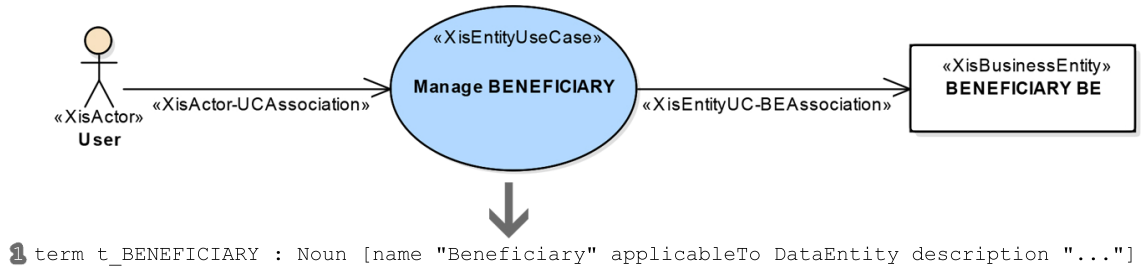


Figure 3.28: RT transformation example.

3.5.4 Reverse Engineering of the Running Example

To give an overall view of the produced XIS* models and RSLingo's RSL specifications with the XIS-Reverse approach, this section presents the generated models and specifications in a wider way.

To do so, the small but also detailed SS-App will be used (see Section 3.2 for further information). In order to generate the specifications we used the following set of configurations:

- **Input:** selected database access to enhance the output specifications.
- **Output:** selected both XIS-Web and RSLingo's RSL.
- **Transformation Rules Guidance:** selected *Status* column from *SystemDocument* to extract its values (Attribute Values Extraction); and activated generalization discovery by the higher number of shared attributes with the minimum of 3 attributes (Generalization discovery).

Figure 3.29 illustrates the produced XIS* Domain view and Listing 3.1 shows an excerpt of the produced RSLingo's RSL dataEntities. The main difference between the produced specifications is that only the XIS* models are capable to represent the aggregation between *Beneficiary* and *Document* through XEA2-b.

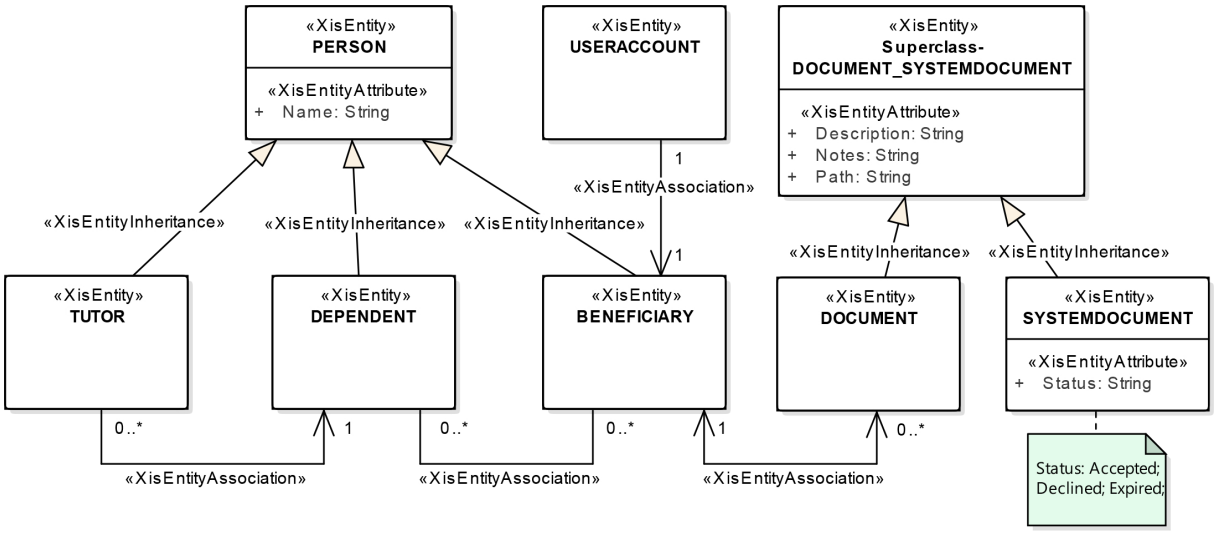


Figure 3.29: SS-App XIS* Domain view.

Listing 3.1: Excerpt of SS-App RSLingo's RSL dataEntities.

```
dataEntity e_BENEFICIARY : Principal : Simple [
  name "Beneficiary"
  isA e_PERSON
  attribute IdNumber: Integer [name "Idnumber" NotNull]
  primaryKey (IdNumber)
  foreignKey e_PERSON(IdNumber)
  description "has 500 records (by xis-reverse in 2017/04/29)"
]
dataEntity e_Superclass_DOCUMENT_SYSTEMDOCUMENT : Principal : Simple [
  name "Superclass Document Systemdocument"
  attribute Path: String [name "Path"]
  attribute Description: String [name "Description"]
  attribute Notes: String [name "Notes"]
  description "has 0 records (by xis-reverse in 2017/04/29)"
]
dataEntity e_DOCUMENT : Secondary : Simple [
  name "Document"
  isA e_Superclass_DOCUMENT_SYSTEMDOCUMENT
  attribute Id: Integer [name "Id" NotNull]
  attribute BenIdNumber: Integer [name "Benidnumber" NotNull]
  primaryKey (Id)
  foreignKey e_BENEFICIARY(BenIdNumber)
  description "has 750 records (by xis-reverse in 2017/04/29)"
]
dataEntity e_SYSTEMDOCUMENT : Principal : Simple [
  name "Systemdocument"
  isA e_Superclass_DOCUMENT_SYSTEMDOCUMENT
  attribute Id: Integer [name "Id" NotNull]
  attribute Status: String [name "Status" values "Accepted; Declined; Expired"]
  primaryKey (Id)
  description "has 200 records (by xis-reverse in 2017/04/29)"
]
...
```

Figure 3.30 illustrates the extracted *XisBusinessEntities* and their relationships with the *XisEntities* from the Domain View, namely Master, Reference and Detail, using the XBE-k rules. Listing 3.2, shows an excerpt of the equivalent entities, as *dataEntityViews*, through the RBEV rule.

Listing 3.2: Excerpt of SS-App RSLingo's RSL dataEntityViews.

```
dataEntityView ev_BENEFICIARY: Complex [
  name "Beneficiary (eview)"
  master e_BENEFICIARY
  detail e_DOCUMENT
  reference e_DEPENDENT
  reference e_USERACCOUNT
  description "Beneficiary (eview)"
]
dataEntityView ev_SYSTEMDOCUMENT: VerySimple [
  name "Systemdocument (eview)"
  master e_SYSTEMDOCUMENT
  description "Systemdocument (eview)"
]
...
```

Regarding Use Cases, User Stories, Functional Requirements, Goals, Business Processes and Terms, those won't be illustrated since they are not as distinct and detailed as the domain and business entities.

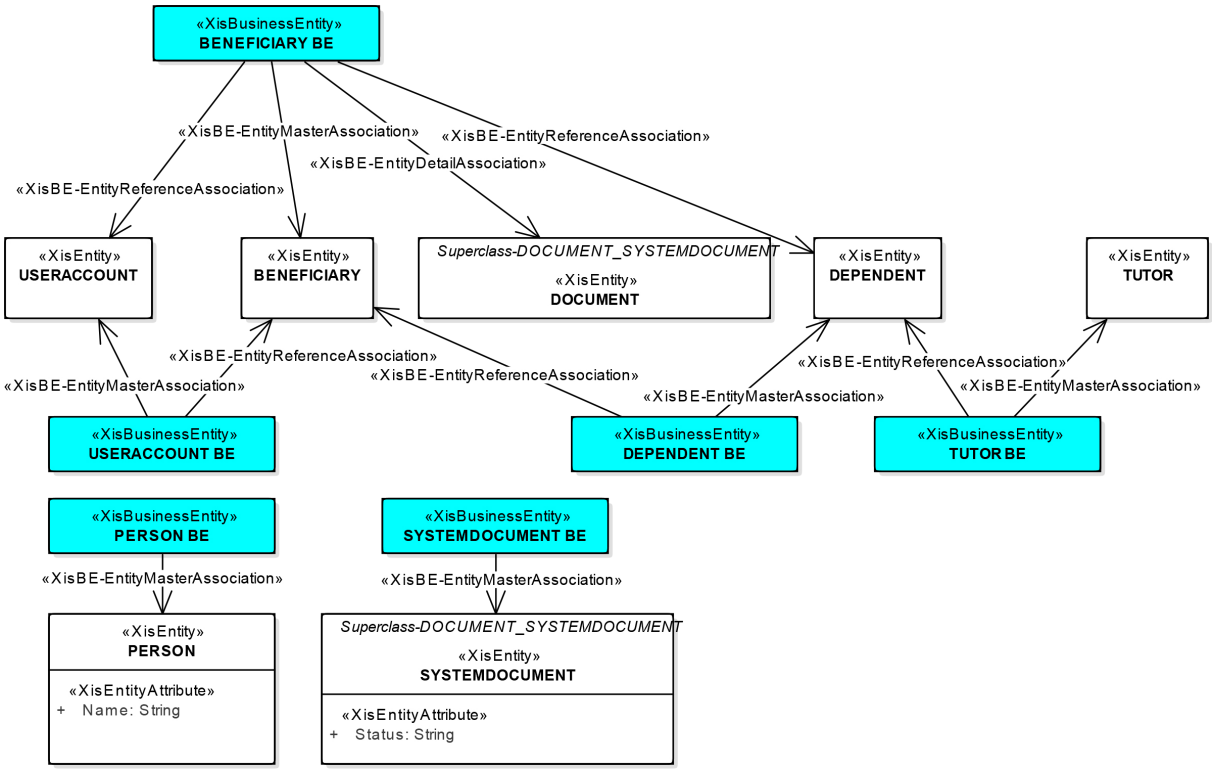


Figure 3.30: SS-App XIS* BusinessEntities view.