A Framework for implementing Web Scheme Transformers By-Example

Diplomarbeit

zur Erlangung des akademischen Grades eines Magisters der Sozial- und Wirtschaftswissenschaften

Eingereicht an der Johannes Kepler Universität Linz
Institut für Wirtschaftsinformatik
Data & Knowledge Engineering

Eingereicht bei: o.Univ.-Prof. Dr. Michael Schrefl
Betreuender Assistent: Dr. Stephan Lechner

Verfasst von: **Michael Karlinger**

Linz, im September 2004

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die Diplomarbeit mit dem Titel "A Framework for implementing Web Scheme Transformers By-Example" selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und alle benutzten Quellen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Linz, im September 2004

Kurzfassung

Transformers By-Example (TBE) ist eine Sprache für das Definieren und Anwenden von Schema Transformern. Solche Transformer können beim konzeptuellen Entwurf für das automatische Durchführen von Entwurfsschritten verwendet werden. Ein beispielhafter Entwurfsschritt im Bereich der konzeptuellen Modellierung von Webapplikationen ist, dass nach dem Definieren eines Entitätstyps im Datenschema eine entsprechende Seitenklasse im Hypertextschema eingefügt wird, die festlegt, wie die Instanzen dieses Entitätstyps auf einer Webseite darzustellen sind. Ein Merkmal von TBE ist, dass ein Transformer grafisch definiert wird, indem zwei generische Beispielschemata definiert werden, die jeweils das Schema vor und nach der Transformation widerspiegeln. Dadurch wird ein Transformer im Wesentlichen in der gleichen Notation definiert, in der auch Schemata definiert werden. Die für die Transformation eines Schemas notwendigen Modifikations-Operationen leitet TBE von diesen Beispielschemata ab. Diese Vorgehensweise unterscheidet TBE von anderen Ansätzen für Schema-Transformationen bei denen der Modellierer diese Modifikations-Operationen auf Basis einer internen Repräsentation der Schemata, wie zum Beispiel einer XML Repräsentation, direkt spezifizieren muss.

Ein TBE-System, das TBE für eine bestimmte Modellierungssprache implementiert, besteht aus zwei Bausteinen: Der erste Baustein (Grafischer Editor) ermöglicht das Definieren von Schemata und Transformern, das heißt das Definieren generischer Beispielschemata. Der zweite Baustein (TBE-engine) ermöglicht das Ableiten der Modifikations-Operationen von der grafischen Definition des Transformers. Weiters führt dieser Baustein die eigentliche Transformation von Schemata durch.

Der Gegenstand dieser Diplomarbeit ist die prototypische Implementierung der TBE-engine. Da der Ansatz von TBE auf verschiedene Modellierungssprachen angewendet werden kann, wird ein Framework (TBE-framework) zur Verfügung gestellt, das die Implementierung von TBE-engines für konkrete Modellierungssprachen erleichtert. In dieser Diplomarbeit wird die Implementierung einer TBE-engine für die Modellierungs-sprache WebML gezeigt.

Der spezifische Entwurf des TBE-frameworks ermöglicht den Einsatz von beliebigen grafischen Editoren. Diese Diplomarbeit zeigt, wie WebRatio, das CASE tool der Modellierungssprache WebML, als grafischer Editor verwendet werden kann. WebRatio bietet keine direkte Unterstützung für das Definieren von TBE-spezifischen Konstrukten an. Deshalb wird in dieser Diplomarbeit weiters gezeigt, wie solche Konstrukte in textueller Form in WebRatio definiert werden können.

Abstract

Transformers By-Example (TBE) is a concept that facilitates the definition and application of scheme transformers. When defining a conceptual scheme modelers can use scheme transformers, which transform an input scheme into an extended or refined output scheme, for automatically performing modelling tasks. An exemplary modelling task in the sphere of web application modelling is "after having defined an entity type, add a page class for displaying the entity type's content". In TBE, a transformer is defined graphically by giving a generic example of an input scheme and an output scheme, i.e. a scheme before and after the transformation, respectively. Therefore, modelers define transformers in a notation that is similar to one which they are familiar with. The scheme modification operations necessary for performing transformations of schemes are derived from the graphical specification. This is in contrast to other approaches for scheme transformers where modelers have to specify such operations based on some internal representation of schemes, e.g. a representation in XML.

A TBE-system, i.e. the implementation of TBE for a particular modelling langugae comprises two building blocks: The first building block is a *graphical editor* for defining schemes and transformer definitions, i.e. generic examples of an input scheme and an output scheme. The second building block is an *engine* (*TBE-engine*) used for deriving scheme modification operations based on the graphical transformer definition *and* for performing transformer applications.

The main contribution of this thesis is a prototype implementation of the TBE-engine. Since the concept of TBE can be applied in arbitrary modelling languages, a framework (*TBE-framework*) is provided that enables the convenient implementation of TBE-engines for concrete modelling languages. We demonstrate the implementation of a TBE-engine for modelling language WebML.

The TBE-framework is designed to cooperate with different graphical editors. We demonstrate the cooperation of the TBE-engine for modelling language WebML with WebRatio, which is a commercial CASE-tool supporting modelling language WebML. WebRatio does not support TBE-specific constructs directly. Therefore, a further contribution of this thesis is to show how these specific TBE constructs can be specified

indirectly within WebRatio, i.e. by annotating such TBE-specific constructs in textual form.

Contents

1 INTRODUCTION 13

1.1 Context: Transformers By-Example 13

1.2 Purpose: Putting Transformers By-Example to work 21

1.3 Outline of the thesis 29

2 WEBML 31

2.1 Development method 31

2.2 Modelling language 34

2.3 XML representation of WebML schemes 37

2.4 CASE tool support – WebRatio 39

3 TRANSFORMERS BY EXAMPLE 43

3.1 The logical representation of schemes 43

3.2 Formal semantics of transformers 45

3.3 Transformers and their application 58

4 DEFINING AND APPLYING TRANSFORMERS WITHIN WEBRATIO 61

4.1 Textual representation of TBE-directives 62

4.2 Customized-directives 66

4.3 Defining transformer *IndexPCForET* 69

4.4 Applying transformer *IndexPCForET* 75

5 ARCHITECTURE 79

5.1 Base architecture 80

5.2 Processes 81

5.3 Datastructures 86

6 IMPLEMENTATION 99

6.1 Choice of technologies 100

6.2 Two-layered implementation of the TBE-engine 107

6.3 Implementation of Datastructures 115

6.4 Implementation of Processes 125

7 RELATED WORK 147

7.1 Approaches to scheme transformations 147

7.2 The graphical editor *TransEd* 152

8 CONCLUSION 157

1 Introduction

Contents

1.1 Context: Transformers By-Example 13

1.1.1 Motivation for transformers 14
1.1.2 By-example approach to transformers 17
1.1.3 Building blocks of a TBE-system 20

1.2 Purpose: Putting Transformers By-Example to work 21

1.2.1 Implementing the graphical editor 21
1.2.2 Implementing the TBE-engine 24

1.3 Outline of the thesis 29

This chapter discusses the context and the purpose of this diploma thesis. SECTION 1.1 introduces Transformers By-Example (TBE), which is the context of the thesis. TBE is a language for defining and applying scheme transformers. Such transformers facilitate the process of modelling web applications in that they assist modellers in performing recurrent modelling tasks. SECTION 1.2 presents the purpose of this diploma thesis, which is to develop a framework for putting TBE to work. Finally, SECTION 1.3 outlines the diploma thesis.

1.1 Context: Transformers By-Example

This section introduces the TBE approach, which is the context of this diploma thesis. SECTION 1.1.1 motivates the usage of transformers during web application development. SECTION 1.1.2 describes the by-example approach to the definition of transformers. SECTION 1.1.3 describes the building blocks of a TBE-system, i.e. the implementation of this by-example approach.

1.1.1 Motivation for transformers

Web-applications are n-tiered information systems accessible via the internet. Their user-interface consists of numerous hypertext pages allowing users to explore and navigate the web application's content. By following links, users may navigate the web-application's content and trigger the execution of operations that change the web-application's business state [Con99].

Web-applications are developed by applying adequate development methods and development tools in order to achieve high-quality products at minimum costs [LRS99]. Developing web-applications is especially complex for the following reasons:

- *User interface personalization*: The web application's content is typically presented to different user groups, where each group requires a personalized view of the content and the operations that can be triggered [MMCF03]. Personalization leads to more complex user-interfaces, thus causing additional development efforts.
- *Multi device delivery*: Web-applications are often accessible via different devices, like for example Personal Digital Assistants or Personal Computers [MMCF03]. The distinctive features of the delivery devices, above all their different screen resolutions, demand the development of individual user interfaces for each delivery device which in turn complicates development.

A recent approach to the development of web-applications is called *model-driven development* [MMCF03, CCP01, MAM03]. Thereby, web-applications are first modelled at the conceptual level and then implemented automatically or semi-automatically through code generation [Fra99]. Examples of model-driven web-application development methods are WebML [MMCF03], OO-H [CCP01] and ARANEUS [MAM03]. Such development methods typically integrate several models, each addressing a different design aspect of a web application. For a detailed comparison of model-driven web-application development methods and their CASE tool support confer to [Fra99].

Applying model-driven development methods achieves:

- *Less efforts for mastering the complexity of web-application development*, as detailed architectural and implementation issues are neglected at conceptualization [MMCF03].

- *Shortened development time*, as manually writing code is replaced by automated code generation [Dro97]. This benefit is especially valuable when developing web applications in the domain of e-commerce, as this market imposes short development cycles [RSL99].
- *Reduced maintenance and evolution efforts*, as requests for changes can be turned into changes at the conceptual level, which are then propagated to the implementation through code generation [MMCF03].

When developing a web-application by applying a model-driven development method, defining the web-application's conceptual scheme, which results from conceptual modelling, is an essential task. A web-application's conceptual scheme (short scheme) typically integrates several sub-schemes, each addressing a distinctive design aspect. Commonly the design of the web-application's content structure and hypertext structure is addressed by distinctive sub-schemes called content scheme and hypertext scheme, respectively.

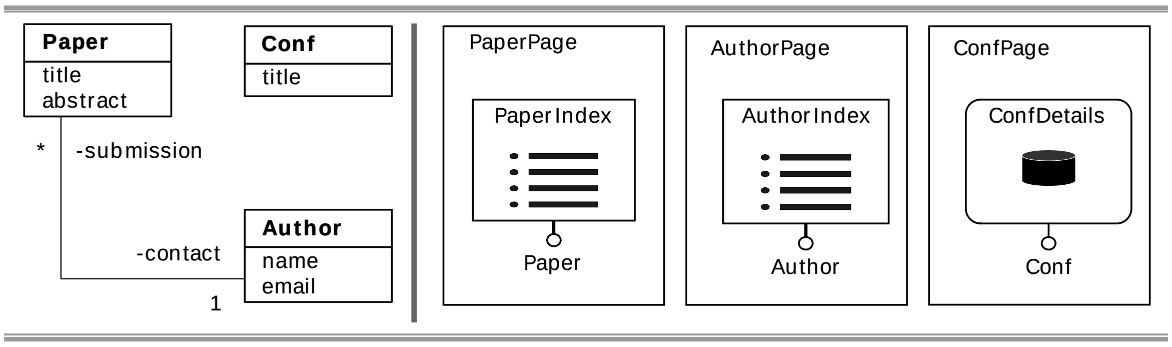


Figure 1.1: WebML scheme of the CMA: content scheme (left) and hypertext scheme (right).

***Example 1.1**: FIGURE 1.1 shows the scheme of a web-application intended for managing conferences defined with the conceptual modelling language WebML [MMCF03]. This web-application is called **C**onference **M**anagement **A**pplication (short: CMA).*

The left part of FIGURE 1.1 shows the content scheme, which defines the hypertext site's content in terms of an entity-relationship diagram. Entity type `Conf`, which is intended to contain exactly one entity, describes data to be presented at the conference's main page. Entity types `Author` and `Paper` describe information about authors and submitted papers, respectively.

The right part of FIGURE 1.1 shows the hypertext scheme, which defines how the content is to be organized in hypertext pages. This is expressed by page classes *containing content units. Each content unit refers to the entity type that is the content source for the content unit. Page class `AuthorPage` contains index unit `AuthorIndex`, which presents a list of all authors at the `AuthorPage`. Page class `PaperPage` is structured analogously. However, for page class `ConfPage`, an index unit is not appropriate. In order to present the sole entity of type `Conf`, page class `ConfPage` contains data unit `ConfDetails`.*

When defining the overall scheme of a web-application, the scheme is extended or refined step by step through adding scheme elements like, for example, entity types, attributes of entity types or page classes, in order to meet the requirements specified for the web-application. A set of scheme extensions and refinements needed to meet a particular requirement, will subsequently be referred to as *modelling task*.

Although the particular modelling tasks differ among the conceptualization of different web-applications, several modelling tasks, have to be fulfilled by modellers again and again in the same manner. Examples for such *recurrent modelling tasks* are listed below. The name of the respective modelling task is denoted in parentheses. [Lec04].

- For some entity types, it shall be possible to add or delete members of that entity type via the web interface. This requires entry forms and links that trigger content management operations like *insert* or *delete*. (`ProvideForInsert`, `ProvideForDelete`).

- It is often the case that for an entity type, which defines some part of the web site's content, a page class with a content unit is required for presenting the members of that entity type. (`PageClassForEntityType`).

As extending and refining schemes again and again manually and in a similar manner is cumbersome, it would be convenient for modellers to have *scheme transformers* that can perform such recurrent modelling tasks. Scheme transformers speed up the definition of a web application's scheme and in turn the entire development process, since modellers can have many modelling tasks performed by applying scheme transformers [Lec04].

***Example 1.2**: Modelling task `PageClassForEntityType` could be automatically performed by a scheme transformer called `IndexPCForET`. This scheme transformer would generate a page class with an index unit, according to some naming policy, for each entity type. Thus, page classes `PaperPage` and `AuthorPage`, shown in*

FIGURE 1.1, could be automatically generated instead of being manually defined by the modeller.

1.1.2 By-example approach to transformers

Transformers-by-example [LS04] is a visual by-example approach for defining scheme transformers. Consequently, the graphical notation used for defining schemes is used for defining transformers. In the following it is shown how transformers are visually defined and how transformers are used for performing scheme transformations.

Defining transformers: The definition of a transformer comprises two constituent parts. One part specifies by means of constraints, which configurations of scheme elements are extended or refined by the transformer. This part is called *query part*, since TBE derives a query, based on this specification, which retrieves all scheme element configurations from a scheme that fulfill the constraints. The other part defines how to generate new scheme elements and how to extend existing ones. This part is called *generative part*.

A transformer's query part and generative part are both expressed by giving an "example" of what is desired. These "examples" are expressed in the same graphical notation as used for defining schemes. However, the "examples" are generic specifications, from which the TBE-system derives an executable transformer. Therefore, these examples are referred to as *templates*. Consequently, a transformer's query part is referred to as *query template* and a transformer's generative part is referred to as *generative template*. [LS04]

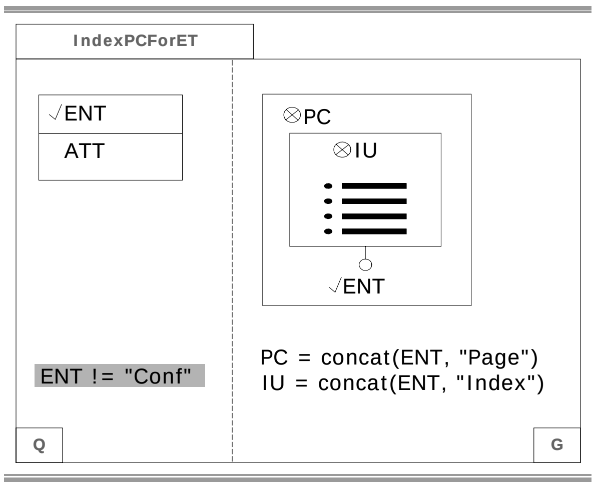


Figure 1.2: By-example definition of transformer `IndexPCForET` in notation of WebML.

***Example 1.3**: FIGURE 1.2 depicts a by-example definition of transformer `IndexPCForET` in terms of modelling language WebML. Thus the transformer defines a transformation of WebML schemes. Apart from the string concatenation expressions, the definition of transformer `IndexPCForET` is graphically notated like a WebML scheme. Please ignore the grey-shaded expression for the moment. The query template, shown in the left part of FIGURE 1.2, defines that entity types are to be selected. The generative template shown in the right part of FIGURE 1.2 defines that a page class comprising an index unit is to be generated.*

Each template is defined by using the same scheme elements and the same notation as used for specifying schemes. However, there are the following two differences: (1) Instead of concrete values, the entries in scheme elements are *variables*. (2) Additionally, a template contains symbols, comparison constraints and construction expression, which are called TBE-directives in summary. A fixed set of symbols is used for tagging variables in order to distinguish different types of variables. Variables of a generative template that are tagged with symbol "⊗" are new-element variables, i.e. variables representing scheme elements to be generated. Variables of a query template that are tagged with symbol "√" are result-variables, i.e. variables that are comprised within the query template's result. Comparison constraints are used to constrain variable bindings. Construction expressions define how to derive new property values, e.g. by means of string concatenation.

***Example 1.4**: Reconsider the by-example definition of transformer `IndexPCForET` in notation of WebML depicted in FIGURE 1.2. The query template comprises variables `ENT` and `ATT` representing entity types and attributes, because they are placed in the name section and in the attribute section of a graphical shape representing an entity type, respectively. When applied to a scheme, the query looks for valid bindings of variables `ENT` and `ATT` out of the domain of entity types and attributes, respectively. Since variable `ATT` is placed inside the graphical shape representing variable `ENT`, only those bindings are valid where the attribute represented by `ATT` is defined at the entity type represented by `ENT`. Variable `ENT` is a result-variable as it is preceded by symbol "√", whereas variable `ATT` is a non-result variable. Thus, the query's result comprises only the entity types but not the attributes.*

The generative template comprises variables `PC`, `IU` and `ENT`, which represent a page class, an index unit and an entity type as expressed by the graphical placement of these variables. Thereby, variable `ENT` is a parameter variable as expressed by symbol "√". Variable `ENT` represents an entity type that is to be provided as parameter each time the generative template is instantiated. Variables `PC` and `IU` are *new-element variable as expressed by symbol "⊗". They represent a page class and a new index unit to be generated.*

Applying transformers: A transformer, once defined, can be applied to various schemes, each time extending or refining the scheme as defined by the transformer's templates. Each transformer application is processed in two steps. First, the query template is evaluated in the context of the scheme to which the transformer has been applied. It achieves a relation whose tuples represent configurations of scheme elements that fulfill the constraints specified by the query template. Second, the generative template is iteratively instantiated for each such tuple `t`, each time having the generative template's parameter variables bound to the corresponding scheme elements in `t`.

***Example 1.5**: Suppose that transformer `IndexPCForET` depicted in FIGURE 1.2 is applied to the `CMA` content scheme off-the-shelf, which is depicted in FIGURE 1.1 Then, entity types `Paper`, `Author` and `Conf` are selected because they all match the pattern "Entity type comprising an attribute". For each of these entity types, the generative template is instantiated separately. For example, for entity type `Paper`, a page class `PaperPage` comprising an index unit `PaperIndex` referring to entity type `Paper` is generated. Similarly, page classes with index units are generated also for entity types `Author` and `Conf`. Note, that it is not desired, in the context of the `CMA` scheme, that page class `ConfPage` contains index unit `ConfIndex`. Therefore, in order to prevent the generation of page class `ConfPage` with index unit `ConfIndex` an individualized application of transformer `IndexPCForET` is required in order to achieve the desired outcome.*

Besides applying transformers off-the-shelf, TBE offers the following alternatives for adapting a transformer's behavior individually for each application: (1) Modellers may individually constrain query template variables in order to control which parts of the scheme shall be considered. Such individual constraints are called *application-specific constraints*. (2) Modellers may specify construction expressions that override those defined in the generative template in order to adapt the transformer's outcome. Such individual construction expressions are called *application-specific construction expressions*.

***Example 1.6**: Suppose that the application of transformer `IndexPCForET` to the `CMA` content scheme shall be individualized such that entity type `Conf` is not considered. For this purpose the application-specific constraint `ENT != "Conf"` depicted in FIGURE 1.2 is required. The outcome of this transformer application is page class*

`PaperPage` with index unit `PaperIndex` and page class `AuthorPage` with index unit `AuthorIndex`.

1.1.3 Building blocks of a TBE-system

A TBE-system implements TBE for a particular modelling language `L`, i.e. it supports the graphical definition and application of transformers. Such a TBE-system comprises two building blocks, i.e. a *graphical editor* for `L` and a *TBE-engine* for `L`, as described in the following.

When a modeller *defines* a transformer for modelling language `L`, she graphically defines a query template and a generative template both in notation of `L` using a graphical editor. Defining a template in notation of `L` means to define a scheme in notation of `L` and to extend the scheme with TBE-directives like "⊗" or "√". These templates are then passed to the TBE-engine in an internal representation of L, for example by means of XML, as the upper part of FIGURE 1.3 depicts. The TBE-engine derives an executable transformer on basis of the query template and the generative template, which is required for performing the application of a transformer. An executable transformer is a sequence of scheme modification operations that is executable on the internal representation of schemes. For example, if schemes are internally represented as XML documents an executable transformer could be an XQuery statement.

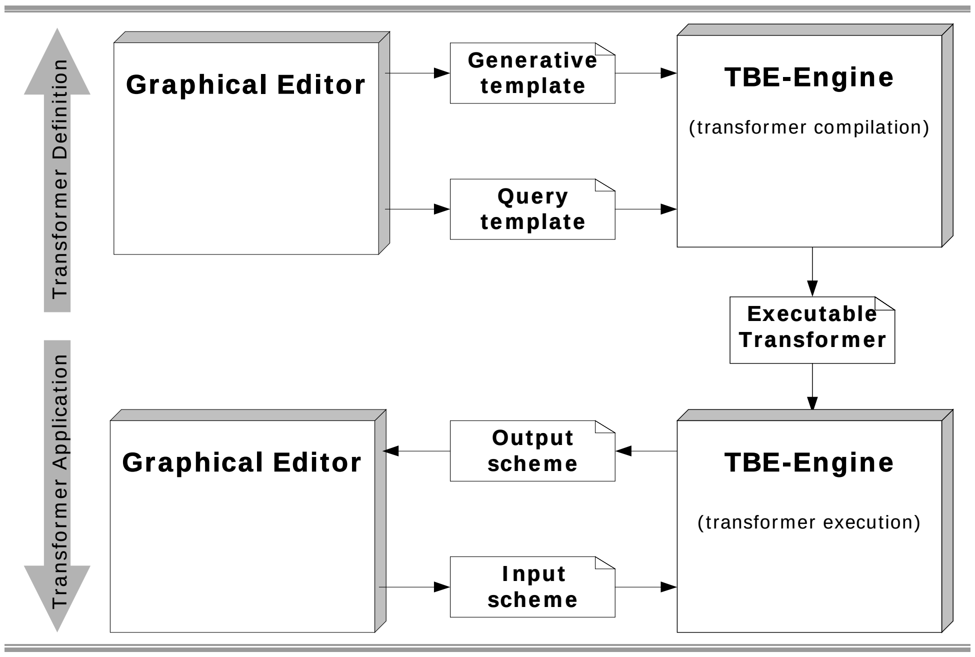


Figure 1.3: Building blocks of a TBE-system - Graphical editor and TBE-engine.

When a modeller *applies* a transformer, she defines an input scheme, i.e. the scheme to be transformed, in notation of L within the graphical editor, as depicted in the lower part of FIGURE 1.3. The input scheme is then transformed within the TBE-engine as specified by the selected transformer, via executing the previously derived executable transformer on the internal representation of the input scheme. Finally the resulting output scheme is displayed within the graphical editor.

The TBE-system presented in this diploma thesis uses an XML representation of WebML schemes as interface between the graphical editor and the TBE-engine. This XML representation is specified by an XML DTD called WebML.dtd and therefore normative, prescribed by the developers of WebML. Hence, the implementations of the graphical editor and the TBE-engine are exchangeable, i.e. the implementation of the graphical editor can be exchanged without adapting the implementation of the TBE-engine, and vice versa.

1.2 Purpose: Putting Transformers By-Example to work

The purpose of this diploma thesis is to develop a framework for putting TBE to work. This framework (*TBE-framework*) provides the components of a TBE-system that are independent of a particular modelling language L (*model-independent components*).

In order to apply the TBE-framework to modelling language L, the components of a TBE-system that are dependent on modelling language L (*model-dependent components*) are plugged in the TBE-framework.

SECTION 1.2.1 and SECTION 1.2.2 presents the alternatives for implementing the graphical editor and the TBE-engine, respectively. Further, the choice of the respective implementation alternative is reasoned.

1.2.1 Implementing the graphical editor

The graphical editor of a TBE-system that implements TBE for modelling language L has to fulfill the following requirements:

- *Defining schemes and templates*: The graphical editor has to provide for defining schemes in notation of `L`. In order to define templates the graphical editor additionally has to provide for specifying TBE-directives.

- *Compatibility to the TBE-engine*: The graphical editor has to provide for exporting schemes in notation of `L` in order to implement the interface to the TBE-engine.

- *Minimum implementation efforts*: A graphical editor is always specifically tailored for editing schemes and templates in notation of `L` and thus model-dependent at all. Yet, this diploma thesis focuses on the implementation of model-independent components of a TBE-system. Thus, a solution for implementing the graphical editor with minimum efforts is desired.

1.2.1.1 Implementation alternatives

For implementing the graphical editor of a TBE-system the following alternatives are available [LS04]:

- *Developing a graphical editor from scratch*: To develop a graphical editor from scratch is one alternative for implementing the graphical editor. Such a graphical editor can be specifically tailored to provide for defining schemes and specifying TBE-directives graphically.

- *Adapting a CASE tool's source code*: Every CASE tool provides for defining schemes. If a CASE tool is used as graphical editor for a TBE-system, it additionally has to provide for specifying TBE-directives. Obviously, if the developer has full control over the CASE tool's source code, she can adapt the CASE tool in order to provide for specifying TBE-directives graphically and for exporting schemes in the required manner.

- *Using a CASE tool off-the-shelf*: The main difference to the former alternative is that a CASE tool is not adapted in order to provide for specifying TBE-directives. Instead, the CASE tool is used off-the-shelf and TBE-directives are annotated in a textual form. For example, the tick preceding a variable's name, determining that the respective variable is a result variable, can be textually represented by a dollars sign, which is entered as prefix to the name of the respective variable. For specifying more complex TBE-directives, like constraints or construction expressions, in textual form, the CASE tool has to provide for annotating text to schemes. Typically, CASE tools enable the annotation of text by means of tag/value-pairs.

1.2.1.2 Choice of an implementation alternative

In the previous sections requirements to graphical editors and implementation alternatives have been described. In this section the implementation alternatives are evaluated and one of these alternatives is chosen.

Concerning the graphical editor's *compatibility to the TBE-engine* each of the implementation alternatives is applicable, since each alternative provides for exporting schemes in an internal representation.

Concerning the *definition of schemes and templates* again each of the implementation alternatives is applicable. Each alternative supports the definition of schemes at the same level. However, the level of support to the definition of templates is rudimentary when using a CASE tool off-the-shelf, since TBE-directives have to be specified in textual form. In contrast, when a CASE tool's source code is adapted or a graphical editor is developed from scratch the convenience of graphically specifying TBE-directives is provided to modellers.

Concerning the *implementation effort* the alternative of using a CASE tool off-the-shelf is preferred to the other alternatives, since actually no implementation effort arises.

The application of a CASE tool as graphical editor is demonstrated by WebRatio, which is the CASE tool for WebML. WebRatio provides for annotating TBE-directives in textual form by means of Properties, i.e. tag/value-pairs. Further, a syntax for TBE-directives in textual form is specified in this diploma thesis.

A convenient graphical editor for the TBE-system, called *TransEd*, is developed from scratch in a related diploma thesis [Wab04]. TransEd is a graphical editor for WebML schemes that additionally provides for graphically specifying TBE-directives.

1.2.2 Implementing the TBE-engine

The TBE-engine of a TBE-system that implements TBE for modelling language L has to fulfill the following requirements:

- *Provide for compiling transformer definitions*: The TBE-engine has to provide for compiling a query template and a generative template both in notation of L into an executable transformer.

- *Provide for performing transformer applications*: The TBE-engine has to provide for transforming a scheme in notation of L as specified by a particular transformer.

- *Applicability to various modelling languages*: The TBE-engine has to be applicable to various modelling languages, i.e. the TBE-engine has to comprise components that can be reused for implementing TBE-engines for other modelling languages than L. Such reusable components are called *model-independent* components. Consequently, components that have to be newly implemented for each modelling language L are called *model-dependent* components. The model-independent components of the TBE-engine make up the TBE-framework that can be refined in order to develop a concrete TBE-engine for modelling language L.

- *Compatibility to the graphical editor*: In order to cooperate with the graphical editor the TBE-engine has to process schemes in notation of L.

1.2.2.1 Implementation alternatives

For implementing a TBE-engine basically two alternatives are available. The first alternative is to implement a model-specific TBE-engine, i.e. a TBE-engine that is specifically tailored to the transformation of schemes of a particular modelling language. The second alternative is to implement a generic TBE-engine, i.e. a TBE-engine that aims at enabling scheme transformations for various modelling languages. These alternatives are illustrated in SECTION 1.2.2.1.1 and SECTION 1.2.2.1.2, respectively.

1.2.2.1.1 Model-specific TBE-engine

The upper part of FIGURE 1.4 depicts the compilation of a transformer definition. The `Generator` takes the query template and the generative template in notation of a particular modelling language `L` as input and generates the corresponding executable transformer. Such an executable transformer is specifically tailored to the transformation of schemes in notation of `L`.

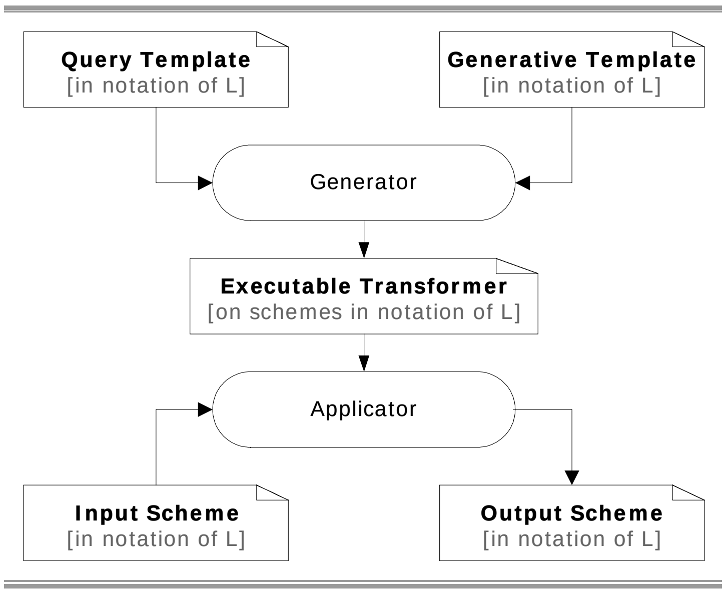


Figure 1.4: Components of a model-specific TBE-engine.

The lower part of FIGURE 1.4 depicts how a model-specific TBE-engine performs a transformer application. For this purpose the `Applicator` takes the input scheme in notation of `L` and executes the previously derived transformer directly on the input scheme. The resulting output scheme, again in notation of `L`, is finally returned.

1.2.2.1.2 Generic TBE-engine

The basic difference between a model-specific TBE-engine and a generic TBE-engine is, that a generic TBE-engine basically works on a standardized representation of schemes instead of working on the schemes directly. Such a standardized representation of schemes, called *logical representation of schemes*, is provided by TBE. The logical representation of a scheme is a representation of the scheme grounding on the consideration of a scheme as being a set of scheme elements and a set of connections between those scheme elements.

FIGURE 1.5 depicts the compilation of a transformer definition. The generic TBE-engine takes a query template and a generative template, both in notation of `L`, as input. These templates are then mapped into their logical representation by the `Mapper`. On basis of the logical representations of the templates the `Generator` derives a sequence of modification operations on the logical representation of a scheme. Such a sequence of modification operations is called transformer definition in terms of TBE. Thus, a transformer definition in terms of TBE specifies how the logical representation of a scheme is to be transformed. To generate a transformer definition in terms of TBE instead of generating an executable transformer directly is a major difference to model-specific TBE-engines.

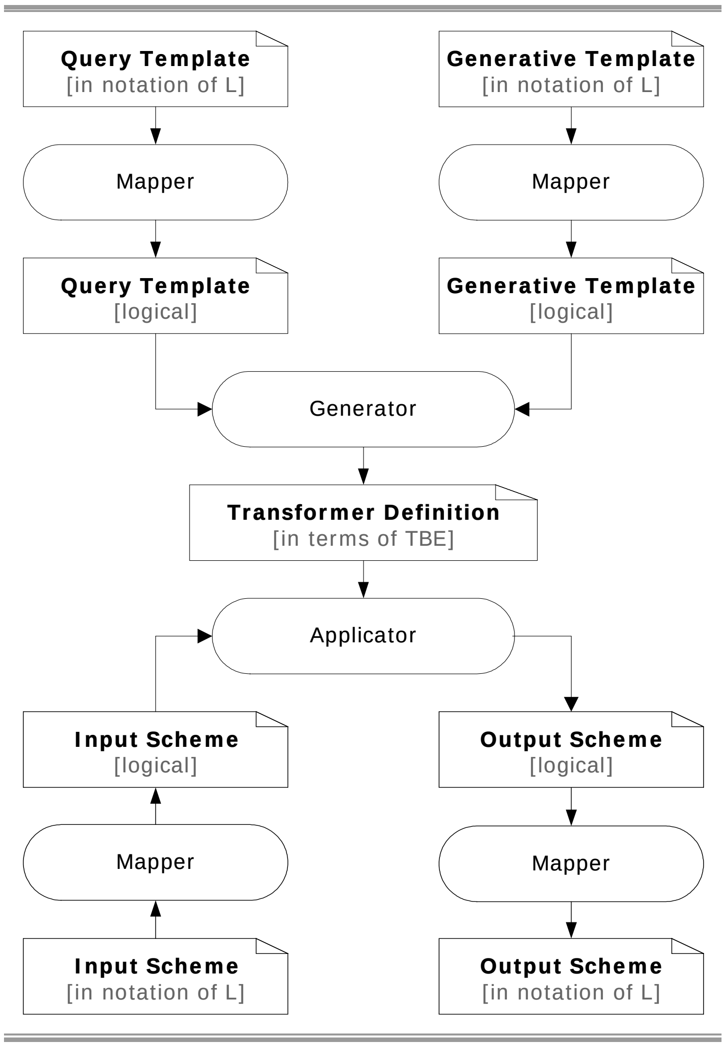


Figure 1.5: Components of a generic TBE-engine.

The lower part of FIGURE 1.5 depicts how a generic TBE-engine performs a transformer application. The input scheme in notation of `L` is first mapped into its logical representation by the `Mapper`. This mapping is necessary, since the generated transformer definition in terms of TBE specifies the transformation of the logical representation of a scheme and not the transformation of the scheme itself. Then the `Applicator` transforms the logical representation of the input scheme into the logical representation of the output scheme as specified by the transformer definition in terms of TBE.

For realizing the `Applicator` of a generic TBE-engine the following two alternatives are available. One alternative is that the `Applicator` interprets the transformer definition in terms of TBE ad-hoc. Another alternative is that the `Applicator` translates the transformer definition in terms of TBE first into an executable transformer, i.e. a script in terms of another (programming) language for which a processor exists. Then the executable transformer is executed by the `Applicator` on the input scheme in logical representation.

Finally the output scheme in logical representation is mapped back into its native representation, i.e. a scheme in notation of `L`.

The compilation of a transformer definition and its actual application is independent of modelling language `L`, since the logical representation of schemes is used. The mapping from schemes in notation of `L` into their logical representation and vice versa is dependent on the modelling language `L`, since each modelling language uses its own internal representation of schemes.

1.2.2.2 Choice of an implementation alternative

In the previous sections requirements to the TBE-engine and implementation alternatives have been described. In this section the implementation alternatives are evaluated and one of these alternatives is chosen.

Concerning the compilation of transformer definitions both alternatives, i.e. a model-specific TBE-engine and a generic TBE-engine, are applicable, since both alternatives provide for compiling a query template and a generative template in notation of WebML. Analogous, both alternatives are applicable for performing transformer applications to WebML schemes.

Concerning the compatibility to the graphical editor again both alternatives are applicable, since both use WebML schemes as interface to the graphical editor.

Concerning the applicability to various modelling languages a generic TBE-engine is preferred, since the components of a model-specific TBE-engine entirely comprises of model-dependent components. Therefore, all components of a model-specific TBE-engine have to be newly implemented in order to be applicable to another modelling language. For example, the `Applicator` of a model-specific TBE-engine for modelling language WebML is specifically tailored to the transformation of WebML schemes. Therefore, such an `Applicator` cannot be reused for implementing a TBE-engine for other modelling languages than WebML.

Whereas, a generic TBE-engine comprises mainly model-independent components that can be used for each implementation of a TBE-engine, i.e. the `Applicator` and the `Generator`. The only model-dependent component of a generic TBE-engine is the `Mapper`.

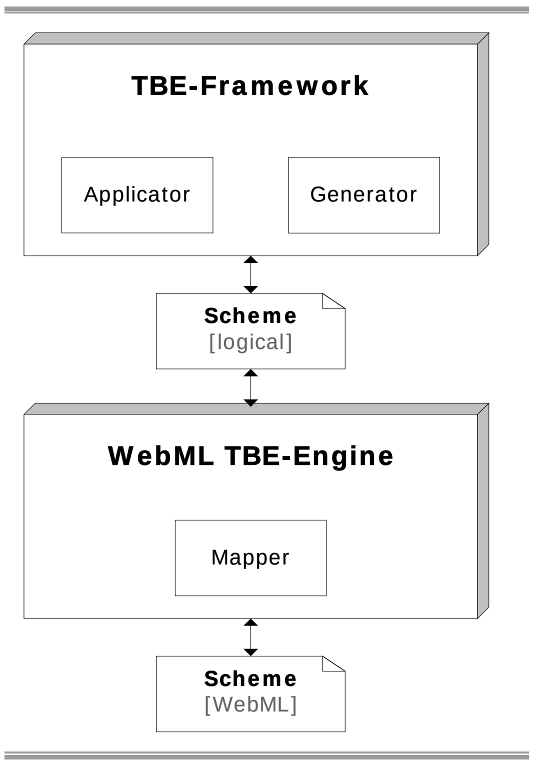


Figure 1.6: Framework for TBE-engines and the WebML TBE-engine.

Therefore we decided to implement a generic TBE-engine. The upper part of FIGURE 1.6 depicts the model-independent components of a generic TBE-engine that basically make up the TBE-framework, i.e. the `Generator` and the `Applicator`. We demonstrate the implementation of an `Applicator` that uses an XQuery statement for transforming the input scheme in logical representation into the output scheme, again in logical representation. However, other implementations of an `Applicator` could be easily plugged into the TBE-framework as well.

We demonstrate the application of the TBE-framework to WebML, i.e. how the model-dependent components are implemented for WebML. The lower part of FIGURE 1.6 depicts the resulting *WebML TBE-engine*. However, TBE-engines for other modelling languages than WebML could be easily implemented as well, just by implementing the (few) model-dependent components.

1.3 Outline of the thesis

Chapter 2 describes the process of model-driven web application development with modelling language WebML. Further, the XML representation of WebML schemes and WebRatio, the CASE tool for WebML, are illustrated.

Chapter 3 presents the concept of TBE. The chapter starts with a description of the logical representation of schemes, which is the basis for defining transformer's semantics. The remained of this chapter presents the formal semantics of transformers.

Chapter 4 demonstrates how transformers are defined and applied within WebRatio. The chapter particularly addresses how TBE-directives are annotated to WebML schemes in textual form. Further, the definition and application of a concrete transformer within WebRatio is shown.

Chapter 5 presents the architecture of the developed TBE-engine. The main purpose of this architecture is to separate model-dependent components from model-independent ones.

Chapter 6 presents the implementation of the TBE-engine. In particular the chapter explains the implementation of the TBE-framework and the WebML TBE-engine.

Chaper 7 presents related work. The chapter briefly describes other approaches to scheme transformers and illustrates TransEd, which is a graphical editor developed in a related diploma thesis for editing WebML schemes and graphically defining by-example transformers.

Chapter 8 concludes the thesis.

2 WebML

Contents

2.1 Development method 31

2.2 Modelling language 34

2.2.1 Data model 34

2.2.2 Hypertext model 36

2.3 XML representation of WebML schemes 37

2.4 CASE tool support – WebRatio 39

2.4.1 The level of support to the WebML development phases 39

2.4.2 A bird eye's view on WebRatio 40

This chapter introduces the conceptual web modelling language WebML. SECTION 2.1 gives an overview of the different phases of developing web applications with WebML. SECTION 2.2 describes the modelling language WebML itself. SECTION 2.3 illustrates the XML representation of WebML schemes. Finally, SECTION 2.4 briefly describes WebRatio, the CASE tool for WebML.

2.1 Development method

The development process offered by WebML consists of different phases, is inspired by Boehm's spiral model [Boe85] and covers all phases of a web application's life cycle from requirements analyses to maintenance and evolution. As depicted in FIGURE 2.1, this process is iterative and incremental where the various phases are repeated and refined until their results meet the particular requirements. Thus, a web application is developed in cycles, where the current version of the web application is first tested and evaluated and afterwards modified to cope with previously specified or newly emerged requirements in each iteration step. Subsequently the particular phases are illustrated by the development of the Conference Manangement Application (CMA) [MMCF03].

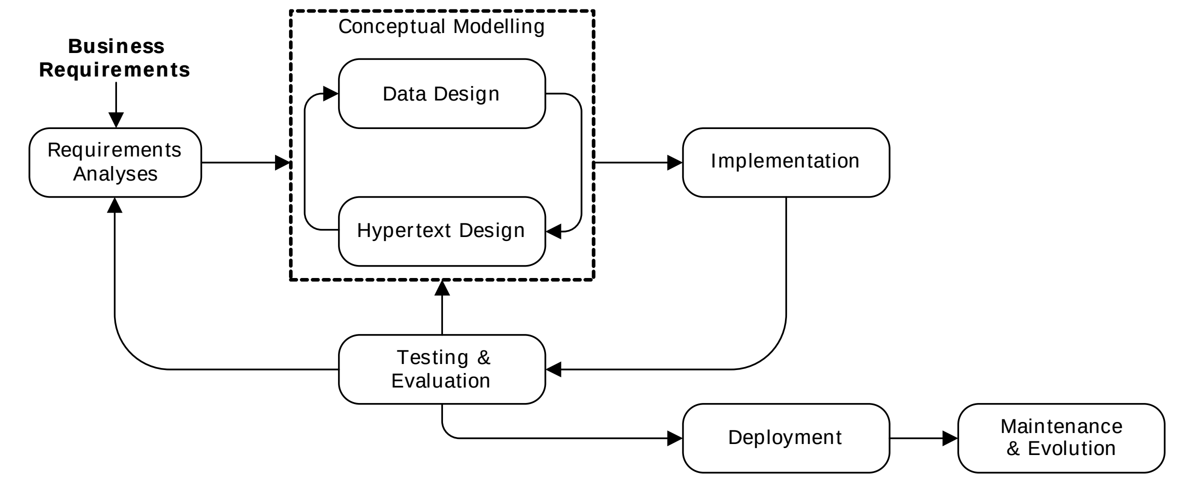


Figure 2.1: Phases in the WebML development process.

The WebML development process starts with the **Requirements Analyses** phase, where business requirements that motivate the application's development serve as input. This phase primary targets on identifying *groups of users* addressed by the web application and the *functions* that will be provided to each user group. Further, this phase aims at identifying *core information objects* representing the web application's content and decomposing the web application into *site views*, i.e. different user interfaces needed for different user groups or delivery devices.

Example 2.1*: There are two groups of users for the CMA, namely authors and members of the Program Committee (short: PC members). Authors shall be enabled to submit papers, and PC members shall be enabled to select those submitted papers that are accepted at the conference. The mentioned activities are examples for functions that are provided to the particular user group. It is straightforward to define core information objects for representing authors, papers and PC members. Further, it is reasonable to define one site view per user group, each presenting just the pieces of information needed by the users of the respective group. Moreover only those functions shall be provided by the user interface to each user group that can be executed by members of the respective group.*

Having obtained the specific requirements, the next phase in the WebML development process is **Conceptual Modelling**. The conceptual modelling language WebML, which is described in SECTION 2.2, is used for defining the conceptual scheme. The phase of conceptual modelling is of special interest within this diploma thesis as the prototypical implementation of TBE, can be used by developers for performing scheme transformations within this phase.

Conceptual modelling starts with *data design*, where core information objects that have been identified during requirements analyses are organized. Afterwards, the core information objects and functions, again identified during requirements analyses, are composed within hypertext pages. This activity is called *hypertext design*.

Data design and hypertext design strongly depend on each other, as only those information objects that are considered within data design can be used for hypertext design. Thus, if information objects are needed for designing the hypertext that were not designed so far, data has to be designed once more. The changes in data design may in turn require changes in hypertext design which may result in a loop between data design and hypertext design. Since it is likely that such loops occur they are explicitly represented within the development process, as FIGURE 2.1 depicts.

When the conceptual scheme has been defined the **Implementation** phase is entered. It is a distinctive target of WebML that the modelled web application can be automatically implemented through code generation. For this purpose WebML provides an XML [xml04] representation of WebML schemes. that can be processed from code generators. SECTION 2.3 illustrates this XML representation.

The quality of the web application's implementation is then improved in the **Testing & Evaluation** phase. If the quality of the web application is insufficient, without any changes in the analysed requirements, a cycle is initialized by re-entering the phase of conceptual modelling. If the quality of the web application is insufficient, because of changes in requirements a bigger cycle is initialized by newly analyzing requirements. These cycles are repeated until the implementation meets the distinctive requirements.

In the **Deployment** phase, the web application get's productive, i.e. the necessary web servers and database servers are configured. The phase of **Maintenance & Evolution** concludes the development process.

2.2 Modelling language

Conceptual modelling within the WebML development process comprises data design and hypertext design. WebML considers each of these activities in a separate model, named *data model* and *hypertext model*, respectively. These models are divided into several sub-models, where each defines a individual set of scheme elements for defining corresponding schemes. These sub-models are subsequently described, as far as needed in the context of this diploma thesis. For a detailed description of these models and their scheme elements confer to [webml04]. Note that the illustration of the individual models by the CMA example is limited to aspects concerning authors and that functions, provided to authors or PC members are neglected at all for reasons of simplicity.

2.2.1 Data model

The data model is divided into the *structure model* and the *derivation model.* The **structure model** describes the high-level organization of data. It is based on the Entity-Relationship model [Che76], and is compatible with class diagrams of the Unified Modelling Language [uml04]. Entity types, attributes and relationships are the scheme elements used for structure modelling. Entities are considered to be individually addressable by means of a unique identifier, which is represented by attribute OID that is implicitly defined at each entity type. The left part of FIGURE 2.2 depicts the content scheme of the CMA, which has already been introduced in the previous chapter. Yet, entity types are now extended with OID attributes. Relationships always connect exactly two entity types. The role of an entity type within a relationship is modelled by relationship roles. Further, the mapping of entity types, attributes and relationships to physical data structures stored in databases is done within structure modelling. However, this mapping can only be specified within the XML representation of a data model, as a graphical representation is inadequate.

WebML focuses on the development of personalized web applications and considers therefore users, groups and site views and their semantic relationships as *constituent* scheme elements of *each* data model, as depicted in the right part of FIGURE 2.2. Each user described by attributes username, password and email belongs to at least one group. Further, each user is associated with one group that acts as the default group for that user. Finally, each group has a target site view, which is served to the users of that group.

Example 2.2*: Users, groups and site views are represented within WebML by the consituent entity types* `user`*,* `group` *and* `site view`*, respectively. Thus, in order to represent the particular users, groups and site views of the* `CMA`*, the respective entity types have to be populated according to the users, groups and site views identified during requirements analyses.*

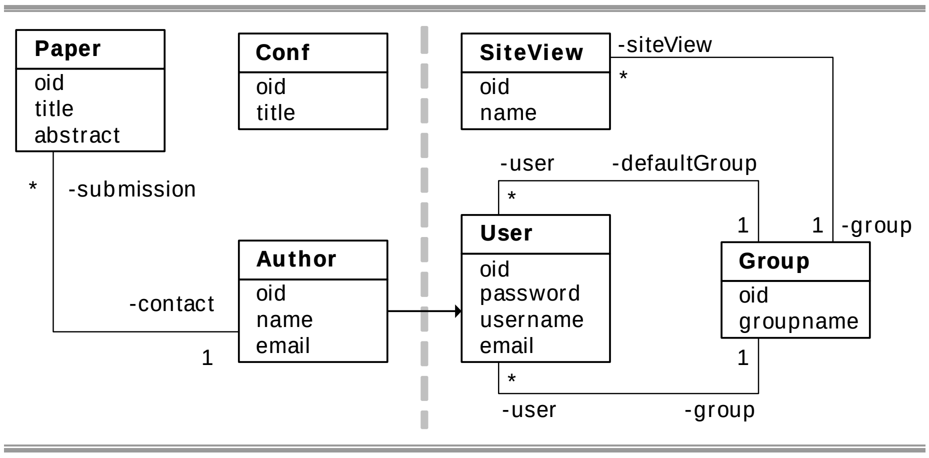


Figure 2.2: Extended content scheme of the CMA.

The **derivation model** provides for adding redundant information to the structure model, in order to augment the expressiveness of content schemes [MMCF03]. Derivation is formally expressed by queries, which apply to the scheme elements of the structure model, i.e. entity types, relationships and attributes. For this purpose WebML provides a set oriented, navigational query language, called WebML-OQL [webml04], which is inspired by OCL [WK98] and OQL [oql04].

Example 2.3*: Entity type* `Author` *depicted in F*IGURE *2.2 is derived from entity type* `User` *as denoted by the arrow pointing from entity type* `Author` *to entity type* `User`*. The WebML-OQL expression "*`User(as SuperEntity) where SuperEntity. defaultGroup.groupname = 'Author'`*" specifies the semantics of this derivation. This WebML-OQL expression determines that each user that states to be member of group* `author` *is selected and therefore considered as being an author. Thereby, it is decided whether a certain user states to be a member of group* `author` *by traversing relationship role* `defaultGroup` *of entity type* `User` *and comparing the value of attribute* `groupname` *with the constant string* `Author`*.*

2.2.2 Hypertext model

The hypertext model is divided into the *composition model* and the *navigation model*. The **composition model** describes the allocation of content to classes of hypertext pages (short: page classes). Composition modelling specifies which page classes make up the user interface, and which content elements make up a page class. The scheme elements used for composition modelling are *page classes* and several *content units*. Content units represent content scheme elements used for publishing the information described in the structure model. A content unit that lists all entities of a certain entity type is called *index unit*, e.g. index unit `AuthorIndex`. Typically, a content unit is associated with one underlying entity type, called *source entity type*, from which the content of the unit is computed. Data units, for example, are used for publishing a single entity. The respective entity to be published is specified by a selector condition. The site views offered to the different user groups are furthermore represented within the composition model, by means of an equally named scheme element.

The **navigation model** describes how page classes are linked to provide a navigable hypertext, and how content units inside a page are connected to permit the flow of context information. Several types of links are distinguished where the most important types are contextual links and non-contextual links. Links connecting content units are called contextual links, as they have to transport contextual information, e.g. the entity presented by a data unit at the time a link is traversed. Links connecting pages are called non-contextual links and do not transport information.

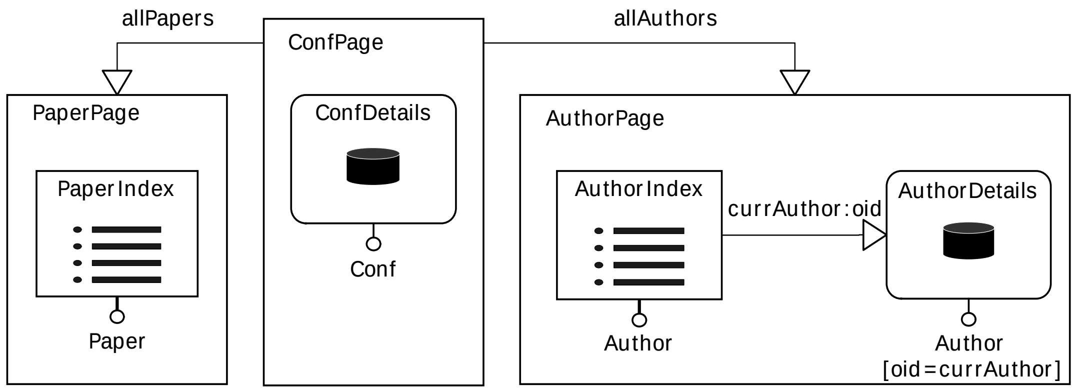


Figure 2.3: Extended hypertext scheme of the CMA.

***Example 2.4**: Basically, the page classes and content units depicted in* FIGURE *2.3 have been introduced in* CHAPTER *1. The additional content unit* `AuthorDetails`*, which is a detail unit, denotes that all information of a distinctive author is presented, as entity type* `Author` *is the source entity type of this detail unit. The selector condition* "`[oid = currAuthor]`" *specifies that the author with the* `oid` *specified by parameter* `currAuthor` *is to be presented. Thereby, parameter* `currAuthor` *is passed to detail unit* `AuthorDetails` *by the contextual link* `currAuthor`*. Thus, when a user traverses the link* `currAuthor`*, details about the certain author that is currently selected in the index of all authors are presented by detail unit* `AuthorDetails`*. Further by traversing the non-contextual links* `allPapers` *and* `allAuthors` *users navigate from the* `ConfPage` *to the* `PaperPage` *and* `AuthorPage`*, respectively.*

Note, that the hypertext scheme depicted in FIGURE 2.3 *is actually the site view of the* `CMA's` *content presented to authors, which is therefore called* `AuthorsSiteView`*. As WebML specifies no graphical representation for site views no such scheme element is presented in* FIGURE 2.3.

There is no separate model for specifying content management operations offered by WebML. Instead, as they are invoked as a side effect of navigation, operations are modelled within the hypertext model. WebML provides scheme elements, called operation units representing some primitive operations like data insertion, deletion or modification. The corresponding operation units are called create unit, delete unit and modify unit, respectively. Operation units always have an source entity type analogously to content units. The source entity type defines the set of entities, which is affected when the operation is executed. Furthermore, WebML provides for representing arbitrary operations by means of black boxes. Such black boxes do not rigidly prescribe the represented operation. Operation units are not illustrated for reasons of conciseness.

2.3 XML representation of WebML schemes

The main purpose of the XML representation of WebML schemes is to enable CASE tools to process WebML schemes. Within this diploma thesis the XML representation of WebML schemes is of special interest, since it is the interface between the WebML TBE-engine and WebRatio, i.e. the graphical editor for WebML schemes. The XML representation of WebML schemes is defined by the WebML DTD in its current version 3.0, which is divided into several sub DTDs as depicted in FIGURE 2.4.

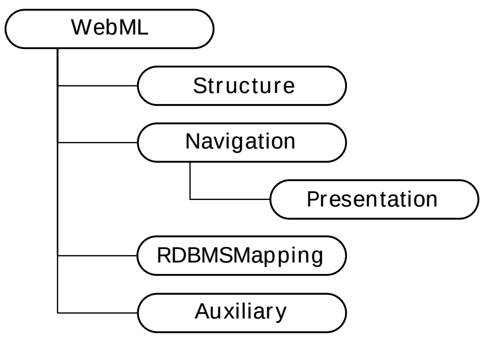


Figure 2.4: Sub DTDs comprised within the WebML DTD.

The **Structure DTD** and the **Navigation DTD** define the XML representations of sorts of scheme elements needed for data modelling and hypertext modelling, respectively. The **Presentation DTD**, which is a sub-DTD of the Navigation DTD, defines the XML representations of sorts of scheme elements needed for specifying the layout of page classes and content units. The **RDBMSMapping DTD** defines the XML representaitons of sorts of scheme elements needed for mapping entity types, attributes and relationships to physical, persistent data storages, e.g. databases. The **Auxiliary DTD** defines sorts of scheme elements needed for representing the graphical arrangement of scheme elements like, for example, entity types and page classes, within the respective WebML scheme.

```
<!ELEMENT ENTITY (ATTRIBUTE*, RELATIONSHIP*, PROPERTY*, ...)>
<!ATTLIST ENTITY
   id         ID     #REQUIRED
   name       CDATA  #IMPLIED
   ...
>
```

Figure 2.5: Structure DTD fragment declaring scheme elements of sort entity type.

***Example 2.5**: FIGURE 2.5 depicts a fragment of the Structure DTD that defines the XML representation of entity types. Entity types may contain attributes and relationships as well as scheme elements of sort PROPERTY. Such scheme elements can be defined at every scheme element and are used for annotating text. Note, that the WebML DTD specifies that the XML representation of each sort of scheme elements* *must comprise XML attribute `id` in order to uniquely address the respective scheme elements.*

```
<ENTITY id="ent1" name="Author" ...>
   <ATTRIBUTE id="att1" name="OID"/>
       ...
</ENTITY>
```

Figure 2.6: XML representation of entity type Author.

***Example 2.6**: The XML fragment depicted in FIGURE 2.6 represents entity type `Author`. It is shown that the name of this entity type is `Author` and that `ent1` is the identifier for this entity type. XML element `ATTRIBUTE` represents the `oid` attribute of entity type `Author`, which is identified by `att1`.*

2.4 CASE tool support – WebRatio

The Web Ratio Site Development Studio is a commercial CASE tool supporting WebML. SECTION 2.4.1 discusses WebRatio's level of support to the different phases of the WebML development cycle. SECTION 2.4.2 gives a bird eye's view on the usage of WebRatio.

2.4.1 The level of support to the WebML development phases

For defining schemes within the phase of *Conceptual Modelling*, WebRatio provides an graphical editor as a kind of fundamental support. Further, WebRatio provides for checking the structural validity of WebML schemes, i.e. WebRatio checks whether all information, necessary for generating the source code of the respective web application, has been defined by the modeller.

The *Implementation* phase is supported by a code generator allowing developers to implement the modelled web application without writing a single line of code. Thereby, web applications may be published under the Tomcat web server and JSP engine but also over the Microsoft .NET architecture.

WebRatio supports the phase of *Testing & Evaluation*, i.e. it provides for testing and evaluating realistic prototypes. For this purpose WebRatio enables the population of data sources with randomly generated data, taken either from a default population or from attribute-specific populations.

WebRatio supports the *Deployment* phase by means of automatically configuring the runtime environment, i.e. the web servers and database servers. Yet, for the phase of *Maintenance & Evolution* and for the phase of *Requirements Analyses* no particular support is provided by WebRatio.

2.4.2 A bird eye's view on WebRatio

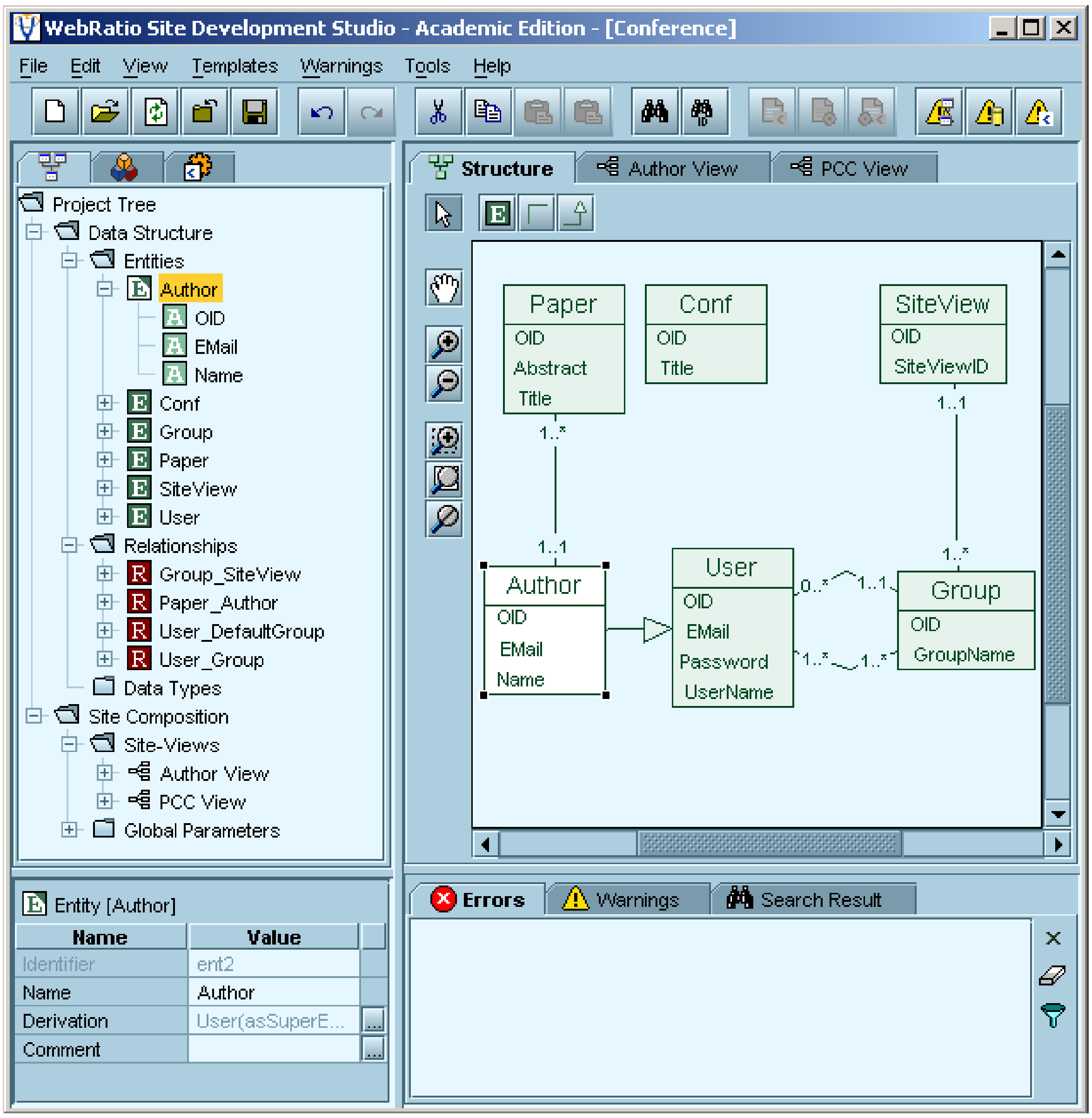
FIGURE 2.7 depicts a screen shot of the user interface of WebRatio, taken while developing the CMA example application. The workspace of Web Ratio is divided into four areas, which are subsequently described.

The **Project Tree** area is depicted in the upper left part of FIGURE 2.7. The project tree comprises three tabs. The first tab, which is selected in FIGURE 2.7, displays all scheme elements in hierarchical order. The second tab is used for specifying the mapping of scheme elements to physical data structures and the third tab is used for specifying the presentation details of page classes and content units. Thereby, style sheets can be specified that render page classes. Further, the arrangement of content units within the respective page classes can be specified.

The **Work Area** is depicted in the upper right part of FIGURE 2.7 and is used for graphically defining scheme elements. The Work Area comprises one tab for editing the data scheme and one tab per site view for editing the hypertext scheme.

The **Properties Frame,** depicted in the bottom left part of FIGURE 2.7, enables developers to specify properties of scheme elements that can not expressed graphically in a reasonable manner. One such property is, for example, a WebML OQL expression that precisely specifies the derivation of an entity type. The properties frame, which is depicted in Figure 2.7, shows the properties of entity type `Author`.

The **Message Area,** depicted in the bottom right part of Figure 2.7, displays errors and warnings identified during the structural validation of a particular scheme.



Entity [Author]

Name	Value
Identifier	ent2
Name	Author
Derivation	User(asSuperE...
Comment	

Figure 2.7: Screnn shot of the user interface of WebRatio.

3 Transformers by example

Content

3.1 The logical representation of schemes 43

3.2 Formal semantics of transformers 45

3.2.1 Query templates 46

3.2.2 Generative templates 52

3.3 Transformers and their application 58

This chapter describes the basic concepts of TBE by means of presenting the formal specification of TBE [Lec04]. The TBE-system presented in this diploma thesis implements these basic concepts of TBE.

The remainder of this chapter is organized as follows. SECTION 3.1 describes the logical representation of schemes, which is the basis for defining the semantics of transformers in terms of TBE, which is described in SECTION 3.2. Finally, SECTION 3.3 explains the application of transformers.

3.1 The logical representation of schemes

The logical representation of a scheme is a *representation* of the scheme grounding on the consideration of schemes as being a set of scheme elements and a set of connections between those scheme elements. Note, that each scheme, neutral of the modelling language used for defining the scheme, can be represented in such a manner. The logical representation of schemes considers the distinctive sorts of scheme elements as *universes* and the distinctive sorts of connections as *relations*.

DEFINITION 3.1: UNIVERSES AND RELATIONS. The logical representation of a scheme is given by a set of universes $\check{U}$ and a set of relations $\check{R}$. Each universe $U \in \check{U}$ represents a sort U of scheme elements. Each relation $R(U_1, \ldots, U_n)$, with $R \in \check{R}$ and, for $i = 1 \ldots n$, $U_i \in \check{U}$, represents connections between scheme elements of sort U_i.

The term "scheme element" used in DEFINITION 3.1 may cause some confusion as it has different meanings in the context of WebML and in the context of the logical representation of schemes. In the context of WebML, the term "scheme element" belongs to modelling constructs like, for example, entity types or page classes that may be more precisely described by properties like, for example, names of entity types or names of page classes. In the context of the logical representation, the term "scheme element" subsumes both instances of modelling constructs, e.g. entity types, *and* properties of such scheme elements, e.g. names of entity types. Throughout this diploma thesis the term "scheme element" denotes scheme elements in the sense of the logical representation of schemes. Whereas, the term "WebML scheme element" denotes instances of modelling constructs.

***Example 3.1**: The formal specification of TBE determines universes E, A, P, I and N for representing entity types, attributes of entity types, page classes, index units and names, respectively. Relation $name(N \times (E \cup A \cup P \cup I))$ expresses that entity types, attributes, page classes and index units have names. Further, relation $defAt((A \times E) \cup (I \times P))$ expresses that each attribute is defined at an entity type and that each index unit is defined at a page class. Each index unit has a content source, which is expressed by relation $source(I \times E)$. The sets of universes and relations defined in the formal specification of TBE for representing a subset of all scheme elements of WebML are thus given by $\check{U} = \{E, A, P, I, N\}$ and $\check{R} = \{name, defAt, source\}$, respectively.*

Members of universes and members of relations represent the particular scheme elements and connections between scheme elements of a concrete scheme, respectively.

DEFINITION 3.2: UNIVERSE MEMBERS AND RELATION MEMBERS. Each scheme element e is member of exactly one universe $U \in \check{U}$ and is dented as $e{:}U$. A tuple of a relation, called relation member, $R(U_1, \ldots, U_n)$ is denoted as $R\langle e_1, \ldots, e_n\rangle$, with $R \in \check{R}$ and, for $i = 1 \ldots n$, $e_i \in U_i$.

XML representation	*Logical representation*	
	Universe members	*Relation members*
`<ENTITY id="ent1"`	`ent1: E`	
`name="Author">`	`Author: N`	`name⟨ent1, Author⟩`
`<ATTRIBUTE id="att1"`	`att1: A`	`defAt⟨att1, ent1⟩`
`name="OID"/>`	`OID: N`	`name⟨att1, OID⟩`
`</ENTITY>`		

Table 3.1: XML representation (left) and logical representation (right) of entity type author and its embedded attribute OID.

Example 3.2: *The left hand side of* TABLE *3.1 depicts the XML representation of entity type* `ent1` *and its embedded attribute* `att1`*. The middle part of* TABLE *3.1 depicts the corresponding universe members. Entity type* `ent1` *and attribute* `att1` *are represented by universe members* `ent1` *and* `att1`*, declaring to be members of universe E and A, respectively. Universe members* `Author` *and* `OID`*, denoting to be members of universe N, represent the names of entity type* `ent1` *and attribute* `att1`*, respectively. The relation members representing the connections between these scheme elements are depicted in the right hand side of* TABLE *3.1. Relation member* `name<ent1,Author>` *represents the connection of entity type ent1 to its name. Relation member* `name<att1,OID>` *is interpreted analogously. Relation member* `defAt<att1,ent1>` *expresses that attribute* `att1` *is defined at entity type* `ent1`*.*

3.2 Formal semantics of transformers

Transformers are defined by "example" schemes and applied to schemes of a particular modelling language `L`. The key idea for achieving the independency of modelling language `L` is to use the logical representation of schemes for defining and applying transformers, instead of using the internal representation of schemes in notation of `L` directly.

Consequently, a transformer's query template specifies which *tuples of universe members* are selected from the *logical representation of a scheme.* Analogous, a transformer's generative template specifies which *universe members* and *relation members* are to be generated.

In the following an excerpt of the formal specification of TBE defining the semantics of transformers is presented. For each template it is started with explaining its formal semantics. Then it is shown how the template's formal representation is derived from its graphical one.

3.2.1 Query templates

A query template specifies a query that, when applied to the logical representation of a scheme, selects tuples of universe members therefrom. The semantics of such queries is specified by modellers via defining an generic "example" in notation of a certain modelling language. Such "examples" are subsequently referred to as the *native representation* of query templates, since this is the representation of query templates, they are initially defined in. However, the formal semantics of query templates is specified by means of variables and constraints. The representation of a query template in terms of variables and constraints, will subsequently be referred to as its *formal representation*.

3.2.1.1 Formal semantics

A query template comprises variables and several sorts of constraints. Variables are placeholders for members of particular universes. The particular universe a variable is capable of holding members therefrom is the domain of this variable. Further, variables are separated into result variables and non-result variables, which are both needed for evaluating the query template. In difference to non-result variables, result variables represent the query template's result.

DEFINITION 3.3: QUERY TEMPLATE. A query template is denoted as $Q(V_r,V_{nr},dom,C)$, where V_r and V_{nr} are disjoint sets of result and non-result variables, respectively. The total function $dom:(V_r \cup V_{nr}) \rightarrow \check{U}$ associates to each variable $v \in V = (V_r \cup V_{nr})$ a universe $U \in \check{U}$ serving as domain for v. C is a set of constraints over variables in V. The domain $U \in \check{U}$ of a variable $v \in V$ is denoted by $v:U$. [LS04]

TBE separates comparison constraints and membership constraints. A comparison constraint restricts the resulting set of universe members by comparing their identifiers to identifiers of other universe members or literal values. A membership constraint specifies relation members that have to exist in a scheme in order to fulfill this constraint. Comparison constraints and membership constraints are simple constraints. Complex constraints can be built from other constraints using common logical connectives.

DEFINITION 3.4: CONSTRAINTS. Each constraint $c \in C$ is either simple or complex. A *simple constraint* has one of the following two forms:

1) a *comparison constraint* has the form v_1 op (v_2 | lit), where "|" is a BNF-symbol denoting alternatives, $v_1, v_2 \in V$, op $\in$ {=, !=}, and lit being a literal value.

2) a *membership constraint* has the form $\langle v_1, ..., v_n\rangle \in R$, with $R \in \check{R}$ and, for $i=1...n$, $v_i \in V$.

Complex constraints can be built from other constraints (simple or complex) using the logical connectives ($\wedge$, $\vee$, $\neg$) in the usual manner.

The meaning of queries is defined in terms of domain relational calculus (DRC), which is, for example, explained in [RG00]. The principle of evaluating a DRC expression is to calculate the cartesian product of all result variables, which results in one tuple of result variables for each possible combination. Those tuples of result variables that fulfill all specified constraints are the result of the DRC expression. Thereby, non-result variables are needed for deciding whether a tuple of result variables fulfills a particular constraint or not.

DEFINITION 3.5: MEANING OF QUERIES. A query Q (V_r, V_{nr}, dom, C) is interpreted as a DRC expression of the form $\{ \langle v_{r1}, ..., v_{rx}\rangle \in \text{dom}(v_{r1}) \times ... \times \text{dom}(v_{rx}) \mid \exists v_{nr1} \in \text{dom}(v_{nr1}), ..., \exists v_{nry} \in \text{dom}(v_{nry})\ c_1 \wedge ... \wedge c_z \}$. Thereby, $v_{r1}, ..., v_{rx}$ and $v_{nr1}, ..., v_{nry}$ represent the sets of result and non-result variables V_r and V_{nr}, respectively, and $c_1 ... c_z$ represent the set C of constraints.

3.2.1.2 Deriving the formal representation from the graphical one

SECTION 1.1.2 has illustrated the graphical notation of query templates by the query template of transformer IndexPCForET. This illustration abstracted from variables representing identifiers of scheme elements for reasons of conciseness. Such variables are subsequently called ID-variables. TBE determines a certain graphical notation for ID-variables, which is illustrated in EXAMPLE 3.3.

Example 3.3: *FIGURE 3.1 depicts the precise graphical notation of the query template of transformer IndexPCForET, which has already been introduced in SECTION 1.1.2. Variables ENT and ATT represent names of entity types and names of attributes,* *respectively. The grey shaded variable `ENT_ID` and `ATT_ID` is an ID-variable representing identifiers of entity types and identifiers of attributes, respectively. ID-variable `ENT_ID` is a result variable because it is preceded by a tick. ID-variable `ATT_ID` is a non-result variable. Therefore, the precise semantics of this query template is to select pairs of entity types, i.e. identifiers of entity types, and names of entity types. Thereby, only those pairs are selected, where the name is that of the respective entity type. Further, only those entity types are selected which comprise at least one attribute. This is expressed by non-result variables `ATT` and `ATT_ID`.*

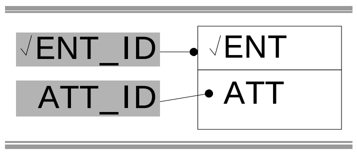


Figure 3.1: Precise graphical notation of transformer `IndexPCForET´s` query template.

A query template T_Q in notation of a modelling language L, is a scheme S in notation of L extended by TBE-directives. The formal representation of a query template T_Q is derived from its graphical representation by analyzing the logical representation of scheme S and the respective TBE-directives. Subsequently, the logical representation of a scheme S is regarded as the logical representation of a query template T_Q.

logical representation	*formal representation*	
universe members	variables	*implicit*
relation members	membership constraints	*implicit*
-	comparison constraints	*explicit*
-	complex constraints	*explicit*

Table 3.2 Approach to the derivation of a query template's formal representation from its logical one.

TABLE 3.2 summarizes the approach to the derivation of a query template's formal representation from its logical one, which is explained in the following. Universe members are implicitly interpreted as variables and relation members are implicitly interpreted as membership constraints. Yet, comparison constraints and complex constraints are explicitly added to the formal representation of a query template as specified by corresponding TBE-directives. EXAMPLE 3.4 illustrates the logical representation of transformer `IndexPCForET's` query template. EXAMPLE 3.5 illustrates how the logical representation of this query template is interpreted in order to derive the query template's formal representation.

XML representation	*Logical representation*	
	Universe members	*Relation members*
`<ENTITY id="ENT_ID"`	`ENT_ID: E`	
`name="ENT">`	`ENT: N`	`name⟨ENT_ID, ENT⟩`
`<ATTRIBUTE id="ATT_ID"`	`ATT_ID: A`	`defAt⟨ATT_ID, ATT⟩`
`name="ATT"/>`	`ATT: N`	`name⟨ATT_ID, ENT_ID⟩`
`</ENTITY>`		

Table 3.3: Logical representation of transformer `IndexPCForET´s` *query template.*

Example 3.4: *Reconsider that the graphical notation of transformer* `IndexPCForET's` *query template is actually a WebML scheme and comprises therefore WebML scheme elements. These WebML scheme elements are depicted in the left hand side of TABLE 3.3, in terms of XML. The right hand side of TABLE 3.3 depicts the universe members and relation members that logically represent these WebML scheme elements. Thus, the right hand side of TABLE 3.3 shows the logical representation of transformer* `IndexPCForET's` *query template.*

Variables		*Membership constraints*
result	*non-result*	
`ENT_ID: E`	`ATT_ID: A`	`⟨ENT_ID, ENT⟩ ∈ name`
`ENT: N`	`ATT: N`	`⟨ATT_ID, ENT_ID⟩ ∈ defAt`
		`⟨ATT_ID, ATT⟩ ∈ name`

Table 3.4: Formal representation of transformer `IndexPCForET´s` *query template.*

Example 3.5: *TABLE 3.4 depicts the formal representation of transformer* `IndexPCForET's` *query template, derived from its logical representation. Considering that variables are derived from universe members the derivation of variable* `ENT_ID`*,* `ENT`*,* `ATT_ID` *and* `ATT` *is self-explanatory. The separation of these variables into result variables and non-result variables is achieved by analyzing the ticks, defined in the query* *template's graphical notation. For example, variable* `ENT_ID`*, as depicted in* FIGURE *3.1, is preceded by a tick and therefore a result variable. Variable* `ATT_ID` *is not preceded by a tick and therefore a non-result variable.*

3.2.1.3 Evaluation of query templates

This section illustrates how a query in terms of DRC, derived from the formal representation of a query template, is evaluated. EXAMPLE 3.6 illustrates the DRC expression derived from transformer `IndexPCForET's` query template. EXAMPLE 3.7 illustrates the evaluation of this DRC expression within the `CMA` content scheme.

```
{
   ⟨ENT_ID, ENT⟩ ∊ E x N |
   ⟨ENT_ID, ENT⟩ ∊ name ∧
   ⟨ENT_ID, ATT_ID⟩ ∊ defAt
}
```

Figure 3.2: Transformer `IndexPCForET´s` *query template in terms of DRC.*

Example 3.6: FIGURE *3.2 depicts transformer* `IndexPCForET´s` *query template by means of domain relational calculus. This DRC expression is interpreted as follows. The first line expresses that all tuples of result variables* `⟨ENT, ENT⟩` *of domains* `E` *and* `N` *are selected, i.e. all possible combinations of entity types and names. The membership constraint depicted in the second line states that only those pairs of entity types and names are selected where the name is that of the respective entity type. The membership constraint depicted in the third line restricts the pairs of entity types and names to those, where the respective entity type defines at least one attribute.*

Cartesian product	*Constraints*		*Result*
	c_1	c_2	
⟨ent1, Conf⟩	-	✓	-
⟨ent1, Author⟩	✓	✓	✓
⟨ent1, Page⟩	-	✓	-
⟨ent2, Conf⟩	-	✓	-
⟨ent2, Author⟩	-	✓	-
⟨ent2, Page⟩	✓	✓	✓
⟨ent3, Conf⟩	✓	✓	✓
⟨ent3, Author⟩	-	✓	-
⟨ent3, Page⟩	-	✓	-

Table 3.5: Evaluation of transformer `IndexPCForET´s` *query template within the* `CMA`*'s content scheme.*

Example 3.7*: TABLE 3.5 illustrates the evaluation of transformer* `IndexPCForET´s` *query template to the* `CMA` *scheme, i.e. to the logical representation of the* `CMA` *scheme. Note, that entity types* `User`*,* `Site View` *and* `Group` *are neglected within this example, as they are not required for the purpose of illustration. The column labelled "Cartesian product" lists all combinations of entity types and names defined in the* `CMA` *scheme. Reconsider that the* `CMA` *scheme defines entity types* `Author`*,* `Paper` *and* `Conf`*, which are identified by* `ent1`*,* `ent2` *and* `ent3`*, respectively. Therefore the logical representation of the* `CMA` *scheme, which is illustrated in TABLE 3.1, comprises universe members* `ent1`*,* `ent2` *and* `ent3` *of domain* `E` *representing entity types and universe members* `Author`*,* `Paper` *and* `Conf` *of domain* `N` *representing the names of entity types. The column labelled "Constraints" shows whether or not a particular tuple of an entity type and a name fulfills constraints* `⟨ENT_ID, ENT⟩ ∊ name` *(c_1) and/or* `⟨ENT_ID, ATT⟩ ∊ defAt` *(c_2). Thereby, the tick denotes that the particular tuple holds the constraint whereas the hyphon denotes the opposite. Tuple* `⟨ent1, Author⟩`*, for example, fulfills constraint c_1 because* `Author` *is the name of entity type* `ent1`*. This constraint is, for example, not fulfilled by tuple* `⟨ent1, Conf⟩`*. The evaluation of the remaining tuples is interpreted analogously. Membership constraint* `c2` *is fulfilled from every tuple, as every entity type defines at least one attribute. The result of the evaluation of transformer* `IndexPCForET´s` *query* *template is depicted in the column labelled "Result". Thereby, the result of the evaluation comprises tuples* $\langle$`ent1, Author`$\rangle$, $\langle$`ent2, Page`$\rangle$ *and* $\langle$`ent3, Conf`$\rangle$.

3.2.2 Generative templates

A generative template specifies the generation of new scheme elements and new connections between scheme elements. SECTION 3.2.2.1 describes the formal semantics of generative templates. SECTION 3.2.2.2 explains, how the formal representation of a generative template is derived from its graphical one.

3.2.2.1 Formal semantics

A generative template comprises parameter variables, new-element variables and relation constructors. Parameter variables represent existing scheme elements whereas new-element variables denote scheme elements to be generated. Relation constructors specify relations between scheme elements to be generated.

DEFINITION 3.6: GENERATIVE TEMPLATE. A generative template $G(P,V_g,R_g)$ is given by a set P of parameter variables, a set Vg of new element variables, and a set R_g of constructors for relations between elements of P and V_g. [LS04]

Parameter variables represent scheme elements that are already defined within the scheme when the generative template is instantiated. Each parameter variable is capable of representing scheme elements of exactly one sort.

DEFINITION 3.7: PARAMETER VARIABLE. Each parameter variable $p \in P$ represents a given scheme element of a universe $U \in \check{U}$. This is denoted as $p:U$.

Relation constructors specify that a connection of a particular sort, e.g. $defAt(A \times E)$, is generated between scheme elements. Thereby the connected scheme elements have to be represented by either parameter variables or new element variables.

DEFINITION 3.8: RELATION CONSTRUCTOR. Each relation constructor $r_g \in R_g$ specifies the creation of a new relation between scheme elements. It has the form $R\langle x_1, \ldots, x_n\rangle$ with $R \in \check{R}$ and, for $i = 1 \ldots n$, $x_i \in (P \cup V_g)$.

When a generative template is instantiated, for each new element variable the represented scheme element is generated by means of generating a corresponding universe member. Generating a universe member actually means to compute a new identifier out of the respective domain. For this purpose the formal specification of TBE determines three types of construction expressions specifying the computation of new identifiers. One type of construction expressions, called new construction expression, specifies the generation of a new identifier out of a particular universe. Constant construction expressions specify the identifier to be generated by means of assigning a constant literal value. Function construction expressions assign the result of a function call. One such function is for example `concat(N x N)`, which assigns the result of string concatenation of the names passed as arguments.

DEFINITION 3.9: NEW-ELEMENT VARIABLE & CONSTRUCTION EXPRESSION. Each new element variable $v_g \in V_g$ represents a scheme element of a universe $U \in \check{U}$ to be generated and has, for that purpose, a construction expression `exp` attached. This is denoted as v_g`:U = exp`. A construction expression has one of the following forms:

1) "`new U`" assigns a new ID out of universe `U`,
2) "`lit`" assigns a literal value, and
3) "$\texttt{func}(x_1, \dots, x_n)$" assigns the result of a function call with, for $i = 1 \dots n$, $x_i \in (P \cup Vg \cup lit)$.

DEFINITION 3.10: INSTANTIATION OF GENERATIVE TEMPLATES. A generative template `G` is instantiated within a scheme `S` by binding each parameter variable of `G` to an equally sorted scheme element of `S`. Such an instantiation is denoted as `G[S,B]`, whereby `B` is a set of parameter bindings. A binding of a parameter variable `p` to a scheme element `e` is denoted as "`p=e`". When an instantiation `G[S,B]` is processed, new scheme elements and new relations are generated within `S` as follows:

1) for each new element variable $v_g : U$, its attached construction expression is evaluated and the result becomes a member of universe `U`
2) for each relation constructor $R\langle x1, \dots, xn\rangle$, a relation of sort `R` is established between the scheme elements represented by variables $x_1, \dots, x_n$.

3.2.2.2 Deriving the formal representation from the graphical one

In SECTION 1.1.2 the graphical notation of generative templates has been illustrated by the generative template of transformer `IndexPCForET`. Again, this illustration abstracted from ID-variables. Therefore, FIGURE 3.3 illustrates the precise graphical notation of transformer `IndexPCForET's` generative template.

***Example 3.8**: FIGURE 3.3 depicts the precise graphical notation of transformer `IndexPCForET's` generative template. New-element variable `PC` represents the name of a page class, whereas the page class itself is represented by ID-variable `PC_ID`. Since ID-variable `PC_ID` is a new-element variable construction expression `PC_ID = new(P)` is attached. ID-variable `IU_ID` is structured analogously. Variable `ENT` representing the name of the content source of index unit `IU` is a parameter variable. Consequently, the corresponding ID-variable is a parameter variable, too.*

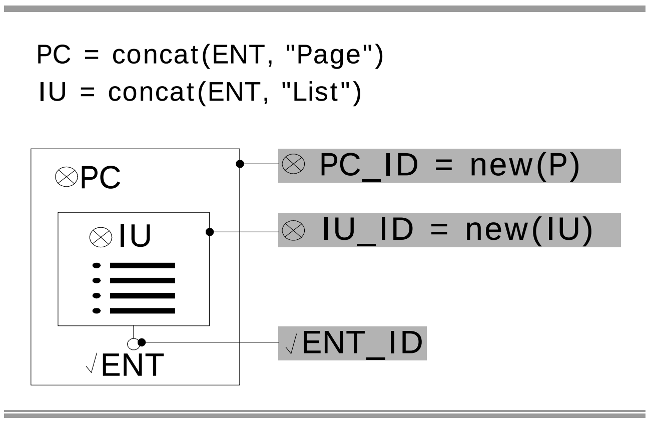


Figure 3.3: Precise graphical notation of transformer `IndexPCForET's` generative template.

A generative template T_G in notation of a modelling language L, is a scheme S in notation of L extended by TBE-directives. The formal representation of a generative template T_G is derived from its graphical representation by analyzing the logical representation of scheme S and the respective TBE-directives. Subsequently, the logical representation of a scheme S is regarded as the logical representation of a generative template T_G.

logical representation	*formal representation*	
universe member	variables	*implicit*
relation member	relation constructor	
-	new construction expression	
-	function construction expression	*explicit*
-	constant construction expression	

Table 3.6 Approach to the derivation of a generative template's logical representation from its formal one.

TABLE 3.2 summarizes the approach to the derivation of a generative template's formal representation from its logical one, which is explained in the following. Universe members are implicitly interpreted as variables and relation members are implicitly interpreted as relation constructors. Yet, construction expressions are explicitly added to the formal representation of a generative template as specified by corresponding TBE-directives. Thereby, if no construction expression is specified for a particular new-element variable, a new construction expression is attached to the respective new-element variable by default. EXAMPLE 3.9 illustrates the logical representation of transformer `IndexPCForET's` generative template. EXAMPLE 3.10 illustrates how the logical representation of this generative template is interpreted in order to achieve the generative template's formal representation.

XML representation	*Logical representation*	
	Universe members	*Relation members*
`<ENTITY id="ENT_ID"`	`ENT_ID: E`	
`name="ENT"/>`	`ENT: N`	
`<PAGE id="PC_ID"`	`PC_ID: P`	
`name="PC">`	`PC: N`	`name⟨PC_ID, PC⟩`
`<INDEXUNIT id="IU_ID"`	`IU_ID: I`	`defAt⟨IU_ID, PC_ID⟩`
`name="IU"`	`IU: N`	`name⟨IU_ID, IU⟩`
`entity="ENT_ID"/>`		`source⟨IU_ID, ENT_ID⟩`
`</PAGE>`		

Table 3.7: Logical representation of transformer `IndexPCForET´s` generative template.

***EXAMPLE* 3.9**: *The graphical notation of transformer `IndexPCForET's` generative template, which is depicted in FIGURE 3.3, is basically a WebML scheme and comprises therefore WebML scheme elements. The left part of TABLE 3.7 depicts these WebML scheme elements in terms of XML. The right right part of TABLE 3.7 depicts the universe members and relation members that logically represent these WebML scheme elements. Thus, the right hand side of TABLE 3.7 depicts the logical representation of transformer `IndexPCForET's` generative template.*

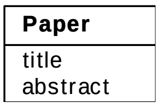
Variables		*Relation constructors*
parameter variables	*new-element variables*	
`ENT_ID: E`	`PC_ID = new (P)`	`name⟨PC_ID, PC⟩`
`ENT: N`	`PC = concat(ENT, 'Page')`	`name⟨IU_ID, IU⟩`
	`IU_ID = new (IU)`	`defAt⟨IU_ID, PC_ID⟩`
	`IU = concat(ENT, 'List')`	`source⟨ENT_ID, IU_ID⟩`

Table 3.8: Formal representation of transformer "IndexPCForET´s" generative template.

***EXAMPLE* 3.10**: *TABLE 3.8 depicts the formal representation of transformer `IndexPCForET´s` generative template derived from its logical representation depicted in TABLE 3.7. Thereby parameter variable `ENT_ID` is derived from universe member `ENT_ID`. Analogous, parameter variable `ENT` is derived from universe member `ENT`. New-element variables `PC` and `IU` are derived from universe members `PC` and `IU`, respectively. New-element variables `PC_ID` and `IU_ID` are analogously derived from the corresponding universe members. The construction expressions are attached to the new-element variables as specified by corresponding TBE-directives that are depicted in FIGURE 3.3. The derivation of relation constructors depicted in the right part of TABLE 3.8 from the relation members depicted in TABLE 3.7 is straightforward and not further explained.*

3.2.2.3 Instantiation of generative templates

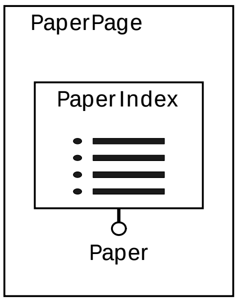
This section illustrates the instantiation of generative templates by the example of transformer `IndexPCForET´s` generative template.



universe members	*relation members*
`ent1: P`	`name⟨ent1, Paper⟩`
`att1: A`	`name⟨att1, title⟩`
`att2: A`	`name⟨att2, abstract⟩`
`Paper: N`	`defAt⟨att1, ent1⟩`
`title: N`	`defAt⟨att2, ent1⟩`
`abstract: N`	

Figure 3.4: Sample input for transformer `IndexPCForET´s` generative template.

Example 3.11: *This example illustrates the instantiation of transformer `IndexPCForET´s` generative template to a scheme defining one entity type called `Paper`. The left part of FIGURE 3.4 depicts the graphical representation of this scheme. The right part of FIGURE 3.4 depicts its logical representation. The representations of this scheme are not further described, as they are subsets of the representations of the `CMA` scheme already explained. Transformer `IndexPCForET's` generative template (short: G) takes two parameters, i.e. an entity type and a name represented by parameter variables `ENT_ID` and `ENT`, respectively. Therefore an instantiation of this generative template within the scheme (short: S) depicted in FIGURE 3.4 would look like `G[S, {ENT_ID = ent1, ENT = Paper}]`. This instantiation specifies, that the generative template `G` is instantiated within `S` by binding parameter variable `ENT_ID` to entity type `ent1` and parameter variable `ENT` to the name `Paper`.*



universe members	*relation members*
`page1: P`	`name⟨page1, PaperPage⟩`
`inu1: I`	`name⟨inu1, PaperIndex⟩`
`PaperPage: N`	`defAt⟨inu1, page1⟩`
`PaperIndex: N`	`source⟨inu1, ent1⟩`

Figure 3.5: Sample output of transformer `IndexPCForET´s` generative template.

EXAMPLE 3.12: *FIGURE 3.5 depicts the output of the generative template instantiation illustrated in EXAMPLE 3.11. This output is depicted in graphical representation (left) and logical representation (right) in FIGURE 3.5. Universe member `page1` is generated according to new element variable `PC_ID`. and therefore representing a* *page class. Index unit `inu1` is generated analogously. Universe member `PaperPage`, representing the name of page class `page1`, is generated according to new-element variable `PC`, which has the construction expression `concat(ENT,'Page')` attached. Thereby, the identifier of universe member `PaperPage` is computed by concatenating literal "`Page`" to the name of the universe member bounded to parameter variable `ENT`, which is `Paper`. The name `PaperIndex` is generated analogously. Relation constructor `name⟨PC,PCN⟩` determines the generation of relation member `name⟨page1,PaperPage⟩`, since new element variable `PC_ID` has the value `page1` and new element variable `PC` has value `PaperPage`. The generation of the remaining relation members works analogously.*

3.3 Transformers and their application

In the formal specification of TBE a transformer is a combination of a query template and a generative template. The parameters needed as input for the generative template are provided by the result variables of the query template.

DEFINITION 3.11: TRANSFORMER DEFINITION. A transformer is a proper combination of a query template Q and a generative template G and is denoted as T(Q, G). A combination of Q and G is proper, if

1) for each result variable of Q there exists an equally named and equally sorted parameter variable of G, and
2) all parameter variables of G are provided by Q.

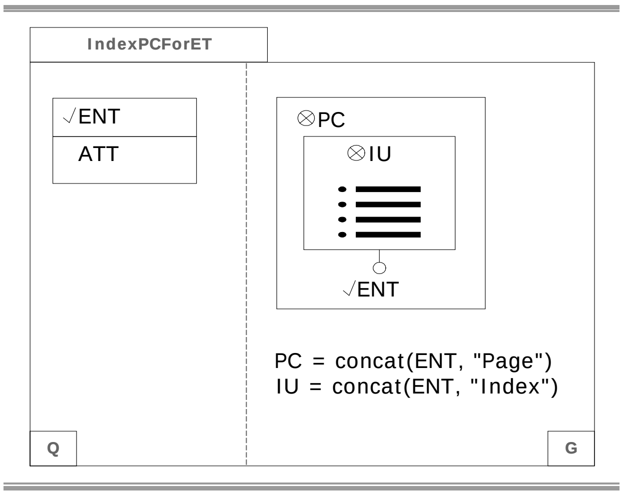


Figure 3.6: Transformer `IndexPCForET`.

***Example 3.13**: Transformer `IndexPCForET` is depicted in FIGURE 3.6. It properly combines the query template and the generative template, as result variables `ENT_ID` and `ENT` of the query template properly match parameter variables `ENT_ID` and `ENT` of the generative template, respectively. The semantics of transformer `IndexPCForET` should be self-explanatory since the corresponding templates have been described in the previous sections.*

When a transformer is applied, its generative template is iteratively instantiated for each tuple of result variables selected by the query template. The application of a transformer within a scheme can be parameterized by *application-specific constraints* and *application-specific construction expressions*. The meaning of application-specific constraints and application-specific construction expressions is equal to that of constraints and constructions expressions, respectively, that are specified at definition time of the transformer. The major difference between application-specific constraints and "conventional" constraints is, that application-specific constraints take only effect in one particular transformer application, whereas "conventional" constraints take effect in every application of the transformer. This is the same for application-specific construction expressions. Application-specific constraints further restrain variables of the query template while application-specific construction expressions override construction expressions of the generative template.

DEFINITION 3.12: TRANSFORMER APPLICATION. An application of a transformer T(Q, G) to a scheme S is denoted as T[S, ASC, ASE]. Thereby ASC is a set of application specific constraints and ASE is a set of application specific construction expressions. Both sets may be empty.

Transformer	*Scheme*	*ASC*	*ASE*
IndexPCForET	S	{ ENT != "Conf", ENT != "User", ENT != "SiteView", ENT != "Group" }	{}

Table 3.9: Exemplary application of transformer `IndexPCForET`.

***Example 3.14:** Reconsider the CMA content scheme depicted in FIGURE 2.2 where entity types `Author`, `Paper`, `Conf`, `User`, `SiteView` and `Group` are defined. With one* *individualized application of transformer* `IndexPCForET` *to the* `CMA` *content scheme, page classes* `AuthorPage` *and* `PaperPage`*, depicted in* F*IGURE 2.3, are generated, as desired. The corresponding transformer application is depicted in* T*ABLE 3.9. Transformer* `IndexPCForET` *is applied to scheme* `S`*, which is the* `CMA` *content scheme. The application-specific constraints depicted in the third column of* T*ABLE 3.9 achieves that page classes are not generated for entity types* `Con`*,* `User`*,* `SiteView` *and* `Group`*, which is clearly desired. The set of application-specific construction expressions is empty in this example.*

4 Defining and applying transformers within WebRatio

Contents

4.1 Textual representation of TBE-directives 62

4.1.1 General approach for annotating TBE-directives 62
4.1.2 TBE-directives *tag result variable* and *tag parameter variable* 64
4.1.3 TBE-directive *constraint* 65
4.1.4 TBE-directive *construction expression* 66

4.2 Customized-directives 66

4.2.1 Customized-directive *alias* 67
4.2.2 Customized-directive *anchor* 68

4.3 Defining transformer *IndexPCForET* 69

4.3.1 Defining the query template 69
4.3.2 Defining the generative template 72
4.3.3 Compiling the transformer definition 74

4.4 Applying transformer *IndexPCForET* 75

4.4.1 Defining the input scheme 75
4.4.2 Transforming the input scheme 76
4.4.3 Viewing the output scheme 76

This chapter demonstrates how to define and apply transformers within WebRatio. Since WebRatio is used off-the-shelf TBE-directives have to be annotated in textual form to schemes and templates. SECTION 4.1 specifies the textual representation of TBE-directives and discusses a general approach for annotating such directives.

SECTION 4.2 introduces directives that are necessary for defining transformers within WebRatio in addition to TBE-directives. These directives are called *customized-directives* since they are custom for defining transformers within WebRatio. When the TBE-engine compiles a transformer definition, customized-directives are translated into TBE constructs. Therefore, customized-directives do not extend the semantics of TBE.

SECTION 4.3 presents a use case for defining a transformer within WebRatio by the example of transformer `IndexPCForET`. SECTION 4.4 shows how this transformer is applied to the `CMA` scheme, again within WebRatio.

4.1 Textual representation of TBE-directives

SECTION 4.1.1 presents a general approach for annotating TBE-directives in textual representation to WebML schemes within WebRatio. In TBE symbols "√" and "⊗" are used for separating result variables from non-result variables and for separating parameter variables from new-element variables, respectively. These TBE-directives are subsequently called `tag result variable` and `tag parameter variable`, respectively. The syntax of their textual representation is specified in SECTION 4.1.2. The syntax of TBE-directives `constraint` and `construction expression` in textual representation is specified in SECTION 4.1.3 and SECTION 4.1.4, respectively.

4.1.1 General approach for annotating TBE-directives

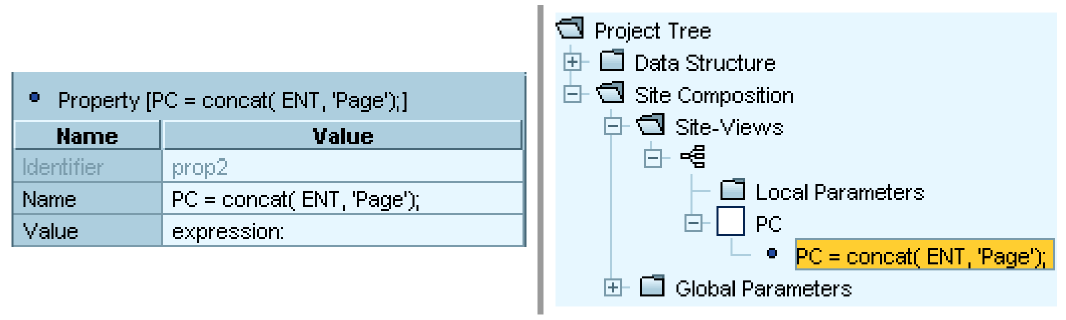
The general syntax of a TBE-directive in textual representation is specified by the EBNF [Wir77] expression depicted in FIGURE 4.1. The textual representation of a directive starts with the determination of its name, like, for example, `expression` for denoting construction expressions. The name of a TBE-directive is merely needed for separating the different TBE-directives when they are processed within the TBE-engine. After an obligatory colon the actual directive in textual form is expected, like, for example, `PC = concat(ENT, 'Page')`, which is a construction expression. An obligatory semicolon finishes each directive.

For annotating directives to schemes or templates, WebML scheme elements of sort property (*property scheme elements*) are used. Property scheme elements may be defined at every scheme element and are used for annotating arbitrary text to WebML schemes. In order to separate properties of scheme elements, like, for example, names of entity types from scheme elements of sort property, the latter ares subsequently referred to as property scheme elements. Thus, if a certain directive is to be specified, a new property scheme element has to be added to the scheme or template.

```
directive  := name ":" text_rep ";".
name       := string.
text_rep   := string.
string     := ('a' ... 'z' | 'A' ... 'Z' | ' ' | '-' | '$' | '_')
              {'a' ... 'z' | 'A' ... 'Z' | ' ' | '-' | '$' | '_'}.
```

Figure 4.1: General syntax of directives in textual representation.

For entering the name and the actual text-representation of a TBE-directive the following policy must be adhered to. The *name of a directive* has to be entered as the *value of the property scheme element* together with the obligatory colon. The *textual representation of a directive* together with the obligatory semicolon has to be entered as the *name of the property scheme element*. This policy for entering the name and the textual representation of a directive achieves a clear visualization of the respective textual representation within the project tree of WebRatio, as illustrated in FIGURE 4.2.



● Property [PC = concat(ENT, 'Page');]

Name	Value
Identifier	prop2
Name	PC = concat(ENT, 'Page');
Value	expression:

Figure 4.2: Annotating directives to WebML schemes.

***Example 4.1**: The left hand side of FIGURE 4.2 depicts the annotation of construction expression `PC = CONCAT(ENT, 'PAGE');` to the generative template of transformer `INDEXPCFORET`, by means of a property scheme element. The text-representation of this construction expression is entered as the name of the property scheme element. Therefore it is achieved that the text-representation of the construction expression is visualized in the project tree of WebRatio, as depicted in the right hand side of FIGURE 4.2.*

4.1.2 TBE-directives *tag result variable* and *tag parameter variable*

The syntax of the textual representations of TBE-directives `tag result variable` and `tag parameter variable` is specified by the EBNF expression depicted in FIGURE 4.3. The textual representation of TBE-directive `tag result variable` starts with the determination of the name of the TBE-directive, i.e. `resultVariable`. After the obligatory colon the name of the variable to be tagged as result variable is expected. The textual representation of TBE-directive `tag result variable` finishes with the obligatory semicolon. The syntax of TBE-directive `tag parameter variable` is structured analogously.

```
result_variable    := "resultVariable:" variable ";".
parameter_variable := "parameterVariable:" variable ";".

variable := string.
```

Figure 4.3: Syntax of TBE-directives result variable and parameter variable in textual representation.

When defining a template modellers frequently tag variables. In order to reduce efforts for annotating TBE-directives `tag result variable` and `tag parameter variable` in textual form a shortcut notation is introduced for specifying these TBE-directives. This shortcut notation is a dollars sign that preceds the name of the variable to be tagged. Thus, if the name of a variable defined within a query template is preceded by a dollars sign, the respective variable is tagged as result variable. Analogous, if the name of a variable defined within a generative template is preceded by a dollars sign, the respective variable is tagged as parameter variable. All variables that are not tagged are consequently non-result variables or new-element variables, depending on whether they are defined in a query template or a generative template, respectively.

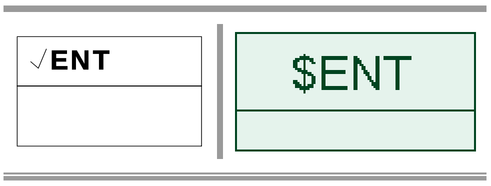


Figure 4.4: Graphical representation (left) and textual representation (right) of TBE-directive `tag result variable`.

***Example 4.2**: The left hand side of FIGURE 4.4 depicts parts of transformer `IndexPCForET´s` query template with graphically defined TBE-directives. Variable*

`ENT` is tagged as result variable since its name is preceded by symbol "√". The right hand side of FIGURE 4.4 depicts again variable `ENT`. Yet, symbol "√" is now represented in textual form by means of the dollars sign preceding the variable's name.

4.1.3 TBE-directive *constraint*

Modellers specify constraints at definition time of transformers and also at their application time for individualizing the application. TBE distinguishes three types of constraints, i.e. comparison constraints, complex constraints and membership constraints. Since membership constraints are implicitly given by the relation members of a query template in logical representation, only comparison constraints and complex constraints are explicitly specified by modellers. Therefore, the syntax of TBE-directive `constraint` in textual representation, which is depicted in FIGURE 4.5, specifies the syntax of a textual representation of comparison constraints and complex constraints.

```
constraint  := "constraint:" (comparison | complex) ";".
comparison  := variable comp_type (literal | variable).
complex     := log_conn "{" (comparison | complex)+ "}".
variable    := string.
comp_type   := ["!"] "=".
log_conn    := "or" | "and".
literal     := "'" string "'".
```

Figure 4.5: Syntax of TBE-directive `constraint` in textual representation.

The textual representation of a comparison constraint starts with the name of the variable to be constrained followed by the type of comparison. The terminal symbol "=" denotes a comparison of type `equals`. Preceding this terminal symbol by terminal symbol "!" denotes a comparison of type `not equals`. Afterwards, the name of the variable or the literal value used for comparison is expected. For distinguishing literal values from variable names, the former have to be entered within quotes.

The textual representation of a complex constraint starts with the logical connective to be used. After the determination of the logical connective, other complex constraints or comparison constraints follows in curly brackets.

4.1.4 TBE-directive *construction expression*

TBE distinguishes three types of construction expressions, i.e. constant construction expressions, new construction expressions and function construction expressions. Each construction expression can be specified at definition time and at application time of a transformer. The syntax of TBE-directive `construction expression` in textual representation is depicted in FIGURE 4.6.

```
expression  := "expression:" variable "=" (constant|function|new)";".
constant    := literal.
new               := "new" universe.
function    := "(" argument {"," argument} ")".
argument    := (variable | literal).
literal     := "'" string "'".
variable    := string.
universe    := string.
```

Figure 4.6: Syntax of TBE-directive `construction expression` in textual representation.

The textual representation of a construction expression starts with the determination of its name. Next, the name of the new-element variable to which the construction expression is to be attached has to be specified. After an equal symbol the actual construction expression is expected.

For specifying a constant construction expression a literal value is expected. For specifying a new construction expression character sequence "new" followed by the name of the universe for which a new member is to be generated has to be specified. For specifying a function construction expression after the name of the function that is to be called, a sequence of arguments separated by commas is to be specified within paranthesis. Thereby, an argument is either the name of a variable or a literal value.

4.2 Customized-directives

This section introduces customized-directives, i.e. directives that are specifically required for defining transformers within WebRatio. SECTION 4.2.1 discusses customized-directive `alias`, which is used for giving variables individual names. SECTION 4.2.2 discusses customized-directive `anchor`, which enables the automatic arrangement of newly generated scheme elements.

Customized-directives are annotated to WebML schemes in the same manner as TBE-directives in textual representation. Thus, the general syntax of customized-directives is the same as the general syntax of TBE-directives, which has been specified in SECTION 4.1.1.

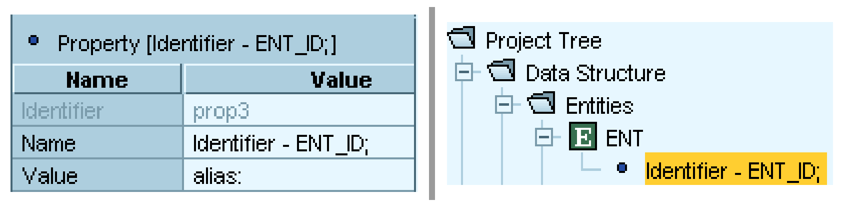
4.2.1 Customized-directive *alias*

For giving variables individual names, it is required to edit the name of the respective scheme element. Since WebRatio prevents editing some kinds of properties of scheme elements, like, for example, cardinalities of relationship-roles, a generic way of specifying *aliases* for non-editable properties is required. Thereby, the alias can then be used for referencing to the variable, like, for example, from within constraints or construction expressions. For the purpose of specifying aliases customized-directive `alias` is introduced. The syntax of the textual representation of customized-directive alias is depicted in FIGURE 4.7.

```
alias := "alias:" prop_name "-" alias_name ";".
alias_name  := string.
propy_name  := string.
```

Figure 4.7: Syntax of customized-directive `alias`.

The syntax of customized-directive alias is simple. After the name of the property for which an alias is to be defined an obligatory hyphen, followed by the actual alias is expected.



● Property [Identifier - ENT_ID;]

Name	Value
Identifier	prop3
Name	Identifier - ENT_ID;
Value	alias:

Figure 4.8: Specifying customized-directive `alias`.

***Example 4.3**: FIGURE 4.8 illustrates customized-directive `alias` by the specification of alias `ENT_ID` for the `Identifier` property of entity type `ENT`. The left part of FIGURE 4.8 depicts this directive by means of a property scheme element. The right part of FIGURE 4.8 shows that the respective alias is specified for the `Identifier`* *property of entity type* `ENT`*, since the corresponding property scheme element is appended to this entity type.*

4.2.2 Customized-directive *anchor*

WebRatio administrates the arrangement of scheme elements by storing the x-coordinate and y-coordinate for each scheme element. When modellers define schemes in WebRatio the positions of scheme elements are implicitly specified by their graphical arrangement.

When a generative template is instantiated within a scheme, new scheme elements are generated. Newly generated scheme elements clearly require certain x-coordinates and y-coordinates in order to be arranged. For generating coordinates of new scheme elements the following alternatives are possible:

- *Using default-values for coordinates*: The first alternative is to use default-values for the coordinates of new scheme elements, like, for example coordinates 0/0. This alternative has the major drawback that new scheme elements are huddled together, which is clearly not desired.

- *Specifying the generation of coordinates explicitly*: The second alternative is that modellers explicitly specify the generation of coordinates of new scheme elements by means of new-element variables representing the coordinates together with appropriate construction expressions. This means that for each coordinate of a new scheme element, a respective new element variable has to be specified by the modeller and additionally a construction expression, which specifies the computation of the value of the respective coordinate. This alternative causes additional efforts for modellers and is therefore not desired.

- *Generating coordinates implicitly*: The third alternative is that new scheme elements are automatically arranged relative to a certain reference point, i.e. relative to an existing scheme element. This reference point is subsequently called anchor. This means that each time a scheme element is generated, its coordinates are generated implicitly, i.e. it is arranged in relation to the anchor. The relative arrangement from new scheme elements to the anchor is derived from the arrangement of scheme elements in the generative template. Therefore, one scheme element of the generative template is marked as anchor. The relative arrangement of new-element variables to the anchor determines also the relative arrangement of new scheme elements at the time the transformer is applied. The scheme element that is marked as anchor, has to represent a parameter variable. Otherwise the anchor scheme element is not available in the scheme, the respective generative template is instantiated within.

Since the most convenient alternative for generating coordinates for modellers is to specify an anchor, customized directive `anchor` is introduced.

```
anchor    := "anchor:" variable ";".
variable  := string.
```

Figure 4.9: Syntax of customized-directive `anchor`*.*

The syntax of this customized-directive, which is depicted in FIGURE 4.9, is fairly simple and therefore not further explained.

4.3 Defining transformer *IndexPCForET*

This section demonstrates how to define transformer `IndexPCForET` within WebRatio. For defining the templates of this transformer the modeller has to create one WebRatio project, i.e. WebML scheme, per template. For defining variables within a template the modeller has to define scheme elements, according to the respective variable. For example, in order to define a new-element variable PC _ID specifying the generation of a new page class, the modeller has to define a page class within WebRatio.

The definition of transformer `IndexPCForET´s` query template and generative template is demonstrated in SECTION 4.3.1 and SECTION 4.3.2, respectively. In SECTION 4.3.3 it is described how to compile transformer `IndexPCForET` with the TBE-engine.

4.3.1 Defining the query template

For defining the query template of transformer `IndexPCForET` variables `ENT`, `ENT_ID` and `ATT_ID` and corresponding directives need to be defined. The semantics of these variables has been described in detail in the CHAPTER 3. Subsequently, the tasks necessary for defining these variables are demonstrated step by step.

Step 1: *Open a new WebRatio project*: Defining a template starts with opening a new WebRatio project. FIGURE 4.10 shows the work area of a new WebRatio project, containing the constituent entity types `User`, `Group` and `SiteView`. Note, that constituent scheme elements do not affect the semantics of a template since they are neglected by the TBE-engine when the transformer definition is compiled.

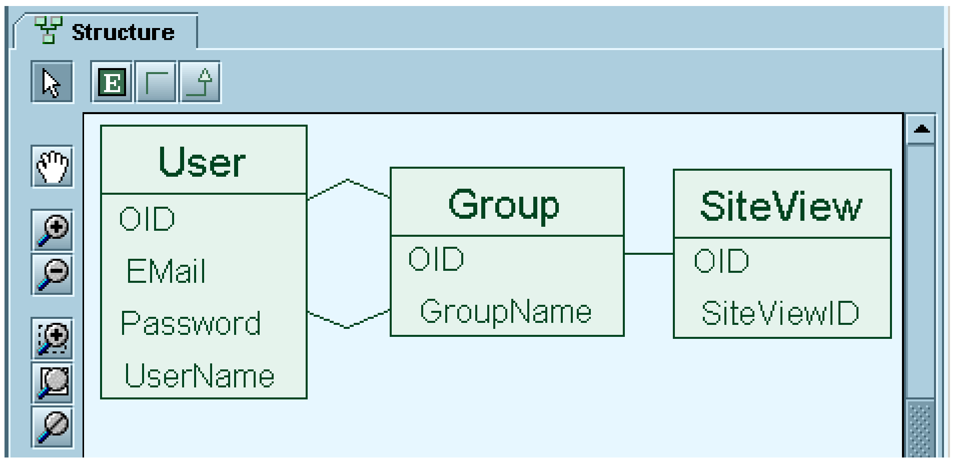
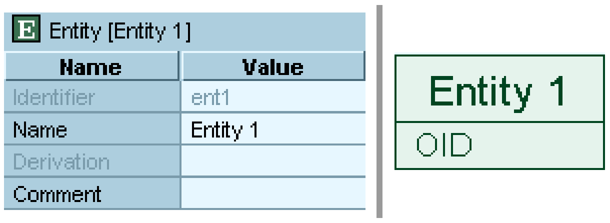


Figure 4.10: WorkArea of WebRatio when a new project is opened.

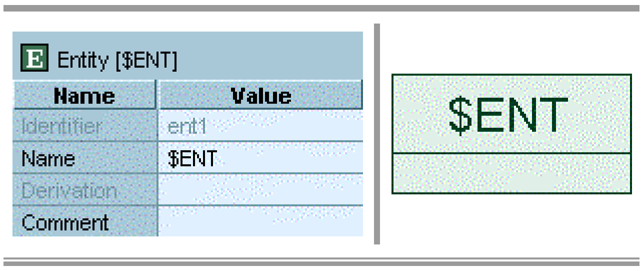
Step 2: *Define result variables* `ENT` *and* `ENT_ID`: In order to define result-variables `ENT_ID` and `ENT` add a new entity type. The properties of this entity type, i.e. entity type `ent1`, are depicted in the left part of FIGURE 4.11. The entity type `ent1`, as depicted in the right part of FIGURE 4.11, defines the constituent attribute `OID`. Again, the TBE-engine neglects this constituent scheme element when it compiles a transformer definition such that attribute `OID` does not effect the semantics of transformer `IndexPCForET´s` query template.


Entity [Entity 1]

Name	Value
Identifier	ent1
Name	Entity 1
Derivation	
Comment	

Figure 4.11: A newly added entity type (right) and its properties (left).

In order to tag variable `ENT`, result-variable enter `$ENT` at the `Name` property of entity type `ent1`. FIGURE 4.12 depicts variable `ENT` in the right hand side and the properties of this variable in the left hand side.

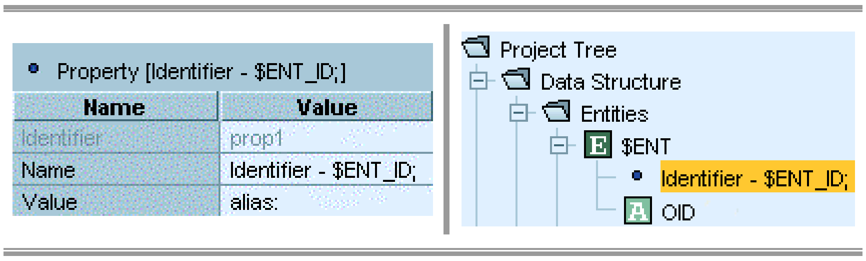


Entity [$ENT]

Name	Value
Identifier	ent1
Name	$ENT
Derivation	
Comment	

Figure 4.12: Result variable `ENT` *(right) and its properties (left).*

The customized-directive `alias: Identifier - $ENT_ID;` has to be specified in order to define result variable `ENT_ID`. This customized-directive achieves that the `Identifier` property of entity type `ent1` is set to "`ENT_ID`" when the TBE-engine compiles the query template, which finally results in generating variable `ENT_ID`. Further, since the name of variable `ENT_ID` is preceded by a dollars sign, it is tagged as result variable. The left part of FIGURE 4.13. depicts this alias-directive by means of a property scheme element. The right part of FIGURE 4.13. depicts the same customized-directive visualized within the project tree of WebRatio.

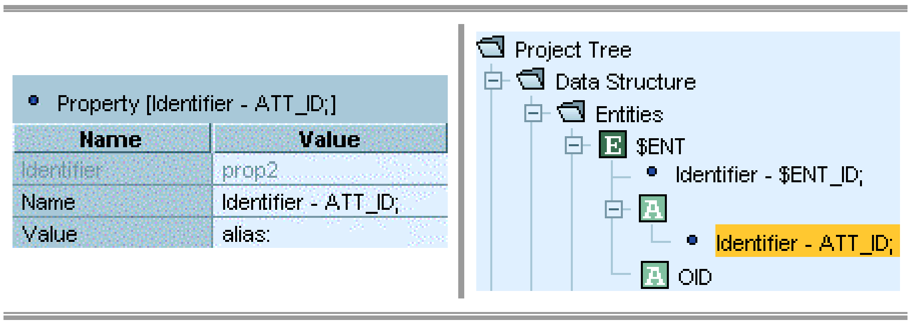


Property [Identifier - $ENT_ID;]

Name	Value
Identifier	prop1
Name	Identifier - $ENT_ID;
Value	alias:

Figure 4.13: Definition of directive `alias: Identifier - $ENT_ID;`.

<u>Step 3:</u> Define non-result variable `ATT_ID`: In order to define non-result variable `ATT_ID` add a new attribute to entity type `ent1` and set the `Name` property to an empty string. Afterwards define the alias `ATT_ID` by specifying directive `alias: Identifier - ATT_ID;` at entity type `ent1`. The outcome of these activities is shown in FIGURE 4.14.



● Property [Identifier - ATT_ID;]

Name	Value
Identifier	prop2
Name	Identifier - ATT_ID;
Value	alias:

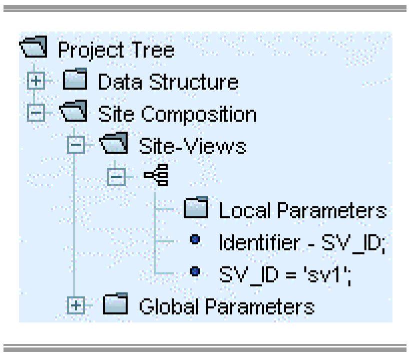
```
Project Tree
  Data Structure
    Entities
      $ENT
        ● Identifier - $ENT_ID;
        A
          ● Identifier - ATT_ID;
        A OID
```

Figure 4.14: Definition of directive `alias: Identifier - ATT_ID;`.

<u>*Step 4:*</u> *Store the query template*: In order to pass the query template to the TBE-engine for compiling transformer `IndexPCForET`, store the query template, i.e. the WebRatio project. Use `IndexPCForET-QT` as the name for the project.

4.3.2 Defining the generative template

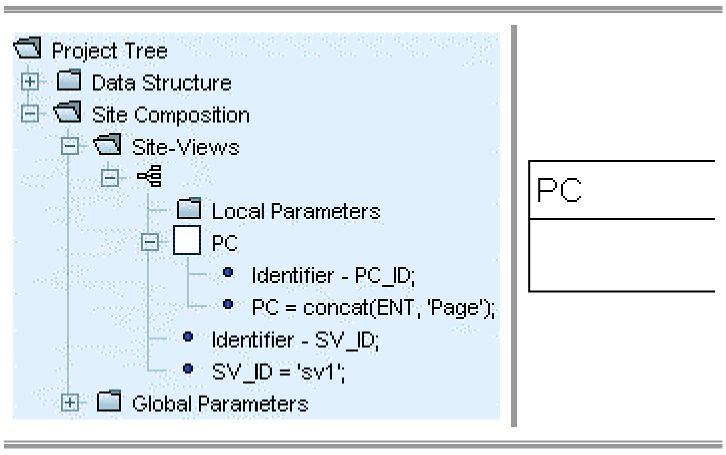
For defining the generative template of transformer `IndexPCForET` variables `PC`, `PC_ID`, `IU` and `IU_ID` and corresponding directives need to be defined. Again, the semantics of these variables has been described in detail in the CHAPTER 3. Subsequently, the tasks necessary for defining these variables are demonstrated step by step.



```
Project Tree
  Data Structure
  Site Composition
    Site-Views
      
        Local Parameters
        ● Identifier - SV_ID;
        ● SV_ID = 'sv1';
    Global Parameters
```

Figure 4.15 New-element variable `SV_ID` *and corresponding directives.*

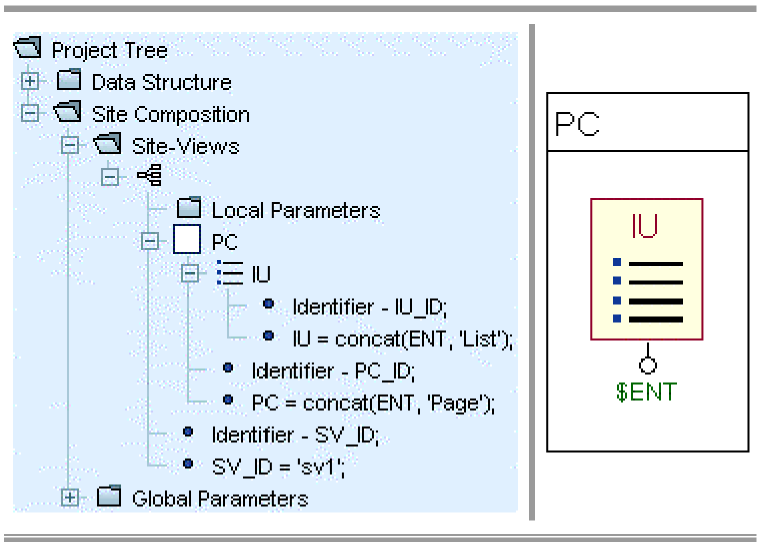
<u>*Step 5:*</u> *Define parameter variables* `ENT_ID` *and* `ENT`: In order to define the generative template of transformer `IndexPCForET` repeat `step 1` and `step 2`. By repeating these steps one has opened a new WebRatio project and additionally specified the parameter variables `ENT_ID`, `ENT`. The project tree after these initial steps looks like the one depicted in the right part of FIGURE 4.14.



Project Tree
Data Structure
Site Composition
Site-Views
Local Parameters
PC
Identifier - PC_ID;
PC = concat(ENT, 'Page');
Identifier - SV_ID;
SV_ID = 'sv1';
Global Parameters

Figure 4.16: New-element variables `PC` *and* `PC_ID` *and corresponding directives.*

<u>*Step 6:*</u> *Define parameter variables* `PC` *and* `PC_ID`: For defining these variables it is required to add a new page class since variable `PC_ID` represents page classes. As discussed in CHAPTER 2, WebML determines that each page class has to be defined within a site view. Therefore, add a new site view and the directives `alias: Identifier - SV_ID;` and `SV_ID = 'sv1';` to this site view, as depicted in FIGURE 4.15. When the generative template of transformer `IndexPCForET` is instantiated a new site view with identifier `sv1` will be generated and new page classes will be added to this site view.

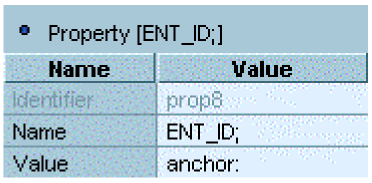


Project Tree
Data Structure
Site Composition
Site-Views
Local Parameters
PC
IU
Identifier - IU_ID;
IU = concat(ENT, 'List');
Identifier - PC_ID;
PC = concat(ENT, 'Page');
Identifier - SV_ID;
SV_ID = 'sv1';
Global Parameters

Figure 4.17 New-element variables `IU` *and* `IU_ID` *and corresponding directives.*

In order to define new-element variables `PC` and `PC_ID`, add a new page class to the generative template. This page class, which has identifier `page1`, is shown in the right part of FIGURE 4.16. Additionally, a directive specifying the alias for the `Identifier` property of page class `page1` and a directive, specifying the construction expression for new-element variable `PC`, is required. Therefore, specify the directives `alias: Identifier - PC_ID;` and `expression: PC = concat(ENT, 'Page');`. The left part of FIGURE 4.16 depicts these directives and variables.

<u>*Step 7:*</u> *Define new-element variables* `IU` *and* `IU_ID`: In order to define new-element variables `IU` and `IU_ID` add a new index unit to page class `page1` and set the content source of this index unit to "`$ENT`". The outcome of this activity is the index unit defined at page class `page1` as depicted in the right part of FIGURE 4.17. Analogous, to the definition of new-element variable `PC` and `PC_ID`, directives need to be additionally specified. Thereby, add directive `alias: Identifier - IU_ID;` to new element variable `IU`. Finally, add directive `expression: IU = concat(ENT, 'List');` to new-element variable `IU`.


Property [ENT_ID;]

Name	Value
Identifier	prop8
Name	ENT_ID;
Value	anchor:

Figure 4.18: Customized-directive `anchor`.

<u>*Step 8:*</u> *Define customized-directive anchor*: In SECTION 4.2.2 it has been argued that it is required to specify the arrangement of newly generated scheme elements by means of customized-directive `anchor`. Therefore, specify directive `anchor: ENT_ID;` at any scheme element of the generative template. This directive achieves, that when transformer `IndexPCForET` is applied all newly generated scheme elements will be arranged relative to the scheme element, represented by parameter variable `ENT_ID`. FIGURE 4.18 depicts this directive.

<u>*Step 9:*</u> *Save the generative template*: Analogous to the query template also the generative template needs to be stored. Use `IndexPCForET-GT` as the name for the WebRatio project.

4.3.3 Compiling the transformer definition

The query template and the generative template of transformer `IndexPCForET` need to be compiled into a transformer definition in terms of TBE in order to be applied. Since

WebRatio is used off-the-shelf as graphical editor for the prototype TBE-system this compilation cannot be triggered from within WebRatio. Instead, the TBE-engine has to be invoked from the command line. The arguments for such an invocation, which is in detail described in the source code documentation of the TBE-engine, are (1) the paths to the templates, stored on the local file system, (2) the name of the transformer and (3) the name of the modelling language, the templates are defined in.

4.4 Applying transformer *IndexPCForET*

This section shows, how to apply transformer `IndexPCForET` to the CMA content scheme. SECTION 4.4.1 illustrates the initial state of the CMA content scheme. SECTION 4.4.2 describes the invocation of the TBE-engine for the purpose of performing the scheme transformation. Last, SECTION 4.4.3 shows the result of the transformer application, i.e. the CMA hypertext scheme.

4.4.1 Defining the input scheme

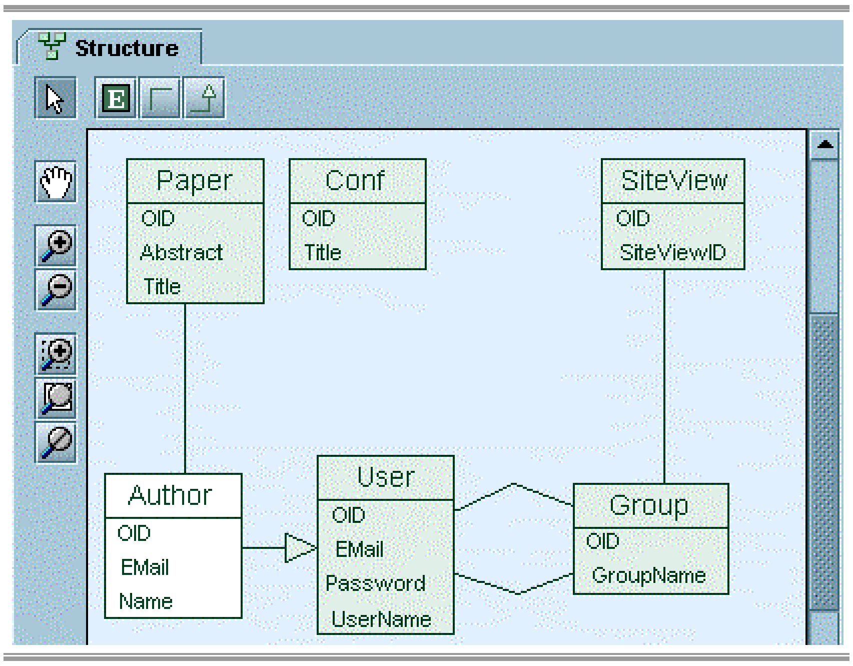


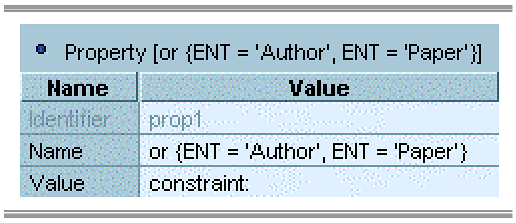
Figure 4.19: Content scheme of the CMA web application.

The CMA content scheme is used as the input scheme for the application of transformer `IndexPCForET`. Since defining the input scheme requires no activities, which are specific to TBE, only the result of defining the content scheme of the `CMA` web application is illustrated. The respective input scheme is depicted in FIGURE 4.19.

4.4.2 Transforming the input scheme

Via applying transformer `IndexPCForET` to the content scheme of the `CMA` it is desired to generate page classes `PaperPage` and `AuthorPage` according to entity types `Paper` and `Author`, respectively. In order to prevent the generation of page classes for entity types `User`, `Group`, `SiteView` and `Conf`, the transformer application needs to be individualized.

<u>*Step 10:*</u> *Individualize the transformer application*: For the purpose of individualizing the application of transformer `IndexPCForET` to the `CMA` content scheme, add the TBE-directive `constraint: or{ENT = 'Author', ENT = 'Paper'};`. This complex constraint achieves that the query template of transformer `IndexPCForET` exclusively selects entity types `Author` and `Paper`. FIGURE 4.20 depicts this TBE-directive.



● Property [or {ENT = 'Author', ENT = 'Paper'}]

Name	Value
Identifier	prop1
Name	or {ENT = 'Author', ENT = 'Paper'}
Value	constraint:

Figure 4.20: Individualizing the application of transformer `IndexPCForET`.

<u>*Step 11:*</u> *Invoking the TBE-engine*: In order to perform the individualized application of transformer `IndexPCForET` to the `CMA` content scheme invoke the TBE-engine from the command line. For details on such invocations confer to the source code documentation of the TBE-engine. The necessary arguments are (1) the name of the transformer to be applied (2) the language used for defining the input scheme and (3) the path to the input scheme on the local file system.

4.4.3 Viewing the output scheme

When the TBE-engine has performed the transformer application, WebRatio recognizes that the `CMA` scheme has changed. Click "yes" when asked by WebRatio whether to update the scheme or not. The resulting page classes `PaperPage` and `AuthorPage` are depicted in FIGURE 4.21.

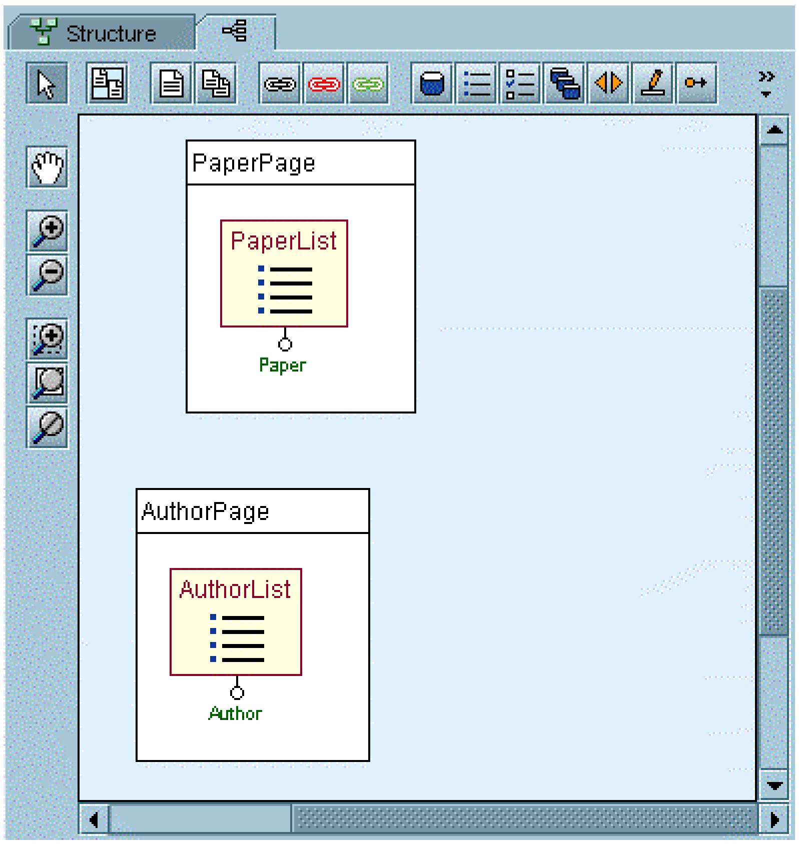


Figure 4.21: Result of the individualized application of transformer `IndexPCForET` *to the* `CMA` *scheme.*

5 Architecture

Content

5.1 Base architecture 80

5.2 Processes 81

5.2.1 Processes for compiling transformer definitions 81
5.2.2 Processes for applying transformers 84

5.3 Datastructures 86

5.3.1 The logical representation of schemes and templates 87
5.3.2 Transformer definitions in terms of TBE 90

This chapter presents the architecture of the TBE-engine, which provides the following functionalities:

1. The TBE-engine provides for compiling a transformer definition in terms of TBE from a transformer definition in notation of a particular modelling language L.
2. The TBE-engine provides for performing the application of transformers defined in notation of L to schemes defined in the same notation.

The goal of this chapter is to identify the model-independent components, i.e. processes and datastructures, of the TBE-engine. Model-independent components are implemented only once for all modelling languages. Consequently, model-dependent components have to be implemented newly for each modelling language. The model-independent components of the TBE-engine make up the TBE-framework, which is the focus of this diploma thesis. SECTION 5.1 describes the base architecture of the TBE-engine. SECTION 5.2 and SECTION 5.3 describe the design of processes and datastructures, respectively, with regard to separate model-independent processes and datastructures from model-dependent ones.

5.1 Base architecture

This section presents the base architecture of the TBE-engine. The base architecture of the TBE-engine is fundamentally appointed by the overall architecture of a TBE-system, which has been introduced in CHAPTER 1. This overall architecture determines that schemes in notation of a particular modelling language L are the interface between the TBE-engine for L and the graphical editor for defining schemes and templates in notation of L.

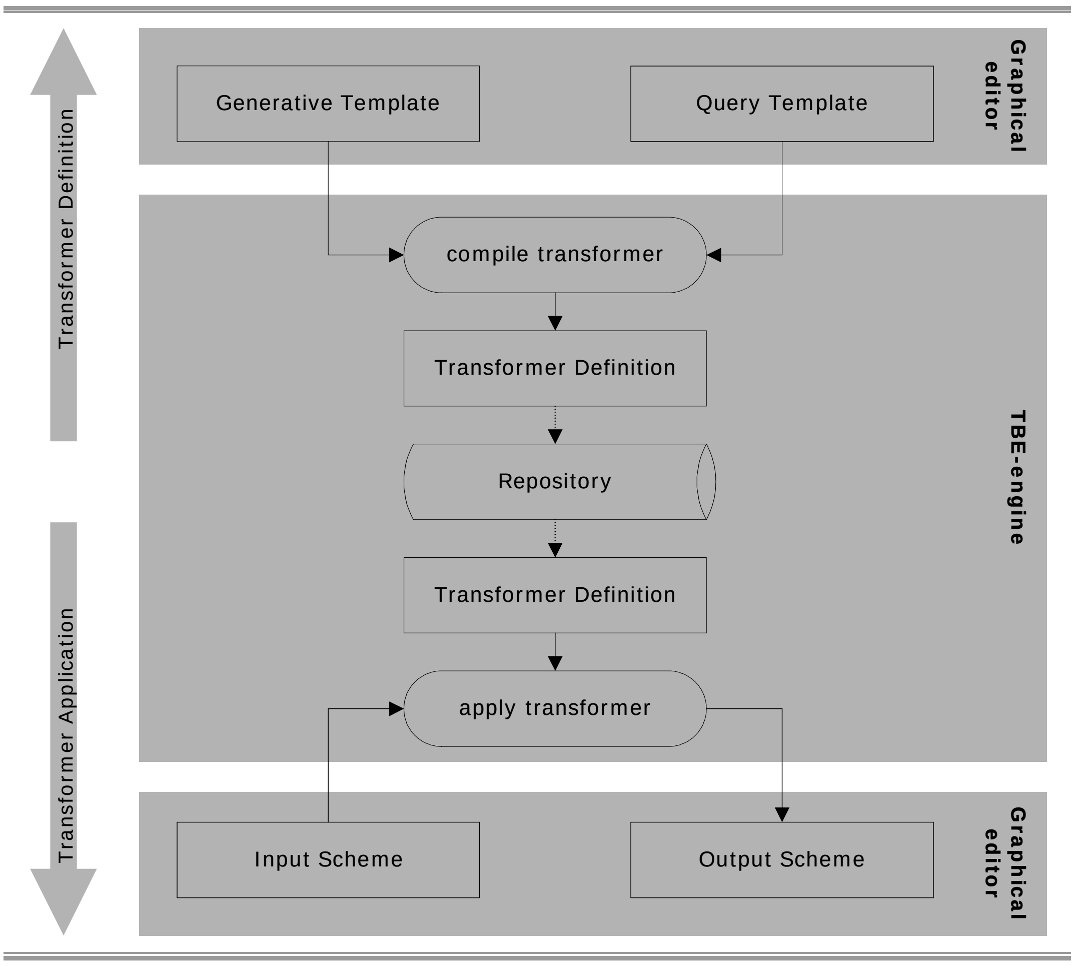


Figure 5.1: Base architecture of the TBE-engine.

When a modeller defines a transformer, she defines a query template and a generative template within the graphical editor of the TBE-system, as depicted in the upper part of FIGURE 5.1. The TBE-engine compiles these templates into a transformer definition in terms of TBE. Finally, the TBE-engine stores the transformer definition into a persistent data storage, called repository.

When a modeller applies a transformer she defines an input scheme within the graphical editor, as depicted in the lower part of FIGURE 5.1. The TBE-engine loads the previously selected transformer definition from the repository and performs the actual transformer application. The result of such a transformer application is the output scheme, which is finally displayed within the graphical editor.

SECTION 5.2 describes the refinement of the base architecture with respect to separate model-independent processes from model-dependent ones. Further, it is shown whether the input or output of a particular process is model-independent or not.

SECTION 5.3 describes the design of datastructures that specify the model-independent inputs and outputs, identified in SECTION 5.2.

5.2 Processes

This section describes the refinement of the processes involved in compiling transformer definitions and the refinement of processes involved in applying transformers, with respect to separate model-dependent processes from model-independent ones. Further, it is described, whether the input or output of a particular process is model-independent or not.

5.2.1 Processes for compiling transformer definitions

From a conceptual point of view, compiling a transformer definition means to derive the formal representation of a transformer definition, i.e. a transformer definition in terms of TBE, from a transformer definition in notation of a particular modelling language L. In the following a transformer definition in notation of L is referred to as *native* transformer definition. For deriving a transformer definition in terms of TBE, the native query template and the native generative template are mapped to their logical one. Then, the logical representations of these templates together with the TBE directives, defined in the native transformer definition are analyzed in order to generate the corresponding transformer definition in terms of TBE.

FIGURE 5.2 depicts an overall view of all processes involved in compiling transformer definitions together with their inputs and outputs. Model-dependent processes and model-dependent inputs and outputs are grey shaded. SECTION 5.2.1.1 describes the processes required for mapping native templates to their logical representation. SECTION 5.2.1.2

describes the processes required for generating a transformer definition in terms of TBE on basis of the templates in logical representation and the corresponding TBE-directives.

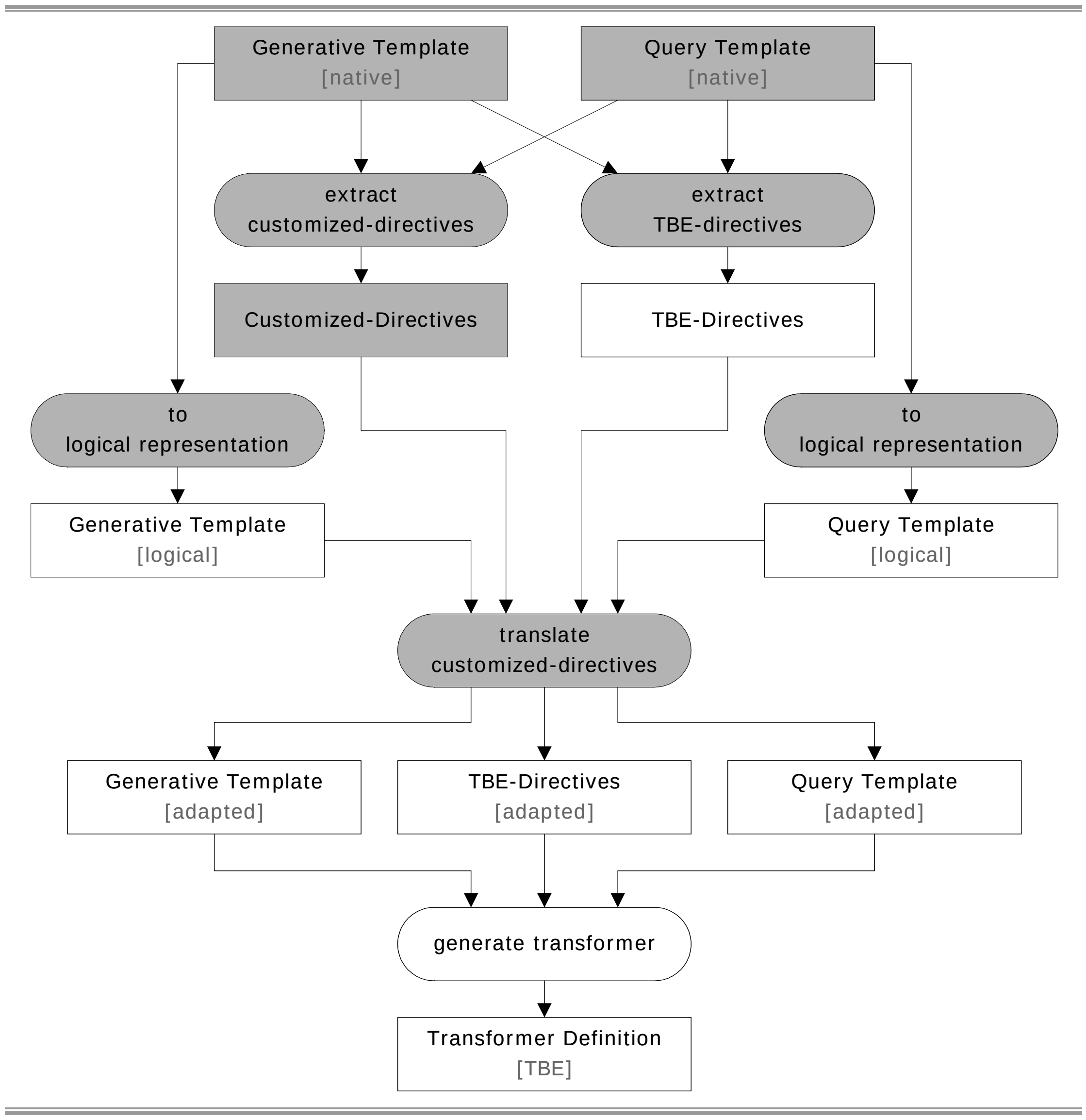

Generative Template [native]
Query Template [native]
extract customized-directives
extract TBE-directives
Customized-Directives
TBE-Directives
to logical representation
to logical representation
Generative Template [logical]
Query Template [logical]
translate customized-directives
Generative Template [adapted]
TBE-Directives [adapted]
Query Template [adapted]
generate transformer
Transformer Definition [TBE]

Figure 5.2: Processes for compiling transformer definitions and their inputs and outputs.

5.2.1.1 Mapping templates to their logical representation

Process `to logical representation` maps the native representation of a template to its logical one. In particular process `to logical representation` maps scheme elements and connections between scheme elements to universe members and relations members, respectively.

The input of process `to logical representation`, i.e. a template in native representation, is clearly model-dependent as each modelling language uses its own datastructures for representing scheme elements. Consequently, process `to logical representation` is model-dependent too. However, the output of this process, i.e. the logical representation of the respective template, is model-independent.

5.2.1.2 Generating transformer definitions in terms of TBE

CHAPTER 4 illustrated that customized-directives are required for defining WebML transformers, i.e. customized-directives `anchor` and `alias`. It is quite fairly to assume that defining transformers in other modelling languages than WebML also requires defining customized-directives. Therefore customized-directives are explicitly addressed within the design of processes for generating transformer definitions in terms of TBE.

The fundamental condition on a customized-directive is, that it can be translated into standard TBE constructs, i.e. variables, constraints and relation constructors. Otherwise, a customized-directive would extend the semantics of TBE, which is not desired. Hence, each customized-directive can be translated into standard TBE constructs by means of some pre-compiler, which is called `translate customized directives` as depicted in FIGURE 5.2.

The remainder of this section introduces the processes for generating a transformer definition in terms of TBE on basis of a query template and a generative template both in logical representation, a set of customized-directives and a set of TBE-directives.

In a first step, processes `extract customized directives` and process `extract TBE directives` filter out the customized-directives and TBE-directives, respectively, defined within either one of the native templates. Both processes are model-dependent since each modelling language uses its own datastructures for representing directives. For example, templates in notation of WebML represent directives by means of tag/value-pairs annotated as user-defined properties.

The output of process `extract TBE directives`, i.e. the set of TBE-directives, is model-independent, since the structure of TBE-directives can be specified, without considering peculiarities of a particular modelling language. The output of process `extract customized directives`, i.e. the set of customized-directives is model-dependent, since specifying templates in different modelling languages may require different customized-directives.

In a second step, process `translate customized directives` prepares the generation of standard TBE constructs that represent the previously extracted customized-directives. For this purpose, process `translate customized directives` adapts the logical representations of the templates and the previously extracted TBE-directives in such a manner, that the following process `generate transformer` generates the desired TBE-constructs.

In a third step process `generate transformer` takes the adapted templates in logical representation and the adapted TBE-directives and generates the transformer definition in terms of TBE accordingly. Since all inputs of this process are model-independent, the process itself is model-independent too. Consequently, the output of process `generate transformer`, i.e. a transformer definition in terms of TBE is also model-independent.

5.2.2 Processes for applying transformers

Applying a transformer means to take a scheme in notation of a particular modelling language `L` as input and perform the transformation of this scheme as specified by the desired transformer. The result of a transformer application is the output scheme, again in notation of `L`. TBE-directives, specifying the individualization of a transformer application, have to be extracted from the input scheme and the respective transformer definition in terms of TBE has to be individualized accordingly.

FIGURE 5.3 depicts the processes involved in applying a transformer. Again, model-dependent processes and model-dependent inputs and outputs are grey shaded.

The native input scheme comprises scheme elements and TBE-directives that specify the transformer application. Process `to logical representation` maps the native input scheme to its logical representation. This process has already been described in SECTION 5.2.1.1. The output of this process is the logical representation of the input scheme and thus model-independent.

Individualizing a transformer application requires to extract the TBE-directives specifying the respective individualization. Again, process `extract TBE directives`, filters out the respective TBE-directives.

These TBE-directives are then passed to process `individualize transformer`. This process loads the specified transformer definition in terms of TBE from the repository and performs the individualization. Since all inputs of process `individualize transformer` are model-independent the process itself is model-independent too.

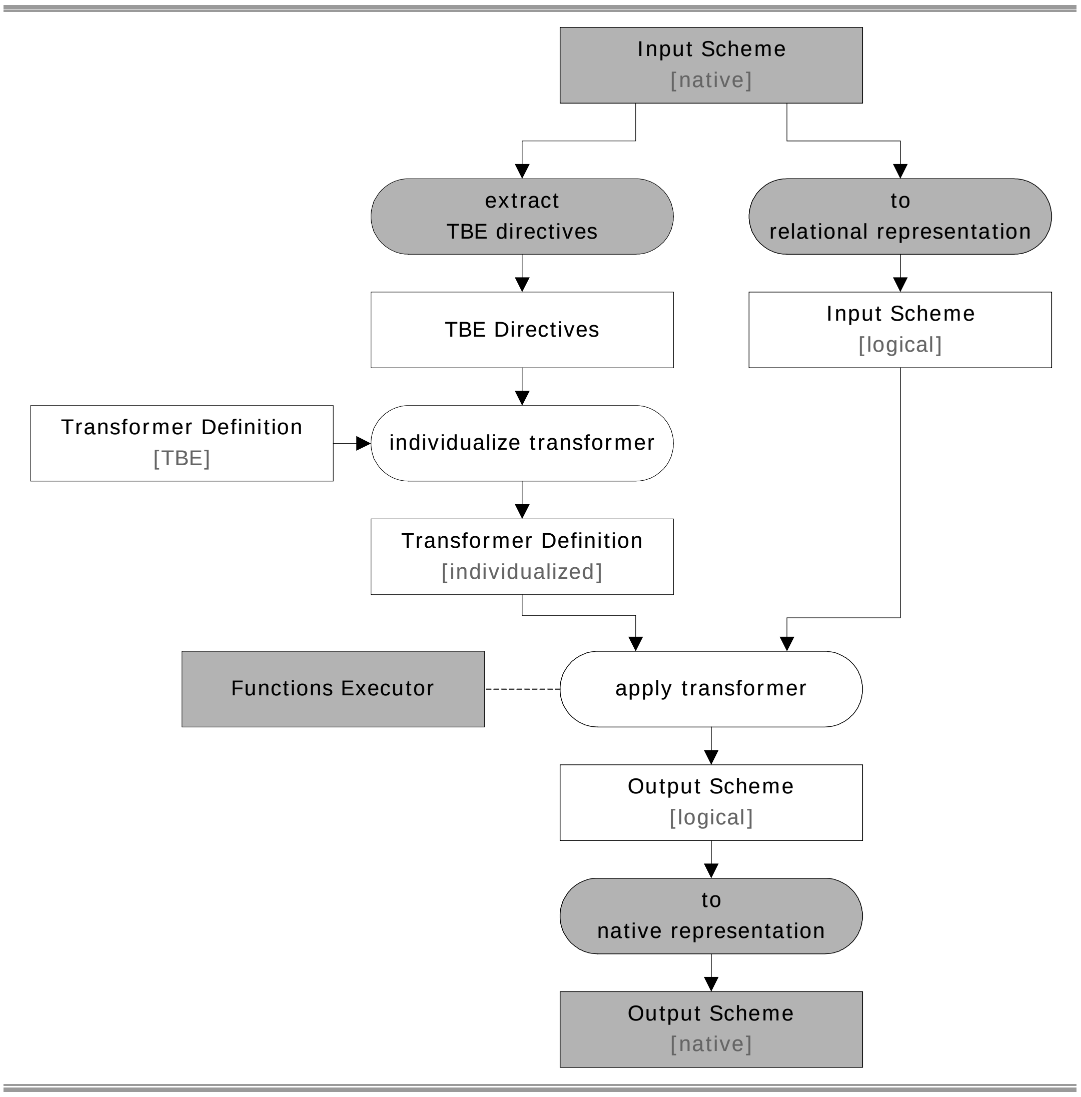


Figure 5.3: Processes and their inputs and outputs required for applying transformers.

The individualized transformer definition and the input scheme in logical representation are then passed to process `apply transformer`. Process `apply transformer` produces the output scheme in logical representation as specified by the transformer definition. Process `apply transformer` is model-independent because its inputs, i.e. a transformer definition in terms of TBE and an input scheme in logical representation, are model-independent too. However, applying a transformer requires the execution of functions as specified within the transformer definition by means of function construction expressions. Since transformers for different modelling languages may generally require the execution of different functions, each implementation of this architecture has to provide a plugin to process `apply transformer` capable of executing such functions. FIGURE 5.3 depicts this plugin, which is called `Functions Executor`.

Finally, model-dependent process `to native representation` maps the output scheme in logical representation to its representation in notation of the respective modelling language.

5.3 Datastructures

The previous section described the processes of the TBE-engine. This section focuses on the structure of model-independent inputs and outputs, since the structure of model-dependent inputs and outputs cannot be generally specified. Model-independent inputs and outputs that are equally structured are specified by a single datastructure, which is summarized in TABLE 5.1.

The structure of a transformer definition in terms of TBE is equal to the structure of an individualized transformer definition. Therefore datastructure `Transformer Definition` specifies both the structure of a transformer definition in terms of TBE and that of an individualized transformer definition.

TBE separates four types of TBE-directives, i.e. directives `tag result variable`, `tag parameter variable`, `constraint` and `construction expression`. TBE-directives `tag result variable` and `tag parameter variable` are trivial. Therefore no datastructures for representing these directives need to be specified. Since constraints and construction expressions are parts of a transformer definition in terms of TBE, datastructure `Transformer Definition` specifies their structures. Therefore, it is not necessary to develop separate datastructures for representing TBE-directives.

The logical representation of both an input scheme and an output scheme solely comprise universe members and relation members and are therefore equally structured. Thus datastructure `Logical Representation` specifies the structure of input schemes and output schemes in logical representation.

Datastructure	*Model independent in-/output*
Transformer Definition	Transformer Definition [TBE] Individualized Transformer Definition TBE Directives
Logical Representation	Generative Template [logical] Query Template [logical] Input Scheme [logical] Output Scheme [logical]

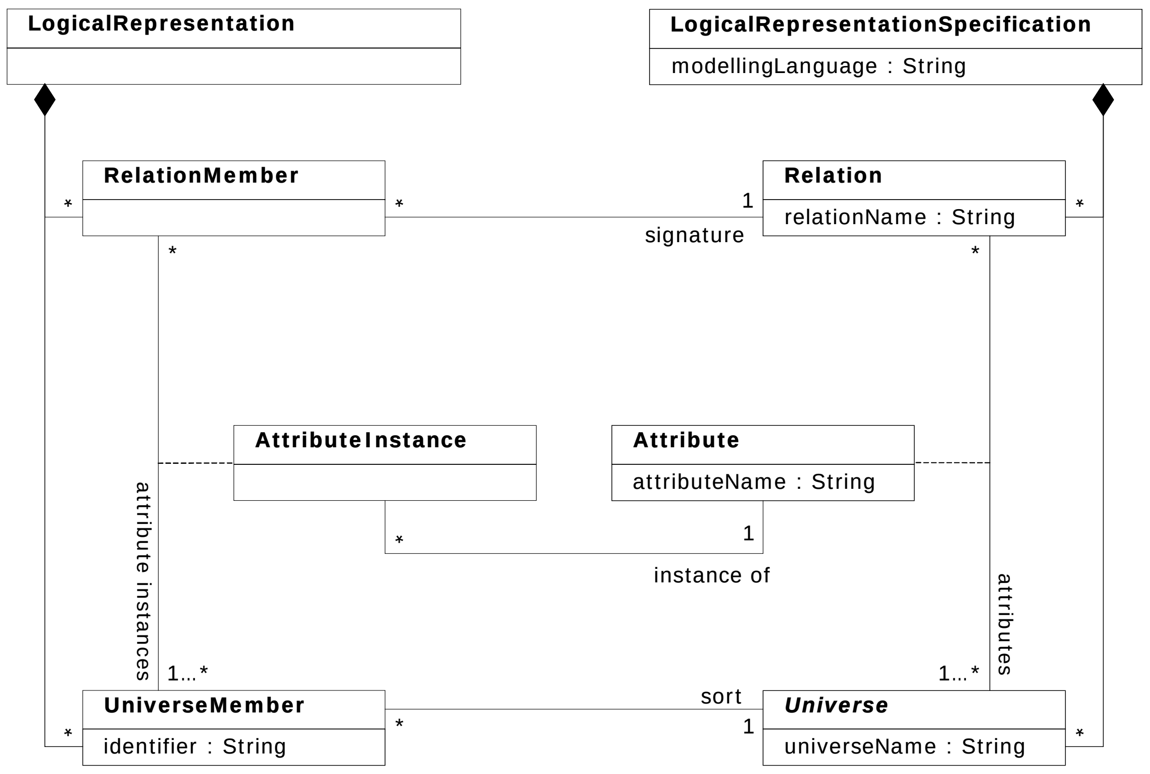
Table 5.1: Datastructures specifying model independent inputs and outputs.

Since templates are schemes extended by directives, their logical representation is equally structured to the logical representation of schemes provided that directives have been extracted previously by process `extract directives`. Thus datastructure `Logical Representation` specifies furthermore the structure of query templates and generative templates in logical representation.

The subsequent sections illustrate the design of model-independent datastructures, where the particular datastructures are specified by means of UML class diagramms and OCL expressions.

5.3.1 The logical representation of schemes and templates

The formal specification of TBE defines that the logical representation of a scheme comprises universe members and relation members. The formal specification of TBE defines further that universe members and relation members are *specified* by universes and relations, respectively. It is notable that the universe members and relation members *representing* a particular scheme are individual for *every* scheme, which comprises individual scheme elements. Whereas, the universes and relations specifying the sorts of universe members and relation members intended for representing schemes defined in a particular modelling language, are the same of for all schemes defined in the respective modelling language.



```
OCL1: context Universe inv:
      self.allInstances ->
      forAll (u1, u2 | u1 <> u2 and u1.LRS = u2.LRS)
      implies u1.universeName <> u2.universeName
OCL2: context Attribute inv:
      self.allInstances ->
      forAll (a1, a2 | a1 <> a2 and a1.Relation = a2.Relation)
      implies a1.attributeName <> a2.attributeName
OCL3: context UniverseMember inv:
      self.allInstances ->
      forAll (um1, um2 | um1 <> um2 and
              um1.LR = um2.LR)
      implies um1.identifier <> um2.identifier
OCL4: context AttributeInstance inv:
      self.Attribute a1 ->
      exists(a2: Attribute |
             a2.Relation = a1.RelationMember.Relation )
```

Figure 5.4: Datastructures `LogicalRepresentation` *and* `LogicalRepresentationSpecification`.

Therefore, it is reasonable to specify the sorts of universes and relations intended for representing schemes of a particular modelling language `L` *once* instead of specifying them newly for each logical representation of a scheme in notation of `L`. Therefore datastructure `Logical Representation Specification` is introduced that specifies the structure of universes and relations. The specification of the logical representation of schemes in notation of `L` is stored in the repository of the TBE-engine for being available for processes, any time it is required.

As depicted in FIGURE 5.4, class `Logical Representation Specification` has property `modellingLanguage` attached in order to be uniquely addressable.

Each universe has a name, as expressed by property `universeName`. The name of a universe must be unique within all universes intended for representing schemes in notation of `L` as expression `OCL1` specifies.

Analogous to universes, relations are named as denoted by property `relationName`. The name of a relation does not have to be unique, since relations may be overloaded. This means that several relations may share the same name but have different attributes. Association `attributes`, depicted in FIGURE 5.4, represents the attributes of a relation.

Generally, there are two alternatives for identifying attributes within relations. First their position within the relation can be used for identification. This approach meets the formal specification of TBE best. However, identifying attributes within relations by their position lacks of readability. Therefore attributes of a relation are identified by unique names, which is denoted by association class `Attribute`. Consequently, the name of an attribute must be unique among all attribute names of one relation, which is specified by expression `OCL2`.

Each universe member has an unique identifier as denoted by property `identifier`. The identifier of a universe member must be unique among all universe members of its domain. Expression `OCL3` specifies this constraint on identifiers of universe members.

Each relation member declares the relation that serves as its signature, as expressed by association `signature`. The universe members connected by a relation member are subsequently referred to as *attribute instances*. For identifying attribute instances within a relation member, each attribute instance declares the attribute it is an instance of, which is denoted by association `instance of`. It has to be ensured that each attribute instance of a particular relation member is an instance of an attribute of the relation, which is the signature for the respective relation member. Expression `OCL4` specifies this constraint.

5.3.2 Transformer definitions in terms of TBE

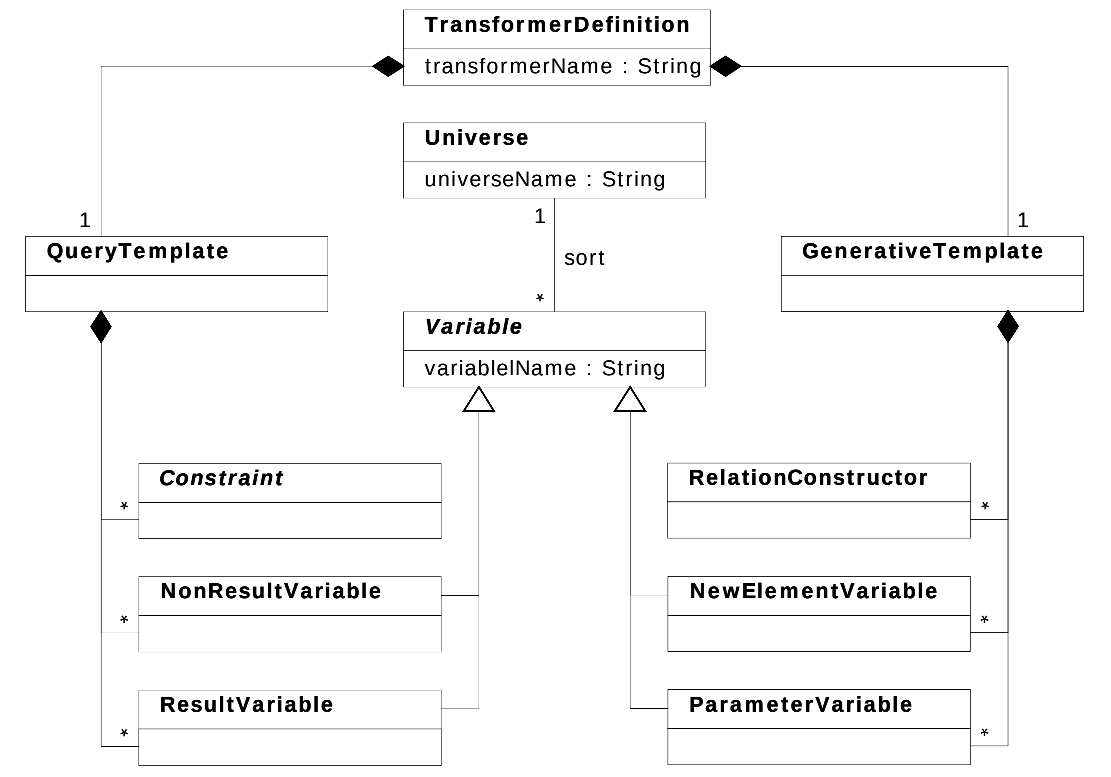
A transformer definition in terms of TBE comprises exactly one query template and one generative template, as depicted in FIGURE 5.5. Each transformer definition is uniquely identified by its name, for being addressable at application time, and has therefore property `transformerName` attached. Expression `OCL5` specifies that the name of a transformer definition must be unique within all transformer definitions stored in the repository.

Query templates and generative templates comprise different types of variables. Therefore, abstract class `Variable` is introduced, which captures the common structure of all types of variables. The common structure of all types of variables is to be addressable by a name as expressed by property `variableName`. Expression `OCL6` specifies that the name of a variable must be unique within all variables of the template the respective variable is defined at. Thus it is generally possible that a variable defined in a query template has the same name as a variable defined in the generative template. This circumstance is even required, since parameter variables at a transformer's generative template must match result variables at its query template. Further, each variable has a universe attached, which serves as domain for this variable. This is captured by association `sort`.

A query template comprises two types of variables, i.e. result variables and non-result variables, which is expressed by sub-classes `ResultVariable` and `NonResultVariable`, respectively. Besides these types of variables a query template comprises several sorts of constraints. Therefore abstract class `Constraint`, which is in detail described in SECTION 5.3.2.1, is introduced.

A generative template also comprises two types of variables, i.e. parameter variables and new-element variables, which are represented by equally named sub-classes of class `Variable`. SECTION 5.3.2.3 describes class `New Element Variable` in detail. Class `Parameter Variable` is not further described, since it has only the features of super class `Variable`. Besides variables, a generative template comprises relation constructors, as class `Relation Constructor` expresses. SECTION 5.3.2.2. describes this class in detail.

A transformer definition is a combination of a query template and a generative template. Such combinations are proper if for each parameter variable at the generative template an equally named and equally sorted result variable is provided at the query template and vice versa. Expression `OCL7` specifies this condition.



```
OCL5: context TransformerDefinition inv:
      self.allInstances ->
      forAll (t1, t2 | t1 <> t2
      implies t1.transformerName <> t2.transformerName
OCL6: context Variable inv:
      self.allInstances ->
      forAll (v1, v2 | v1 <> v2 and
             ((v1.GT = v2.GT) or (v1.QT = v2.QT))
      implies v1.variableName <> v2.varableName
OCL7: context Parameter inv:
      self.variableName vn, self.UniverseName un ->
      exists (vr: ResultVariable |
              vr.QT.TD = self.GT.TD and
              vr.variableName = nv and
              vr.universeName = un)
```

Figure 5.5: Overview of datastructure `TransformerDefinition` *(top)*
and required OCL expressions (bottom).

5.3.2.1 Constraints

The formal specification of TBE distinguishes complex constraints, membership constraints and comparison constraints. Therefore accordingly named classes specifying the structure of these constraints are introduced, as depicted in FIGURE 5.6.

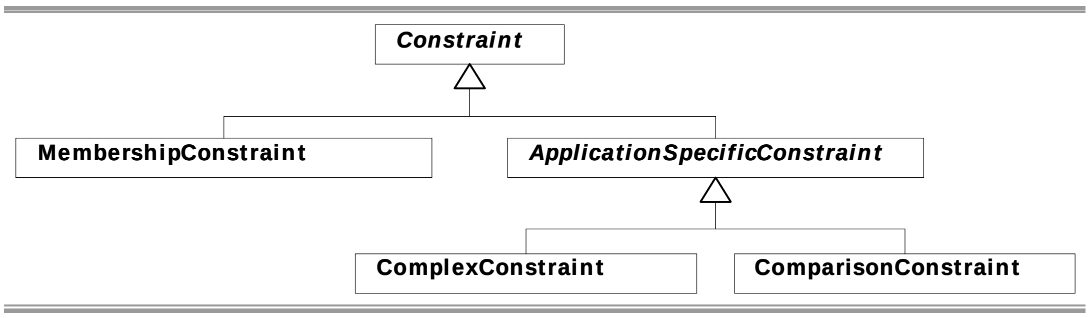
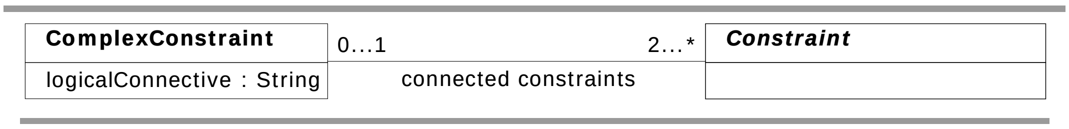


Figure 5.6: Type hierarchy of constraints.

A subset of all types of constraints can be used at application time for individualizing the transformer application. Thus, abstract class `Application Specific Constraint` is introduced. Classes `Complex Constraint` and `Comparison Constraint` are sub-classes of class `Application Specific Constraint` and can therefore be used for individualizing transformer applications. In the following the distinctive types of constraints are described.

5.3.2.1.1 *Complex Constraints*



```
OCL8: context ComplexConstraint inv:
      self.logicalConnective = 'and' | 'or'
```

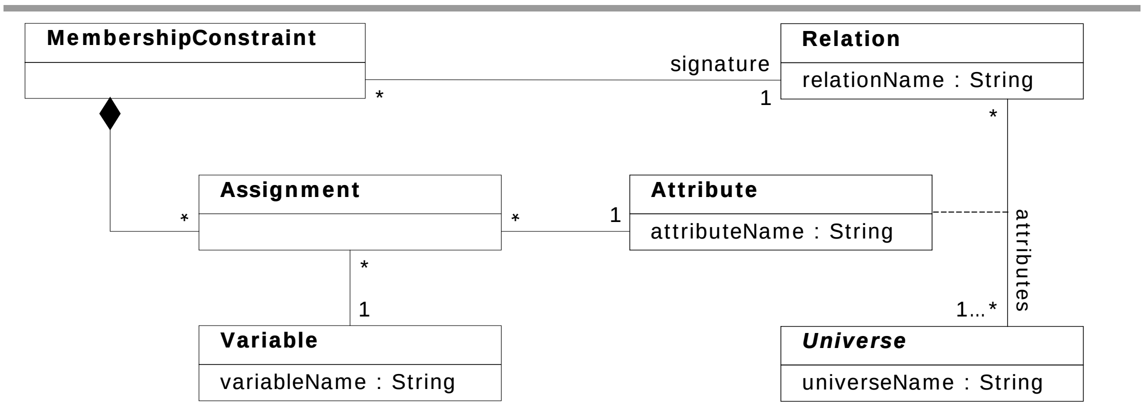
Figure 5.7: Structure of complex constraints.

Complex constraints connect at least two constraints with a logical connective as depicted in FIGURE 5.7. Class `ComplexConstraint` has property `logicalConnective` attached determining the constraint's logical connective. Valid logical connectives are `and` and `or` as specified by expression `OCL8`.

5.3.2.1.2 *Membership Constraints*

A membership constraint specifies relation members that have to exist within the logical representation of an input scheme in order to fulfill the constraint. FIGURE 5.8 depicts the structure of membership constraints.

Each membership constraint declares the signature of relation members that have to be examined on evaluation time, as denoted by association `signature`. Evaluating membership constraints means to compare instances of attributes to values of variables, as conceptually shown in SECTION 3.2.1.3. Therefore, assignments of variables to attributes are required, which is expressed by class `Assignment`. Each assignment relates one attribute to one variable. As specified in the formal specification of TBE, only result variables or non-result variables can be assigned to attributes within a membership constraint, which is specified by expression `OCL9`. Further, a membership constraint that has a relation `R` as signature may assign variables only to attributes of `R`, which is specified by expression `OCL10`.

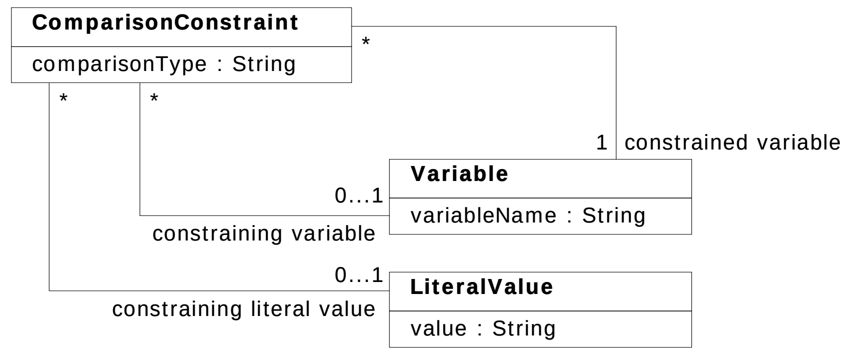


```
OCL9:  context MembershipConstraint inv:
       forAll (a: Assignment | a.MembershipConstraint = self)
       implies (oclIsTypeOf(a.variable: ResultVariable) or
                oclIsTypeOf(a.variable: NonResultVariable))
OCL10: context Assignment inv:
       self.MembershipConstraint.Relation = self.Attribute.Relation
```

Figure 5.8: Structure of membership constraints.

5.3.2.1.3 Comparison Constraints

A comparison constraint specifies that a variable, called constrained variable, has to hold a particular value at evaluation time in order of fulfilling the constraint, as depicted in FIGURE 5.9. For evaluating comparison constraints the value of the constrained variable is compared to the values of other variables or literal values according to the type of comparison. Therefore, datastrucutre `ComparisonConstraint` has property `comparisonType` attached specifying the type of comparison. Expression `OCL11` specifies that the comparison type has to be one of `equal` or `notEqual` as defined in the formal specification of TBE.



```
OCL11: context ComparisonConstraint inv:
       self.comparisonType = 'equal' | 'not equal'
OCL12: context ComparisonConstraint inv:
       oclIsTypeOf(self.constrainingVariable: ResultVariable) or
       oclIsTypeOf(self.constrainingVariable: NonResultVariable)
OCL13: context ComparisonConstraint inv:
       self.constrainingVariable -> notEmpty() implies
       self.constrainingLiteralValue -> isEmpty()
       self.constrainingLiteralValue -> notEmpty() implies
       self.constrainingVariable -> isEmpty()
```

Figure 5.9: Structure of comparison constraints.

The formal specification of TBE defines that a comparison constraint compares the value of the constrained variable to either the value of *one* variable or *one* literal value, which is denoted by associations `constraining variable` and `constraining literal value`, respectively. Expression `OCL12` specifies that a variable may only be constrained by either another variable or an arbitrary literal value. Further, expression `OCL13` specifies that only result variables or non-result variables can be used as constraining variables.

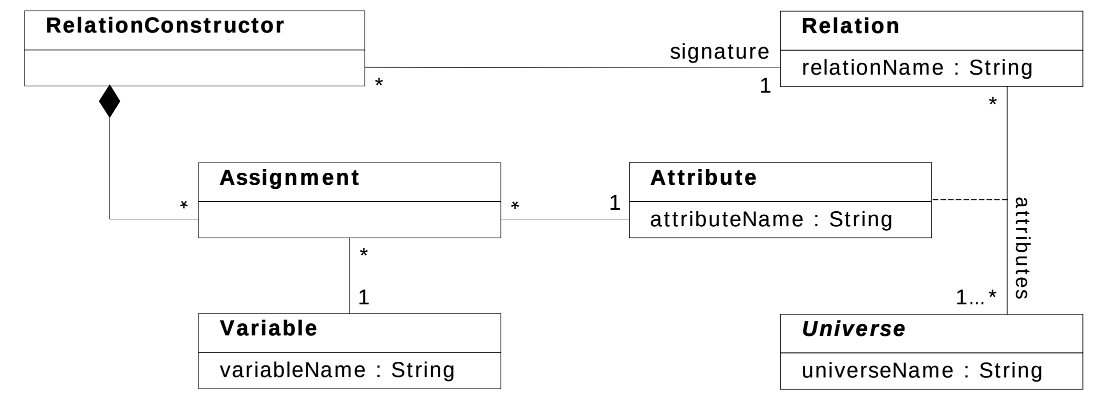
5.3.2.2 Relation constructors

A relation constructor specifies the generation of a relation member. The signature of the relation member to be generated is given by a particular relation and therefore declared by the relation constructor. This is expressed by association `signature`, as depicted in FIGURE 5.10.

Generating a relation member requires to generate attribute instances. The values of variables are used for generating attribute instances as conceptually shown in SECTION

3.2.2.3. Therefore, assignments of attributes to variables are required for determining which attribute instance is to be generated according to the value of which variable.

Since assignments of attributes to variables required in the scope of relation constructors are similar to the assignments required in the scope of membership constraints, class `Assignment` is reused.



```
OCL14: context Assignment inv:
       self.RelationConstructor.Relation = self.Attribute.Relation
OCL15: context RelationConstructor inv:
       forAll  (a: Assignment | a.RelationConstructor = self)
       implies (oclIsTypeOf(a.variable: NewElementVariable) or
                oclIsTypeOf(a.variable: ParameterVariable))
```

Figure 5.10: Structure of relation constructors.

The requirement that relation constructors may exclusively assign variables to attributes of the relation, which is the signature of the relation constructor, is specified by expression `OCL14`. This requirement is analogous to assignments in the scope of membership constraints. Yet, relation constructors additionally require that only new-element variables or parameter variables can be assigned to attributes. This additional requirement is by `OCL15`.

5.3.2.3 New-element variables

New-element variables specify the generation of universe members and have for that purpose a construction expression attached. Therefore, class `New Element Variable` declares to comprise exactly one `Construction Expression`. In the formal specification of TBE three types of construction expressions are distinguished as described by classes

`New Construction Expression`, `Constant Construction Expression` and `Function Construction Expression`. Each of these classes is a sub-class of class `Construction Expression`. In contrast to application-specific constraints no class for application-specific construction expressions is designed, as *every* construction expression can be used at application time for individualizing the transformer application.

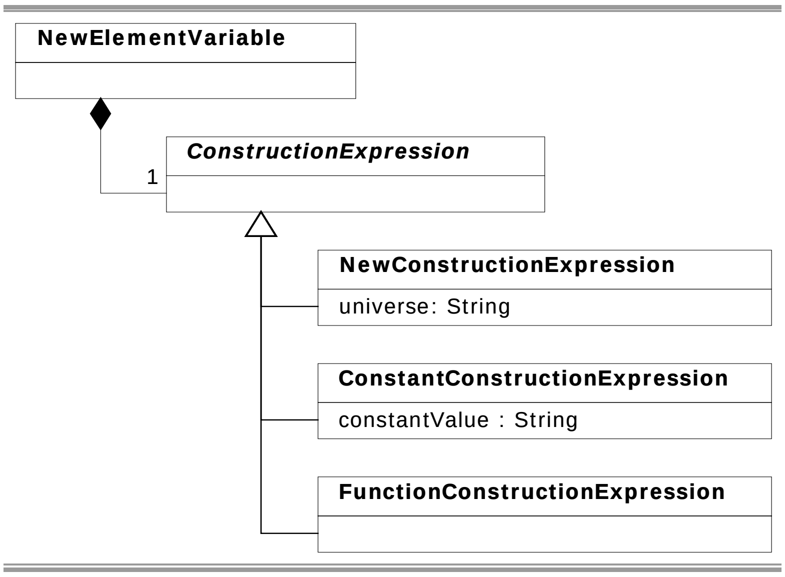

NewElementVariable
1
ConstructionExpression
NewConstructionExpression
universe: String
ConstantConstructionExpression
constantValue : String
FunctionConstructionExpression

Figure 5.11: Structure of new-element variables.

Class `New Construction Expression` has property universe attached. This property determines the sort of the universe member that is to be generated.

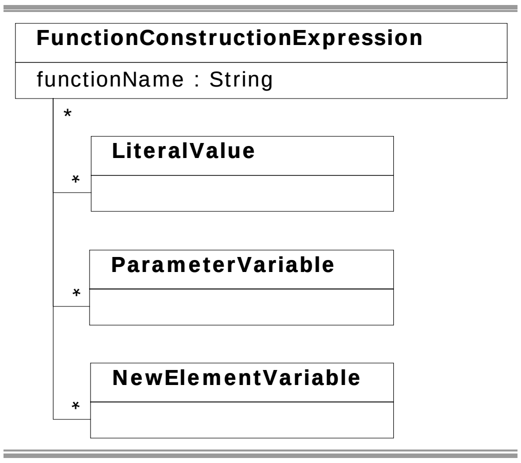


Figure 5.12 Structure of function construction expressions.

Constant construction expressions determine the constant value representing the identifier of the universe member to be generated. Therefore, class `Constant Construction Expression` has property `constantValue` attached.

Function construction expressions determine (i) which function is to be executed and (ii) which parameter variables, new-element variables or literal values, are to be passed to the function as arguments. For reasons of clarity, class `Function Construction Expression` is depicted separately in FIGURE 5.12.

6 Implementation

Contents

6.1 Choice of technologies 100

6.1.1 Technologies for compiling transformer definitions 101
6.1.2 Technologies for applying transformers 105

6.2 Two-layered implementation of the TBE-engine 107

6.2.1 Allocation of process implementations 108
6.2.2 Allocation of datastructure-implementations 113

6.3 Implementation of Datastructures 115

6.3.1 Datastructure *Logical Representation Specification* 115
6.3.2 Datastructure *Logical Representation* 122
6.3.3 Datastructure *Transformer Definition* 124
6.3.4 Datastructure *TBE-Directives* 124

6.4 Implementation of Processes 125

6.4.1 Compressed logical representations of schemes 126
6.4.2 Processes for compiling transformer definitions 127
6.4.3 Processes for applying transformers 137

This chapter presents the implementation of the TBE-engine. SECTION 6.1 discusses the choice of technologies for implementing the components of the TBE-engine.

The implementation of the TBE-engine is realized on two layers. The *model-independent layer* provides implementations of the components that make up the TBE-framework, i.e. model-independent components. Consequently, the *model-dependent layer* provides implementations of model-dependent components. SECTION 6.2 describes this two-layered implementation of the TBE-engine.

SECTION 6.3 and SECTION 6.4 describe the concrete implementations of datastructures and processes, respectively.

6.1 Choice of technologies

This section focuses on the choice of technologies for implementing the components of the TBE-engine. For this purpose the approaches to the implementation of the processes of the TBE-engine are described at a high level of abstraction in order to justify the respective technology choices. Basically, the TBE-engine is implemented in JAVA for the following reasons:

- *Library support*: The selection of technologies for implementing model-dependent components clearly depends on given factors of the respective modelling language. Consider, for example, the implementation of process `To Logical Representation` for modelling language WebML, which can be conveniently developed using XSLT, since WebML schemes are provided as XML documents. There are numerous XSLT processors available in JAVA, e.g. Xalan [xal04] and Saxon [sax04]. Thus, in order to enable the flexible use of technologies, for the implementation of model-dependent components, it is reasonable to implement the TBE-engine in JAVA, since libraries for the support of numerous technologies are available.
- *Operating system independency*: In order to support the application and definition of transformers, the TBE-engine requires a graphical editor for defining schemes and templates. Thus the TBE-engine should be executable within the operating system the graphical editor is executed within as well. Since JAVA byte code is executable within every established operating system, it is convenient to implement the TBE-engine in JAVA.

SECTION 6.1.1 discusses the choice of technologies for realizing the components of the TBE-engine used for compiling transformer definitions. SECTION 6.1.2 discusses the choice of technologies for realizing components used for applying transformers. For choosing technologies the following basic considerations are taken into account.

- Technologies for realizing model-dependent components are selected with regard to modelling language WebML. Implementations of model-dependent components for other modelling languages may use different technologies.

- Technologies for realizing model-independent components are selected with regard to the adequacy of a technology for the respective process. Thus, peculiarities of modelling languages, for example the format of the internal representation of schemes, are not considered by the choice of technologies for realizing model-independent components.

6.1.1 Technologies for compiling transformer definitions

This section discusses the choice of technologies for realizing those components of the TBE-engine that are used for compiling transformer definitions. FIGURE 6.1 summarizes the chosen technologies, where the particular technologies are depicted in the left hand side. In the following the respective choices are discussed.

Process *To Logical Representation*: The input of model-dependent process `To Logical Representation` is a template in the native notation of a particular modelling language. In the case of WebML, the input of this process is a scheme in terms of XML. XSLT is used for implementing process `To Logical Representation`, since it is convenient to develop XSLT template rules for mapping scheme elements and connections between scheme elements to universe members and relation members, respectively. Consequently, the output of model-dependent process `To Logical Representation` is some XML data.

Datastructure *Logical Representation*: An XML-schema (XSD) is used for specifying the structure of logical representations of templates (schemes) in terms of XML.

Process *Extract TBE Directives*: The input of model-dependent process `Extract TBE Directives` is again an XML document representing a template. An XSLT stylesheet extracts the textual representations of TBE-directives, i.e. directives `constraint`, `construction expression`, `tag result variable` or `tag parameter variable`. TBE-directives `result variable` and `parameter variable` are not directly defined within templates in notation of WebML. Instead they are defined by means of defining shortcuts, i.e. dollars signs preceding the name of a variable. Therefore, process extract TBE directives filters out these shortcuts and translates them into the corresponding textual representation.

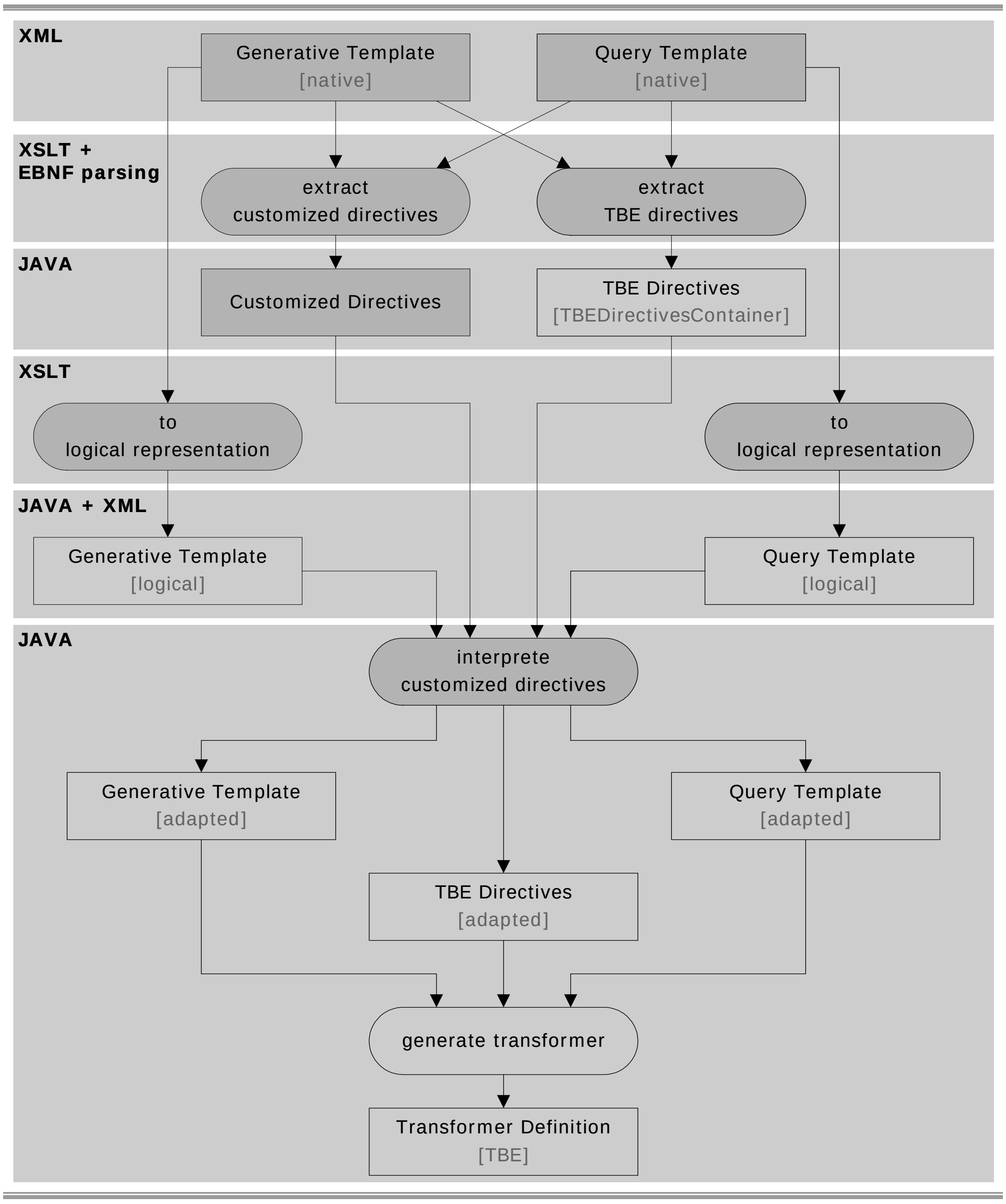


Figure 6.1: Technologies for implementing components for compiling transformer definitions.

Datastructure *TBE-Directives*: The output of process `extract TBE directives` comprises JAVA representations of the extracted TBE-directives. Therefore, subsequent processes work directly with JAVA representations of TBE-directives instead of parsing their textual representation newly each time. A specific JAVA class called `TBEDirectivesContainer` stores the JAVA representations of the extracted TBE-directives. The JAVA representations of TBE-directives `tag result variable` and `tag` `parameter variable` are collections of variable names, i.e. one collection that stores names of result variables and one collection that stores names of parameter variables. The JAVA representations of TBE-directives `construction expressions` and `constraint` are specifically designed JAVA classes. These JAVA classes are derived from the UML-class diagrams specifying the structure of construction expressions and constraints, which are depicted in SECTION 5.4.2. Consequently, the `TBEDirectivesContainer` provides for storing collections of JAVA classes representing construction expressions and constraints. Thus, the output of process e`xtract TBE directives` is a `TBEDirectivesContainer`.

An EBNF grammar parser generates the JAVA representations of TBE-directives on basis of their textual representation. This EBNF grammar parser is generated by AntLR [antlr04], which is a parser generator. This EBNF grammar parser has to be implemented only once, since the syntax of textual representations of TBE-directives is specified, as shown in SECTION 5.4.3.

Process *Extract Customized Directives*: An XSLT stylesheet extracts customized-directives, i.e. directives `anchor` and `alias`. Again, the extracted customized-directives are represented by JAVA objects in order to be conveniently processed within subsequent processes. In contrast to TBE-directives no specifically designed JAVA class is necessary for representing customized-directives for the following reasons. Aliases for non-editable properties of scheme elements are immediately resolved by process t`o logical representation`. Therefore customized-directive `alias` does not have to be represented by a specifically designed JAVA object at all. Further, since the structure of customized-directive `anchor` is fairly simple, a JAVA object of class `String` is proper for representing this customized-directive.

Process *Interpret Customized Directives*: Process i`nterpret customized directives` is implemented in JAVA, since expressing the semantics of customized-directives via (i) adapting the logical representations of templates and (ii) adapting TBE-directives, suggests the use of a procedural programming language like JAVA.

The inputs of process `interpret customized directives` are a set of customized-directives, a set of TBE-directives and the templates in logical representation. Thereby, customized-directives and TBE-directives are passed to this process in terms of JAVA objects, as previously argued, and can therefore be conveniently processed. However, the query template and the generative template in logical representation are passed in terms of

XML by process `to logical representation`. For processing XML in JAVA the following three alternatives are available:

- *DOM parser*: One alternative for processing XML within JAVA is to use a document object parser (DOM parser). The XML document is represented as a tree, which can be traversed, in order to process the XML document. Thereby, each node of a DOM tree represents an element (element node) or an attribute (attribute node) of the represented XML document. The main advantage of this alternative is that each XML document can be processed, regardless of the DTD the XML document adheres. Therefore, DOM parsing is a generic approach to processing XML documents. The drawback of this approach is, that it is inconvenient to use such generic representations of XML data for implementing processes.
- *SAX parser*: Another alternative for processing XML within JAVA is to use a serial access parser (SAX parser). The XML document is traversed by the SAX parser in a serial way and each time an XML element or XML attribute is recognized a particular event is thrown by the SAX parser, which can be further processed from within JAVA. The advantages and the drawbacks of SAX parsing are analogous to those of DOM parsing.
- *Specific JAVA objects*: A convenient alternative for processing XML within JAVA is to use JAVA objects, which enable the manipulation of the respective XML data by means of getter-methods and setter-methods. Clearly, this approach requires that specific JAVA classes are designed for representing the structure of the respective XML document. Further, it is required to implement a parser that generates the specific JAVA objects on basis of the respective XML document. However, the manipulation of an XML document can be conveniently implemented using specific JAVA objects because they provide specific getter-methods and setter-methods.

Process `interpret customized directives` uses specific JAVA objects for adapting the logical representations of templates for the following reasons. First, it is the most convenient alternative for processing XML within JAVA. Second, the additional efforts for (i) implementing specifically designed JAVA classes for representing the logical representation of templates and (ii) implementing a corresponding parser are little since the structure of the logical representation of templates is fairly simple. Thus, datastructure Logical Representation is additionally implemented by means of specifically designed JAVA classes.

Process *Generate Transformer*: Process `Generate Transformer` is implemented in JAVA since generating a transformer definition in terms of TBE on basis of two templates in logical representation and a set of TBE-directives suggests the use of a procedural programming language like JAVA. Further, storing the generated transformer definition into the repository of the TBE-engine can be easily achieved by serializing the JAVA objects representing the transformer definition.

Datastructure *Transformer Definition*: Consequently, specifically designed JAVA classes for representing the output of process Generate Transformer, i.e. a transformer definition in terms of TBE, have been developed. Again, the design of these classes is derived from the UML classes specifying datastructure transformer definition, which have been described in SECTION 5.3.2.

6.1.2 Technologies for applying transformers

This section discusses the choice of technologies for implementing these components of the TBE-engine that are used for applying transformers. FIGURE 6.2 summarizes the chosen technologies. Again, the particular technologies are depicted in the left hand side. The respective choices are discussed in the following.

The input scheme is provided in terms of XML as determined by modelling language WebML. The choices of technologies for model-dependent processes `extract TBE` d`irectives` and `to` l`ogical` r`epresentation` have already been discussed in the previous section. For repetition, the TBE-directives specifying the individualization of the transformer application, i.e. constraints and construction expressions, are extracted from the input scheme using an XSLT stylesheet and returned as a JAVA object of class `TBEDirectivesContainer`. An XSLT stylesheet maps the input scheme to its logical representation.

Process *Individualize Transformer*: The inputs of process `individualize transformer` are (i) a `TBEDirectivesContainer` comprising application-specific constraints and application-specific construction expressions and (ii) a transformer definition in terms of TBE. This transformer definition is loaded from the repository of the TBE-engine and represented as a set of specifically designed JAVA objects. The task of individualizing a transformer definition is comparable to the task of generating a transformer since both tasks manipulate a transformer definition. Therefore process `individualize` `transformer` is implemented in JAVA. The output of this process, i.e. the individualized transformer definition, is passed to process apply transformer by means of specifically designed JAVA objects.

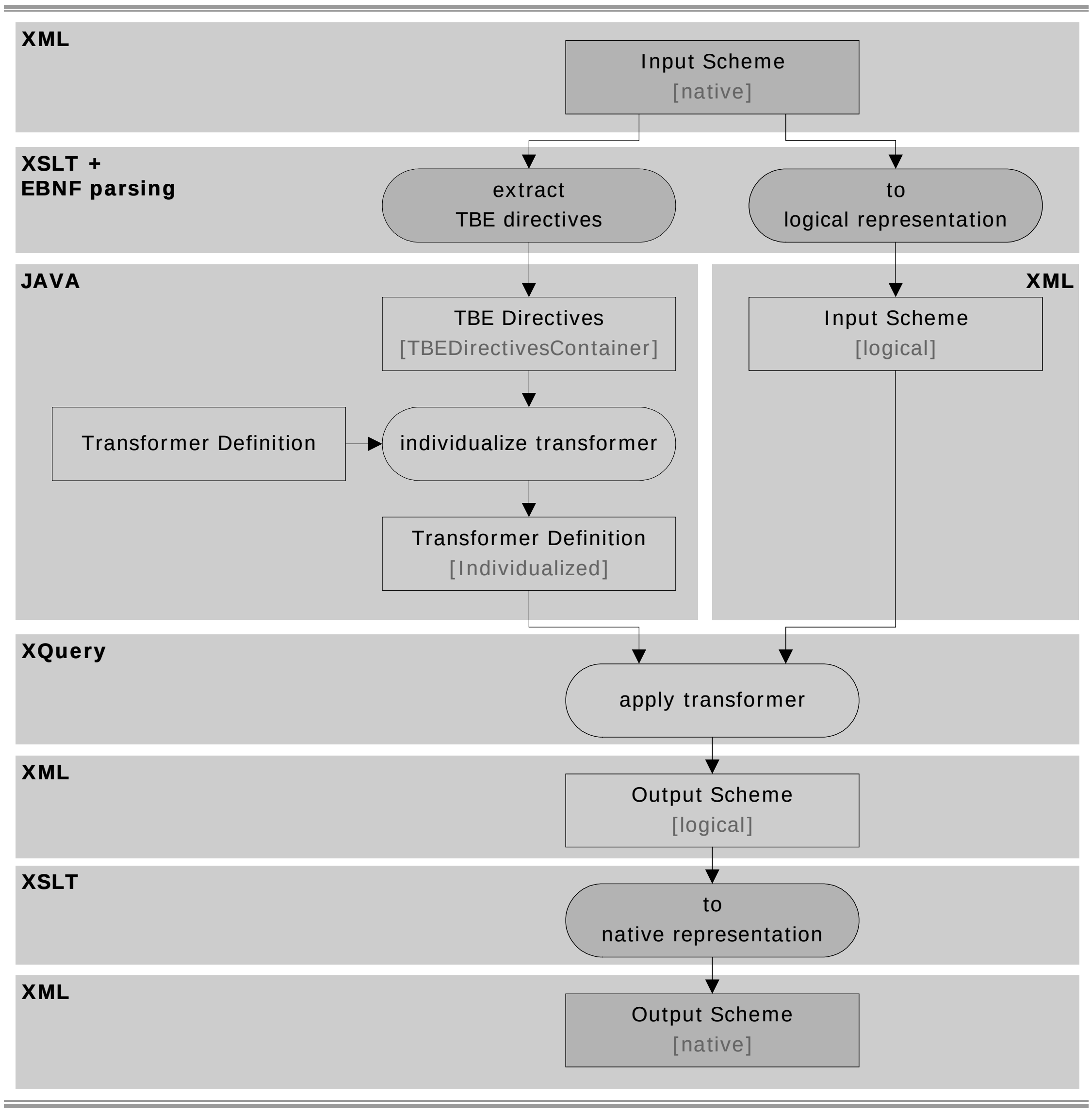


Figure 6.2: Technologies for implementing components for applying transformers.

Process *Apply Transformer*: For performing the application of a transformer two alternatives are available. First, one can develop an ad-hoc interpreter that performs the scheme transformation. The second alternative is to generate a script in terms of another language like, for example, XQuery that is then executed by an adequate processor. For quickly getting a prototype, we decided to follow the second approach.

For choosing a language the following requirements need to be met. First the language must be adequate for expressing the semantics of a transformer. This means that the language must enable the evaluation of query templates and the instantiation of generative templates.

Concerning query template evaluation, following two options have been identified. First a deductive language like, for example, F-logic, Prolog, or Datalog can be used. This option is straightforward since query templates are based on domain relational calculus, which itself is based on first-order logic. Second, a query template can be translated into an XQuery statement.

Concerning generative template instantiation, any of these languages can be chosen as soon as it supports data manipulation, i.e. creating and adding relation members and universe members. Datalog and F-logic, which are basically data retrieval languages, are not adequate since they do not support expressions for creating new universe members. Therefore, only Prolog and XQuery are candidate technologies for implementing process `apply transformer`. This diploma thesis demonstrates the implementation of process `apply transformer` with XQuery. Thus, the output of process `apply transformer` is the output scheme's logical representation in terms of XML, since the execution of an XQuery statement produces XML data.

Process *To Native Representation*: Model-dependent process `to native representation` is implemented in XSLT, since process `apply transformer` passes the output scheme's logical representation in terms of XML and the output scheme in terms of WebML, i.e. the output of process `to native representation` is also expected to be represented in XML.

6.2 Two-layered implementation of the TBE-engine

The architecture of the TBE-engine, described in CHAPTER 5, consists of model-dependent and model-independent components. For repetition, model-dependent components have to be newly implemented for each modelling language, the TBE-engine supports scheme transformations for, whereas model-independent components are implemented once for all modelling languages. In order to consider this separation of components in the implementation of the TBE-engine, it is implemented on two layers. The *model-*

independent layer provides implementations of model-independent components and the *model-dependent layer* provides implementations of model-dependent components.

This section focuses on the allocation of implementations of components between the two implementation layers of the TBE-engine. SECTION 6.2.1 describes the allocation of process implementations. SECTION 6.2.2 describes the allocation of implementations of datastructures.

6.2.1 Allocation of process implementations

FIGURE 6.3 illustrates the allocation of process implementations to the implementation layers of the TBE-engine.

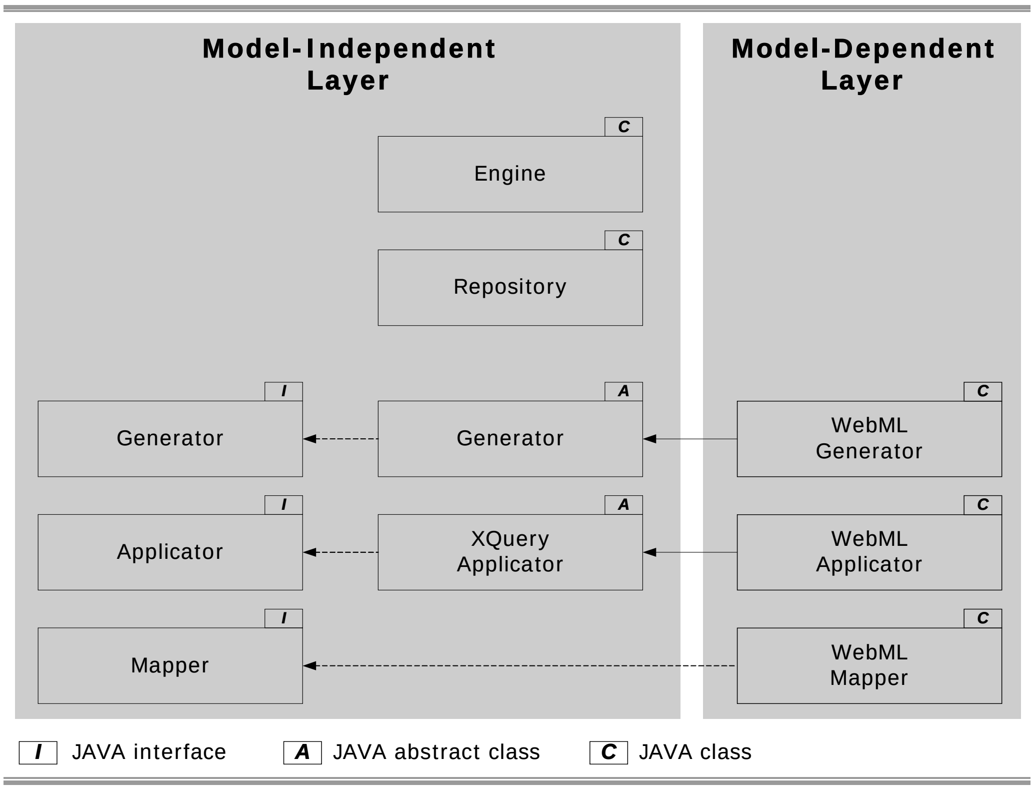


Figure 6.3: Allocation of process-implementations.

The model-independent layer provides interfaces that describe the processes of the TBE-engine, i.e. interfaces `Mapper`, `Applicator` and `Generator`. Basically, each process is represented by one method. Methods that represent model-independent processes are implemented in classes provided at the model-independent layer, i.e. classes `XQuery-Applicator` and `Generator`. Consequently, methods that represent model-dependent processes are implemented at the model-dependent layer, i.e. classes `WebMLMapper`, `WebMLApplicator` and `WebMLGenerator`, and extend classes provided at the model-independent layer if these classes implement the respective interface. SECTION 6.2.1.1, SECTION 6.2.1.2 and SECTION 6.2.1.3 describe the design of interfaces `Mapper`, `Applicator` and `Generator` and their implementing classes, respectively.

Class `Engine` uses the interfaces of the model-independent layer, for controlling the flow of processes that are involved in either compiling a transformer definition or performing a transformer application. Class `Engine` invokes methods of classes implemented at the model-dependent layer. For example if it is to map a WebML scheme to its logical representation, class `Engine` invokes the respective method of class `WebMLMapper`.

Class repository implements the repository of the TBE-engine and is implemented at the model-independent layer.

The interfaces provided at the model-independent layer are designed with regard to the following requirements:

1. *Generic interfaces for model-dependent processes*: Each modelling-language uses its own format for representing schemes, like, for example, XML is used by WebML for representing schemes. Clearly, different technologies are adequate for processing different formats, like, for example, XSLT is adequate for processing schemes represented in terms of XML. Therefore, it is desired to design generic interfaces for model-dependent processes, i.e. interfaces that do not constrain the choice of technologies used for implementation.
2. *Exchangeable execution engine*: In SECTION 6.1.2 it has been argued that different engines may be used for applying transformers, like, for example XQuery engines or Jess engines. Therefore, it is desired that interfaces are designed with regard to enable the usage of different execution engines.

The subsequent sections discuss the design of interfaces with regard to the previously identified requirements. Further, the classes implementing these interfaces at the two implementation layers of the TBE-engine are described.

6.2.1.1 Interface Mapper and implementing class

Interface `Mapper` provides methods that describe model-dependent processes `to logical representation`, `to native representation`, `extract TBE directives` and `extract customized directives`, as depicted in FIGURE 6.4.

In order to design generic interfaces representing model-dependent process, all parameters of such processes are of class `Object`. For example, method `toLogicalRepresentation` takes a scheme in any format as input and returns the logical representation of this scheme, again in any format. Therefore, the input-parameter and the return-parameter of method `toLogicalRepresentation` are both of the least specific type, i.e. of class `Object`.

```
public interface Mapper{
   public Object toLogicalRepresentation(Object scheme);
   public Object toNativeRepresentation(Object scheme);
   public Object extractTBEDirectives(Object scheme);
   public Object extractCustomizedDirectives(Object scheme);
}
```

Figure 6.4: Interface `Mapper`.

All methods of interface `Mapper` describe model-dependent processes. Therefore classes implementing this interface are exclusively allocated at the model-dependent layer. For example, class `WebMLMapper` implements interface `Mapper` for modelling language `WebML`.

6.2.1.2 Interface Applicator and implementing classes

Interface `Applicator` provides one method describing process `apply transformer` as depicted in FIGURE 6.5. The input scheme in logical representation, which is provided by model-dependent process `to logical representation` is of any format and therefore represented as an `Object`. The transformer to be applied is represented in terms of JAVA. Therefore, input-parameter `transDef` is of type `TransformerDefinition`. In order to enable the use of different execution engines, the return-parameter of method `applyTransformer`, i.e. the output-scheme, is an `Object` in order to be generic.

```
public interface Applicator{
   public Object applyTransformer(Object inputScheme,
                                  TransformerDefinition transDef);
}
```

Figure 6.5: Interface `Applicator`.

Class `XQueryApplicator` allocated at the model-independent layer implements interface `Applicator`, as depicted in FIGURE 6.6. This class uses an XQuery engine as execution engine. Therefore, methods `generateXQuery` and `executeXQuery` are implemented at class `XQueryApplicator`, for generating the XQuery and executing the XQuery, respectively. Method `executeXQuery` demands the input scheme in terms of XML in order to execute the XQuery. Since the input scheme provided by process `to Logical Representation` is of type `Object` a converter is required that converts the format of the input scheme to XML. Abstract method `convertInputScheme()` represents this converter. This method has to be implemented by applicators at the model-dependent layer, like for example, class `WebMLApplicator`. Note that an implementation of process `to logical representation` may already return an Object representing an XML document, such that method `convertInputScheme` has to perform just a type cast.

```
public abstract class XQueryApplicator implements Applicator {
   public Object applyTransformer(Object inputScheme,
                                  TransformerDefinition transDef);
   public String generateXQuery(TransformerDefinition transDef);
   public Document executeXQuery(Document inputScheme, String xQuery);
   public abstract Document convertInputScheme(Object inputScheme);
   public abstract String translateFunction(FunctionConstructionExp exp);
}
```

Figure 6.6: Abstract class `XQueryApplicator`.

Each applicator has to provide a model-dependent plugin called `functions executor` that executes model-dependent functions, like, for example function `concat()`. Abstract method `translateFunction` represents this plugin for the `XQueryApplicator`. This method takes a function construction expression in terms of JAVA as input and returns a `String` that represents the respective function in terms of XQuery. Method `generateXQuery()` calls method `translateFunction` each time a function construction expression is to be translated into terms of XQuery. This method has to be implemented by each applicator at the model-dependent layer that extends class `XQueryApplicator`.

6.2.1.3 Interface Generator and implementing classes

Interface `Generator` provides methods describing model-independent processes `generate transformer` and `individualize transformer`. Further, method `interpret-CustomizedDirectives()` represents the equally named model-dependent process. This interface is depicted in FIGURE 6.7.

The parameters of method `interpretCustomizedDirectives()` are of class Object, in order to be generic. Again, the format of templates and extracted TBE-directives has to be converted to corresponding JAVA representations. For example, method `convert-Template()` takes a template in logical representation in any format as input and returns the corresponding JAVA representation, i.e. a JAVA object of class `Logical-Representation`.

```
public interface Generator {
   public TransformerDefinition generate(
                         LogicalRepresentation qt,
                         LogicalRepresentation gt,
                         TBEDirectivesContainer tbeDirs);
   public void interpretCustomizedDirectives(
                         Object queryTemplate,
                         Object generativeTemplate,
                         Object tbeDirs,
                         Object customizedDirectives);
   public TBEDirectivesContainer convertTBEDirectives(Object tbeDirs);
   public LogicalRepresentation convertTemplate(Object template);
   public TransformerDefinition individualize(
                         TransformerDefinition transDef,
                         TBEDirectivesContainer tbeDirs);
}
```

Figure 6.7: Interface `Generator`.

Class `Generator` allocated at the model-independent layer implements those methods of interface `Generator` that describe model-independent processes, i.e. method `generate()` and method `individualize()`. FIGURE 6.8 depicts this class.

The model-dependent methods are implemented by classes allocated at the model-dependent layer. For example, class `WebMLGenerator`, implements the model-dependent processes and the converter methods of interface `Generator` for modelling-language `WebML`.

```
public abstract class Generator implements Generator {
  public TransformerDefinition generateTransformerDefinition(
                LogicalRepresentation queryTemplate,
                LogicalRepresentation generativeTemplate,
                TBEDirectivesContainer tbeDirs);
  public TransformerDefinition individualizeTransformerDefinition(
                TransformerDefinition transDef,
                TBEDirectivesContainer tbeDirs);
  public TBEDirectivesContainer convertTBEDirectives(Object tbeDirs);
}
```

Figure 6.8: Abstract class `Generator`.

6.2.2 Allocation of datastructure-implementations

This section describes the allocation of datastructure implementations to the implementation layers of the TBE-engine. Again, implementations of model-independent datastructures are provided at the model-independent layer and implementations of model-dependent datastructures are provided at the model-dependent layer. FIGURE 6.9 gives an overview of the various implementations.

At the model-independent layer of the TBE-engine, implementations of model-independent datastructures are allocated, i.e. datastructures `Logical Representation`, `Logical Representation Specification` and `Transformer Definition`.

All of these datastructures are implemented in terms of specifically designed JAVA classes as demanded by model-independent processes. Further, datastructure `Logical`

`Representation` is additionally implemented in terms of an XML-schema. This is because model-independent process `apply transformer` is implemented using XQuery such that the input and output of this process is a scheme's logical representation in terms of XML, which is specified by this XML-schema.

Developers that extend the implementation of the TBE-engine in order to support TBE for other modelling languages than WebML have to specify the logical representation of schemes of the respective modelling language once. For this purpose an XML-schema is provided that specifies the XML-representation of datastructure `Logical Representation Specification`. Thus, developers can conveniently define the specification of logical representations of schemes of the respective modelling language by means of an XML document that adheres to the provided XML-schema. This XML document is stored within the repository of the TBE-engine.

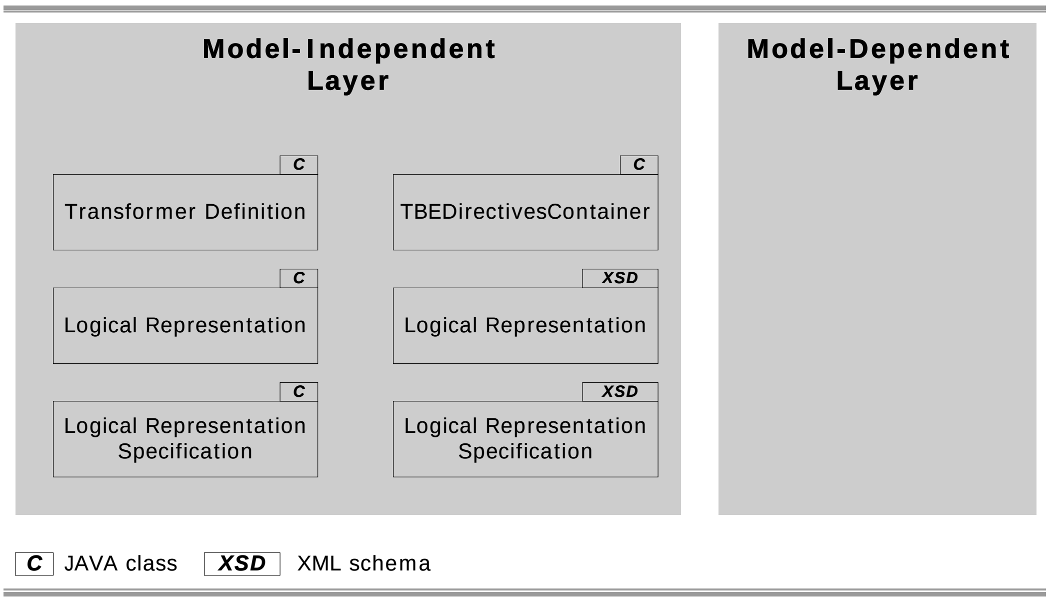


Figure 6.9: Allocation of datastructure-implementations.

For developing the model-dependent layer of the WebML TBE-engine no model-dependent datastructures needed to be newly implemented for the following reasons. The implementation of native schemes (templates) is provided by WebML by means of the WebML.dtd that specifies the structure of WebML schemes. Further, for representing customized-directive `anchor` a JAVA object of class String is used. Therefore no additional implementation is required.

6.3 Implementation of Datastructures

The implementations of datastructures are derived from the specification of the respective datastructure in terms of UML class diagrams and OCL expressions, which have been described in CHAPTER 5.

It is frequently required to violate some of the constraints specified for a datastructure while processing the respective datastructure. Therefore, these datastructures are implemented without considering the specified constraints. However, if it is desired to validate the input or output of a process, the constraints specified for the respective datastructure have to be fulfilled.

6.3.1 Datastructure *Logical Representation Specification*

This section describes the implementation of datastructure `Logical Representation Specification`. SECTION 6.3.1.1 argues that the structure of the specification of the logical representation of schemes, as specified by the architecture of the TBE-engine, needs to be refined. SECTION 6.3.1.2 and SECTION 6.3.1.3 illustrate the implementation of this datastructure in terms of specifically designed JAVA classes and in terms of an XML-schema, respectively. SECTION 6.3.1.4 describes the specification of the logical representation of WebML schemes, used within the TBE-engine.

6.3.1.1 Refinement of datastructure Logical Representation Specification

In order to perform a transformer application new universe members are generated as specified by construction expressions attached to new-element variables. If a new-element variable has a new construction expression attached, it is required to compute an identifier for each universe member that is generated according to the respective new-element variable.

***EXAMPLE 6.1**: Transformer `IndexPCForET` defines new-element variable `PC_ID` having construction expression `PC_ID = new (P)` attached. Therefore, each time a new page class is generated according to new-element variable `PC_ID` an identifier, like, for example, `page57`, has to be computed.*

For computing identifiers of newly generated scheme elements some order is required. For example, if the highest existing identifier of a page class is `page56` the next identifier is expected to be `page57`. In turn, it is required to know the structure of identifiers for computing new identifiers. The EBNF expression depicted in FIGURE 4.1 specifies the chosen structure of identifiers. The prefix and suffix of an identifier is some text. For example, the prefix of identifiers of page classes is `page`. Between the prefix and the suffix of identifiers an obligatory number is expected, which can be incremented in order to compute new identifiers.

```
identifier := prefix number suffix.
prefix      := string.
suffix      := string.
number      := ('0' ... '9') {'0' ... '9'}.
string      := ('a' ... 'z' | 'A' ... 'Z' | ' ' | '-' | '$' | '_')
               {'a' ... 'z' | 'A' ... 'Z' | ' ' | '-' | '$' | '_'}.
```

Figure 6.10: Syntax of identifiers.

Since the structure of identifiers of scheme elements is equal for all scheme elements of one sort the prefix and suffix of such identifiers can be specified for each sort of scheme element. In order to store these prefixes and suffixes datastructure `Logical Representation Specification` is refined in a way such that prefixes and suffixes are regarded as properties of universes. FIGURE 6.11 depicts the refined datastructure `Logical Representation Specification`.

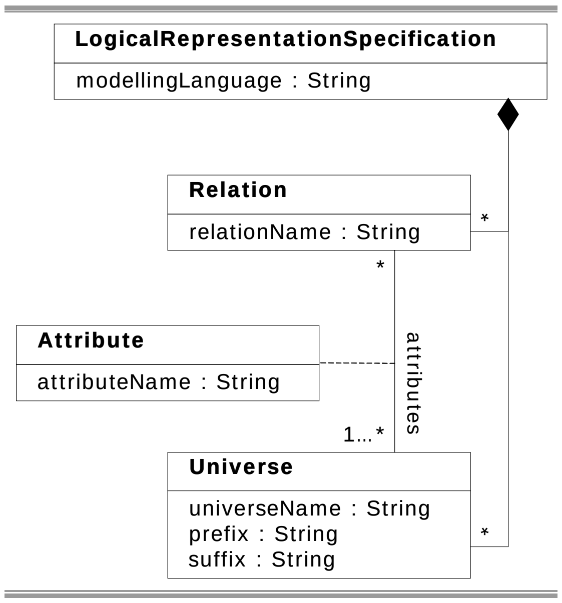


Figure 6.11: Refined datastructure `LogicalRepresentationSpecification`.

***EXAMPLE* 6.2**: *Consider, for example, that it is to compute an identifier for a newly generated page class. The domain of the new-element variable, which specifies the generation of the respective page class, is `page`. The prefix and suffix of identifiers of page classes can be looked up at the specification of the logical representation. In turn, a new identifier can be computed by the previously described approach.*

6.3.1.2 Implementation in terms of specifically designed JAVA classes

The JAVA classes, which are specifically designed for representing datastructure `Logical Representation Specification`, are derived from the UML class diagram specifying the respective datastructure. This UML class diagram is depicted in FIGURE 5.4. Each class of the respective UML class diagram results in a corresponding JAVA class. In order to express the properties and associations of classes specified in the respective UML class diagram corresponding member variables and getter-methods and setter-methods are developed.

```
public class LogicalRepresentationSpecification {
   public void setRelations(Collection relations);
   public Collection getRelations();
   public void setUniverses(Collection universes);
   public Collection getUniverses();
   public void setModellingLanguage(String language);
   public String getModellingLanguage();
   ..
}
```

Figure 6.12: Illustration of class `LogicalRepresentationSpecification`.

Thus, datastructure `Logical Representation Specification` is represented by JAVA classes `LogicalRepresentationSpecification`, `Universe`, `Relation` and `Attribute`. FIGURE 6.12 illustrates the method signatures of class `LogicalRepresentationSpecification`.

6.3.1.3 Implementation in terms of an XML-schema

The XML-schema representing datastructure `Logical Representation Specification` is again derived from the UML class-diagram specifying this datastructure. The resulting XML-schema is depicted in FIGURE 6.13.

```
 1 <xs:schema>
 2    <xs:element   name="LogicalRepresentationSpecification"
 3                type="LogicalRepresentationSpecificationType"/>
 4
 5    <xs:complexType name="LogicalRepresentationSpecificationType">
 6       <xs:sequence>
 7          <xs:element name="Universe" type="UniverseType"/>
 8          <xs:element name="Relation" type="RelationType">
 9       </xs:sequence>
10       <xs:attribute name="modellingLanguage" type="xs:string" use="required"/>
11    </xs:complexType>
12
13    <xs:complexType name="RelationType">
14       <xs:sequence>
15          <xs:element name="Attribute" type="AttributeType"/>
16       </xs:sequence>
17       <xs:attribute name="name" type="xs:string" use="required"/>
18    </xs:complexType>
19
20    <xs:complexType name="AttributeType">
21       <xs:attribute name="name" type="xs:string" use="required"/>
22       <xs:attribute name="universe" type="xs:string" use="required"/>
23    </xs:complexType>
24
25    <xs:complexType name="UniverseType" abstract="true">
26       <xs:attribute name="name" type="xs:string" use="required"/>
27       <xs:attribute name="prefix" type="xs:string" use="optional"/>
28       <xs:attribute name="suffix" type="xs:string" use="optional"/>
29    </xs:complexType>
30 </xs:schema>
```

Figure 6.13: XML-schema specifying datastructure `LogicalRepresentationSpecification`.

Each class of this UML class-diagram is represented by a corresponding XML-schema type. For example, UML class `LogicalRepresentationSpecification` is represented by the equally named XML-schema type depicted in LINE 2 of FIGURE 6.13. Further, properties of UML classes are expressed as XML-schema attributes within the corresponding XML-schema type. For example, XML-schema attribute `modellingLanguage` depicted in LINE 10 of FIGURE 6.13 represents property `modellingLanguage` of UML class `LogicalRepresentationSpecification`.

6.3.1.4 The specification of the logical representation of WebML schemes

The design of relations is basically affected by the requirement to keep the size of a scheme's logical representation in terms of XML small. The effect of the design of relations to the size of a scheme's logical representation in terms of XML is that the number of designed relations determines the number of relation members, required for representing the connections between scheme elements of a certain scheme. In turn, the number of relation members effects the size of the scheme's logical representation in terms of XML, i.e. the more relation members the bigger the size of the scheme's logical representation in terms of XML.

The requirement to keep the size of a scheme's logical representation in terms of XML as small as possible arises since the execution time of a transformer application depends on the size of the input scheme's logical representation in terms of XML. This is since an XQuery engine performs the transformer applications and the larger the input scheme's logical representation in terms of XML the longer the execution time of the XQuery statement, representing the respective transformer.

SECTION 6.3.1.4.1 introduces *merged relations*, i.e. relations that keep the size of a scheme's logical representation in terms of XML small. SECTION 6.3.1.4.2 introduce a guideline to the design of the logical representation of WebML schemes and illustrates the chosen design.

6.3.1.4.1 Merged relations

A straightforward approach to the design of relations is to design one separate relation for *each* connection between scheme elements. The drawback of this approach is that it results in a large number of relation members, i.e. a large input scheme, which is not intended as discussed in the previous section. Therefore *merged relations* are introduced, i.e. relations that represent more than one connection between scheme elements.

***Example 6.3**: This example illustrates how several connections are represented by one merged relation. The connection between entity types and their names and the connection between entity types and their super entity types are used for the purpose of illustration.*

TABLE 6.1 depicts relations `name(E × N)` *and* `superEntityType(E × N), which` *result from designing separate relations for the respective connections, in the upper left corner. The semantics of these relations is self-explanatory and therefore not further described. The bottom left corner of TABLE 6.1 depicts merged relation* `entity (E × N × E)`*. This merged relation expresses that every entity type may have a name and a super entity type. The right hand side of TABLE 6.1 depicts relation members that are required for representing the connections of entity type* `Author` *from the* `CMA` *example to its name and super entity type. In the upper right corner of TABLE 6.1 the relation members are depicted, which are required for representing the particular connections if one relation is designed per connection that represents the respective connection. In the bottom right corner of TABLE 6.1 the merged relation member is depicted that represents the same connections.*

Instead of two relation members and four attribute instances only one relation member and three attributes are required if a merged relation is designed.

	Relations	*Relation members*

separate	`name(E × N)`
	`superEntityType(E × E)`
		`name⟨ent2, Author⟩`
		`superEntity⟨ent2, User⟩`

merged	`entityType(E × N × E)`	`entityType⟨ent2, Author, User⟩`

Table 6.1: Separate relations versus merged relations.

6.3.1.4.2 Guideline to the design of the logical representation of WebML schemes

WebML defines numerous sorts of scheme elements. Since this diploma thesis aims at a prototype implementation of TBE for WebML, only a subset of all WebML scheme elements is represented by individual universes and relations. In particular, individual universes and relations have been designed for the sorts of WebML scheme elements required for structure modelling and hypertext modelling. All other sorts of WebML scheme elements are represented by a dummy universe, i.e. universe Dummy.

For designing the logical representation of WebML schemes the following guideline has been developed:

- For each sort of WebML scheme elements like, for example, entity types, page classes or attributes of entity types, one universe is designed.
- For each property of a particular sort of WebML scheme elements like, for example names of entity types or names of attributes, one universe is designed.
- For each sort of WebML scheme elements a merged relation is designed that represents the connections between the respective sort of WebML scheme elements and its properties, like, for example names of entity types.
- Merged relations, designed for a particular sort of WebML scheme elements, are extended in order to represent additional connections, like, for example the connection between an attribute of an entity type to the entity type it is defined at.

Applying the guideline to the design of the logical representation of WebML schemes achieves that only one relation is required for representing the connections of a particular WebML scheme element. Thus, the chosen design positively effects the duration of transformer applications, since few relation members are required for representing a particular scheme and thus the size of the input scheme's logical representation in terms of XML is kept small.

WebML DTD fragment specifying attributes of entity types	*Merged Relation*	
	Relation	*Universe*
`<!ATTLIST ATTRIBUTE`	`attribute (`	
`id ID #REQUIRED`	`attribute`	`Attribute`
`name CDATA #IMPLIED`	`name`	`Name`
`type (String|Number|...) #IMPLIED`	`type`	`Dummy`
`...`		
	`definedAt`	`EntityType`
`>`	`)`	

Table 6.2: Design of merged relations.

***EXAMPLE 6.4**: The left hand side of TABLE 6.2 depicts the WebML DTD fragment that specifies the properties of attributes of entity types. The right hand side of TABLE 6.2 depicts the merged relation* `attribute(Attribute, Name, Dummy EntityType)` *that represents the connections between properties of attributes of entity types and the respective attribute itself. For example, the connection between the name of an attribute of an entity type and the attribute itself is represented by attribute* `name`*. Further, the merged relation represents the connection between attributes of entity types and the respective entity type they are defined at. This connection is represented by attribute* `definedAt`*.*

APPENDIX A lists all merged relations designed for representing WebML schemes. EXAMPLE 6.5 illustrates the design of universes and relations. The complete specification of the logical representation of WebML schemes in terms of XML is listed in APPENDIX B.

```
<LogicalRepresentationSpecification modellingLanguage="WebML">
   <Universe name="Name" prefix="name"/>
   <Universe name="Attribute" prefix="att"/>
   <Universe name="EntityType" prefix="ent"/>
   ...
   <Relation name="attribute">
      <Attribute name="attribute" universe="Attribute"/>
      <Attribute name="name" universe="Name"/>
      <Attribute name="definedAt" universe="Entity"/>
   </Relation>
   ...
</LogicalRepresentationSpecification>
```

Figure 6.14: Fragment of the specification of the logical representation of WebML schemes in terms of XML.

***EXAMPLE 6.5**: FIGURE 6.14 depicts a fragment of the specification of the logical representation of WebML schemes in terms of XML, which specifies (i) the merged relation designed for connections of attributes of entity types and (ii) the universes that represent attributes of entity types and their properties.*

6.3.2 Datastructure *Logical Representation*

Datastructure `Logical Representation` is implemented (i) in terms of specifically designed JAVA classes and (ii) in terms of an XML-schema. The XML-schema representing datastructure `Logical Representation` is again derived from the UML class-diagram specifying this datastructure. FIGURE 6.15 depicts the resulting XML-schema. Each class of this UML class-diagram is represented by a corresponding XML-schema type. For example, UML class `LogicalRepresentation` is represented by the equally named XML-schema type depicted in LINE 2 of FIGURE 6.15. Further, properties of UML classes are again expressed by XML-schema attributes within the corresponding XML-schema type.

```
<xs:schema >
   <xs:element name="LR" type="LogicalRepresentationType"/>

   <xs:complexType name="LogicalRepresentationType">
      <xs:sequence>
         <xs:element name="RM" type="RelationMemberType"/>
         <xs:element name="UM" type="UniverseMemberType"/>
      </xs:sequence>
   </xs:complexType>

   <xs:complexType name="UniverseMemberType">
      <xs:attribute name="id" type="xs:string" use="required"/>
      <xs:attribute name="domain" type="xs:string" use="required"/>
   </xs:complexType>

   <xs:complexType name="RelationMemberType">
      <xs:sequence>
         <xs:element name="AI" type="AttributeInstanceType"/>
      </xs:sequence>
      <xs:attribute name="sig" type="xs:string" use="required"/>
   </xs:complexType>

   <xs:complexType name="AttributeInstanceType">
      <xs:attribute name="att" type="xs:string" use="required"/>
      <xs:attribute name="id" type="xs:string" use="required"/>
   </xs:complexType>
</xs:schema>
```

Figure 6.15: XML-schema specifying datastructure `Logical Representation`.

The JAVA classes, which are specifically designed for representing datastructure `Logical Representation` are again derived from the UML class diagram specifying the respective datastructure. Each class of the respective UML class-diagram results in a corresponding JAVA class. In order to express the properties and associations of classes specified in the respective UML class-diagram again corresponding member variables, getter-methods and setter-methods are defined.

```
public class LogicalRepresentation {

   public void setRelationMembers(Collection relations);

   public Collection getRelationMembers();

   public void setUniverseMembers(Collection universes);

   public Collection getUniverseMembers();

   ...

}
```

Figure 6.16: Illustration of class `LogicalRepresentation`.

Thus, datastructure `Logical Representation` is represented by JAVA classes `LogicalRepresentation`, `UniverseMember`, `RelationMember` and `AttributeIntance`. FIGURE 6.16 illustrates the method signatures of class `LogicalRepresentation`.

6.3.3 Datastructure *Transformer Definition*

Datastructure `Transformer Definition` is implemented in terms of specifically designed JAVA classes. The derivation of these JAVA classes from the respective UML class-diagram is analogous to the derivation of JAVA classes representing datastructures `Logical Representation` and `Logical Representation Specification`.

```
public class TransformerDefinition {
   public void setQueryTemplate(QueryTemplate qt);
   public QueryTemplate getQueryTemplate();
   public void setGenerativeTemplate(GenerativeTemplate qt);
   public GenerativeTemplate getGenerativeTemplate();
   ...
}
```

Figure 6.17: Illustration of class `TransformerDefinition`.

Datastructure `Transformer definition` is basically represented by classes `TransformerDefinition`, `QueryTemplate` and `GenerativeTemplate`. However, for representing a query template or a generative template additional classes were developed. For example, comparison constraints are represented by class `ComparisonConstraint`. FIGURE 6.17 illustrates the method signatures of class `TransformerDefinition`.

6.3.4 Datastructure *TBE-Directives*

Datastructure `TBE-directives` is implemented in terms of specifically designed JAVA classes. Class `TBEDirectivesContainer` is used for storing the TBE-directives of a scheme or transformer definition. FIGURE 6.18 depicts the method signatures of class `TBEDirectivesContainer`.

TBE-directives `tag parameter variable` and `tag result variable` are represented by ordinary `Strings`. Therefore class `TBEDirectivesContainer` provides collections for storing the names of variables that are tagged as parameter variables or result variables. FIGURE 6.18 depicts the corresponding getter-methods and setter-methods.

```
public class TBEDirectivesContainer {
   public void addParameterVariableName(String variableName);
   public Collection getParameterVariableName();
   public void addResultVariableName(String variableName);
   public Collection getResultVariableName();
   public void addConstraint(Constraint constraint);
   public Collection getConstraints();
   public void addConstructionExpression(ConstructionExpression exp);
   public Collection getConstructionExpressions();
   ...
}
```

Figure 6.18: Illustration of class `TBEDirectivesContainer`.

Further, class `TBEDirectivesContainer` provides for storing collections of constraints and construction expressions, i.e. for storing JAVA objects representing directives `constraint` and `construction expression`. Again, FIGURE 6.18 depicts the corresponding getter-methods and setter-methods.

The specifically designed JAVA classes representing directives `constraint` and `construction expressions` are derived from the UML-class diagrams that specify the structure of constraints and construction expressions, respectively. Since datastructure `Transformer Definition` specifies the structure of constraints and construction expressions, the corresponding JAVA classes are comprised within the implementation of datastructure `Transformer Definition`. These JAVA classes are reused for representing TBE-directives `constraint` and `construction expression`.

6.4 Implementation of Processes

This section describes the implementation of processes. The running example of transformer `IndexPCForET` and its application to the `CMA` scheme is used for illustrating the inputs and outputs of the processes.

SECTION 6.4.2 describes the processes for compiling the definition of transformer `IndexPCForET` in notation of WebML to a corresponding transformer definition in terms of TBE. SECTION 6.4.3 describes processes for performing the application of transformer `IndexPCForET` to the CMA scheme.

Before these processes are described, SECTION 6.4.1 introduces a *compressed logical representation* of WebML schemes.

6.4.1 Compressed logical representations of schemes

In SECTION 6.3.1.4 it has been argued that the duration of a transformer application depends on the size of the input scheme's logical representation in terms of XML.

For reducing the size of a scheme's logical representation in terms of XML, a compressed logical representation of schemes is introduced that does not materialize universe members.

```
<UM id="ent2" domain="Entity"/>  |
<UM id="Author" domain="Name"/>  |
                                 |
<RM sig="entity">                | <RM sig="entity">
   <AI att="entity" id="ent2">   |   <AI att="entity" id="ent2">
   <AI att="name" id="Author">   |   <AI att="name" id="Author">
</RM>                            | </RM>
```

Figure 6.19: Illustration of the compressed logical representations of schemes.

***EXAMPLE 6.6**: The left hand side of FIGURE 6.19 depicts the logical representation of entity type `Author`. The universe members and relation members are self-explanatory. The right hand side of FIGURE 6.19 depicts the compressed logical representation of entity type `Author`, i.e. universe members are not materialized.*

Clearly the size of a scheme's compressed logical representation is always smaller than the size of the scheme's (conventional) logical representation, since universe members are not materialized.

Although universe members are not materialized, they are virtually represented since they are referenced from attribute instances. Thus, it is possible to reconstruct universe members on basis of a scheme's compressed logical representation. For reconstructing universe members the specification of the logical representation of the respective scheme is required. EXAMPLE 6.7 illustrates how universe members are reconstructed.

```
<RM sig="entity">
   <AI att="entity" id="ent2">
   <AI att="name" id="Author">
</RM>
```

```
<Relation name="entity">
  <Attribute name="entity" universe="Entity">
  <Atribute name="name" universe="Name">
</Relation>
```

Figure 6.20: Reconstructing universe members.

***EXAMPLE 6.7**: The left hand side of FIGURE 6.20 depicts the compressed logical representation of entity type `Author`, i.e. the relation member that represents the connection between entity type `Author` and its `name`. The right hand side of FIGURE 6.20 depicts the signature of this relation member. Since the signature of relation member `entity` specifies that attribute `entity` references a member of universe `Entity`, it can be reconstructed, that a universe member with identifier `ent2` of universe `Entity` exists. Analogous it can be reconstructed that a universe member with identifier `Author` of universe `Name` exists.*

The processes of the TBE-engine are implemented with regard to use compressed logical representations of schemes.

6.4.2 Processes for compiling transformer definitions

This section describes the processes for compiling the definition of transformer `IndexPCForET` in notation of WebML to a corresponding transformer definition in terms of TBE.

6.4.2.1 Mapping schemes to their logical representation

This section illustrates process `to logical representation`. Besides mapping a scheme (template) in notation of WebML to its compressed logical representation, this process also resolves aliases for non-editable properties of WebML scheme elements.

Two XSLT stylesheets implement process to logical representation: The first stylesheet, called `pre-mapper`, resolves aliases as illustrated in EXAMPLE 6.8. The second stylesheet, called `main-mapper` performs the actual mapping as illustrated in EXAMPLE 6.9.

***Example 6.8**: The upper part of FIGURE 6.21 shows variables `ENT`, `ENT_ID` and `ATT_ID` in notation of WebML. For a complete listing of transformer `IndexPCForET's` query template and generative template in notation of WebML confer to APPENDIX C. and APPENDIX D, respectively. Entity type `ent1` represents variables `ENT` and `ENT_ID` as well as attribute `att2` represents variable `ATT_ID`. Further, property scheme elements `prop1` and `prop2` represent customized-directives that specify aliases for the identifier property of entity type `ent1` and attribute `att2`, respectively. The lower part of FIGURE 6.21 depicts again variables `ENT`, `ENT_ID` and `ATT_ID` but with resolved aliases, i.e. the result of applying the `pre-mapper` is shown. The identifier property of the entity type is now `$ENT_ID`. Thereby, the preceding dollars sign denotes that variable `ENT_ID` is a result variable. Consequently the identifier property of former attribute `att2` is now `ATT_ID`.*

```
<ENTITY id="ent1" name="$ENT">
   <ATTRIBUTE id="att2">
      <PROPERTY id="prop2" name="Identifier - ATT_ID" value="alias:"/>
   </ATTRIBUTE>
   <PROPERTY id="prop1" name="Identifier - $ENT_ID" value="alias:"/>
</ENTITY>
```

```
<ENTITY id="$ENT_ID" name="$ENT">
   <ATTRIBUTE id="ATT_ID"/>
</ENTITY>
```

Figure 6.21: Variables `ENT`, `ENT_ID` and `ATT_ID` in notation of WebML (top) and resolved aliases (bottom).

***EXAMPLE 6.9**: The upper part of FIGURE 6.22 shows the fragment of transformer `IndexPCForET's` query template in notation of WebML, which represents variables `ENT_ID`, `ENT` and `ATT_ID`. The lower part of this figure shows the logical representation of the same fragment of transformer `IndexPCForET's` query template in terms of XML, which results from applying the `main-mapper`. Thereby, universe members are not generated but referenced from the attribute instances of the resulting relation members. The first relation member has signature `entity` and represents therefore the connections of the entity type, depicted in the upper part of FIGURE 6.22. Consequently, attribute instances `entity` and `name` represent properties `id` and `name` of this entity type, respectively. The second relation member is interpreted analogously. APPENDIX E and APPENDIX F list the complete logical* *representations of transformer* `IndexPCForET's` *generative template and query template, respectively.*

```
<ENTITY id="$ENT_ID" name="$ENT">
   <ATTRIBUTE id="ATT_ID"/>
</ENTITY>
```

```
<RM sig="entity">
   <AI att="entity" id="ENT_ID"/>
   <AI att="name" id="ENT"/>
</RM>
<RM sig="attribute">
   <AI att="definedAt" id="ENT_ID"/>
   <AI att="attribute" id="ATT_ID"/>
</RM>
```

Figure 6.22: Fragment of the query template of transformer IndexPCForET in notation of WebML (top) and its logical representation (bottom).

6.4.2.2 Extracting TBE-directives

This section illustrates process `extract TBE directives`. Extracting TBE-directives comprises two steps. First the TBE-directives in textual form are filtered out of the respective scheme or template. Then the JAVA representations of these TBE-directives are generated by means of EBNF grammar parsing. The result, i.e. a `TBEDirectives-Container` is finally returned by process `extract TBE directives`.

```
<ENTITY id="$ENT_ID" name="$ENT"/>
<PROPERTY id="prop3" name="PC=concat(ENT,'Page');" value="expression:"/>
<PROPERTY id="prop4" name="IU=concat(ENT,'List');" value="expression:"/>
```

```
parameter_variable:ENT_ID;
parameter_variable:ENT;
construction_expression:PC=concat(ENT,'Page');
construction_expression:IU=concat(ENT,'List');
```

Figure 6.23: TBE-directives in notation of WebML (top) and their textual representation (bottom)

The XSLT stylesheet used for extracting TBE-directives is subsequently called `TBEDirectivesExtractor`. The EBNF grammar parser used for generating the JAVA representations of the extracted TBE-directives is subsequently called `TBEDirectives-`

`Parser`. This EBNF grammar parser was generated by AntLR [antlr04], which is an EBNF parser generator.

EXAMPLE 6.10: *The upper part of FIGURE 6.23 depicts a fragment of transformer `IndexPCForET´s` generative template in notation of WebML. The first line depicts variables `ENT_ID` and `ENT`, which are parameter variables as denoted by the preceding dollars sign. The first and the second line depicted in the lower part of FIGURE 6.23 shows the respective TBE-directives that are extracted by applying the `TBEDirectivesExtractor`. The textual representations of construction expressions, depicted in the third and fourth line of the lower part of FIGURE 6.23 are generated analogously. After the application of the `TBEDirectivesExtractor`, the `TBEDirectivesContainer` is filled with the JAVA representations of these textual TBE-directives via invoking the `TBEDirectivesParser`.*

6.4.2.3 Extracting customized-directives

This section illustrates process `extract customized directives`. The XSLT stylesheet used for extracting customized-directives is subsequently called `Customized-DirectivesExtractor`.

The `CustomizedDirectivesExtractor` filters out customized-directive `anchor`, which is annotated to the generative template. The JAVA representation of directive anchor is a simple `String` that represents the identifier of the scheme element that is used as the anchor within the respective generative template. This String is finally returned by process `extract customized directives`.

```
<PROPERTY id="prop4" name="ENT_ID;" value="anchor:"/>
```

```
anchor:ENT_ID;
```

Figure 6.24: Customized-directive `anchor` in notation of WebML (top) and its textual representation (bottom).

Customized-directives that specify aliases for non-editable properties of WebML scheme elements are not filtered out by this process, since aliases are immediately resolved by process to logical representation. This has already been discussed in SECTION 6.4.2.1.

***EXAMPLE* 6.11**: *The upper part of FIGURE 6.24 depicts directive anchor as it is defined in the generative template of transformer `IndexPCForET` in notation of WebML. The `CustomizedDirectivesExtractor` filters out the textual representation of the anchor directive, which is depicted in the lower part of FIGURE 6.24.*

6.4.2.4 Interpreting customized-directives

This section illustrates the implementation of process i`nterpret customized directives`, i.e. directives `anchor` and `alias`. Process to logical representation resolves aliases for non-editable properties of scheme elements as illustrated in SECTION 6.4.2.1. Therefore, the remainder of this section deals with interpreting customized-directive `anchor`.

SECTION 4.2.2 has introduced the basic idea of arranging new scheme elements in relation to the anchor by means of implicitly generating new coordinates for such scheme elements. In order to implicitly generate new coordinates, the templates of a particular transformer additionally define variables and construction expressions as follows:

- *Variables representing the anchor's coordinates*: In order to compute the coordinates of new scheme elements in relation to the coordinates of the anchor, the generative template requires the anchor's coordinates. Therefore, the query template additionally defines two result variables that represent the x-coordinate and the y-coordinate of the anchor scheme element, respectively. Analogous, the generative template additionally defines parameter variables corresponding to these result variables such that the anchor's coordinates retrieved by the query template get passed to the generative template.

- *Variables representing new coordinates*: In order to generate new coordinates for a new scheme element, the generative template additionally defines one new-element variable per coordinate that is to be generated. Further, each such new-element variable has a construction expression attached that specifies the computation of the coordinate.

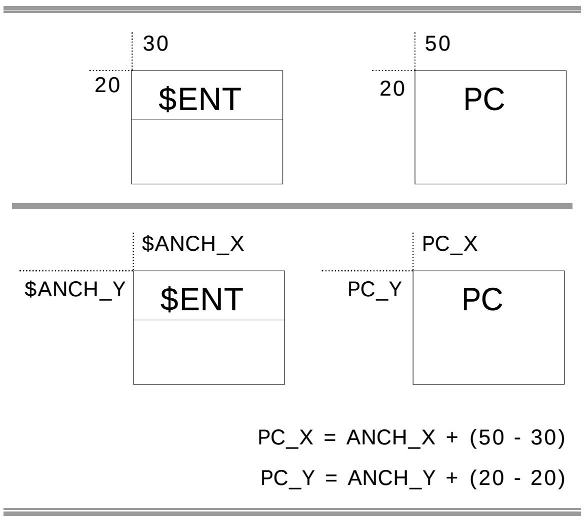


Figure 6.25: New-element variables and construction expressions for generating coordinates of new scheme elements.

EXAMPLE 6.12: *The upper part of FIGURE 6.25 depicts the generative template of transformer `IndexPCForET` in graphical notation. For reasons of conciseness details like, for example, attributes of entity types are neglected. Within transformer `IndexPCForET` the scheme element identified by `ENT_ID` is marked as anchor. Therefore the entity type represented by result variable `ENT` is the anchor within transformer `IndexPCForET`. This entity type has position (20/30) in the generative template. The page class represented by new-element variable `PC` has position (20/50).*

Generating the coordinates of new page classes requires to replace the concrete coordinates of new-element variable `PC` with variables. The lower part of FIGURE 6.25 depicts again transformer `IndexPCForET's` generative template. Yet, variable `PC_X` and variable `PC_Y` replace the concrete x-coordinate and y-coordinate of the page class represented by new-element variable `PC`, respectively. Analogous, variable `ANCH_X` and `ANCH_Y` replace the concrete x-coordinate and y-coordinate of the entity type, which is the anchor within the generative template, respectively. Further, the construction expressions for new-element variables `PC_X` and `PC_Y` are depicted in the lower part of FIGURE 6.25. These construction expressions are defined such that at application time a newly generated page class will be arranged relatively to the entity type in the same way as page class `PC` in the generative template is arranged relatively to entity type `ENT`.

Process `interpret customized directives` does not generate variables and construction expressions directly. Instead, this process adapts the query template and generative template, both in logical representation, and the TBE-directives in such a manner that the subsequent process `generate transformer` will generate these variables and construction expressions. EXAMPLE 6.13 illustrates how process `interpret customized directives` prepares the generation of variables representing the anchor's coordinates. EXAMPLE 6.14 illustrates how process `interpret customized directives` prepares the generation of variables representing the coordinates of new scheme elements.

```
<RM sig="EntityPos">
   <AI att="element" id="ENT_ID"/>
   <AI att ="X_Coord" id="30"/>
   <AI att ="y_Coord" id="20"/>
</RM>
```

```
<RM sig="EntityPos">
   <AI att="element" id="ENT_ID"/>
   <AI att="X_Coord" id="ANCH_X"/>
   <AI att="y_Coord" id="ANCH_Y"/>
</RM>
```

Figure 6.26: Preparing the generation of variables `ANCH_X` and `ANCH_Y`.

***EXAMPLE 6.13**: Process `interpret customized directives` prepares the generation of parameter variables representing the anchor's coordinates in the following two steps: (1) the concrete x-coordinate and y-coordinate of the anchor scheme element are identified within the generative template's logical representation. (2) The concrete values are replaced by variable names "`ANCH_X`" and "`ANCH_Y`", respectively. The result of step 1 and step 2 is shown in the upper and in the lower part of FIGURE 6.26, respectively. Note that the concrete coordinates are temporarily stored as they are required for deriving the construction expressions defining the coordinates of newly generated scheme elements as described in EXAMPLE 6.14.*

```
<RM sig="PagePos">
   <AI att="element" id="PC_ID"/>
   <AI att="X_Coord" id="50"/>
   <AI att="y_Coord" id="20"/>
</RM>
```

```
<RM sig="EntityPos">
   <AI att="element" id="PC_ID"/>
   <AI att="X_Coord" id="PC_X"/>
   <AI att="y_Coord" id="PC_Y "/>
</RM>
```

Figure 6.27: Preparation the generation of new-element variables for coordinates.

***EXAMPLE 6.14**: Process `interpret customized directives` prepares the generation of new-element variables representing the coordinates of new scheme elements in the following three steps: (1) the concrete x-coordinate and y-coordinate of scheme elements representing new-element variables are identified within the generative template's logical representation. The upper part of FIGURE 6.27 depicts the coordinates of page class `PC_ID` in logical representation, which represents the equally named new-element variable. (2) The concrete values are replaced by corresponding variable names, e.g. `"PC_X"` and `"PC_Y"` as depicted in the lower part of FIGURE 6.27 (3) Corresponding construction expressions are added to the `TBEDirectivesContainer` by means of JAVA objects, e.g. construction expressions `PC_X = ANCH_X + (50 - 30)` and `PC_Y = ANCH_Y + (20 – 20)`.*

6.4.2.5 Generating a transformer definition in terms of TBE

This section illustrates the implementation of model-independent process `generate transformer`. Again, an XML representation of datastructure `Transformer Definition` is used for the purpose of illustration. Note, however, that the TBE-engine works with JAVA representations of datastructure `Transformer Definitions`.

SECTION 6.4.2.5.1 illustrates the generation of the query template of transformer `IndexPCForET`. SECTON 6.4.2.5.2 illustrates the generation of the generative template of transformer `IndexPCForET`.

6.4.2.5.1 Generating the query template

For generating a transformer's query template the relation members comprised within the template's compressed logical representation are iterated.

Within each iteration step the following TBE-constructs are generated and added to the query template: (1) variables are generated according to the universe members (virtually) represented by the respective relation member. (2) a membership constraint is generated according to the respective relation member.

Finally, JAVA representations of comparison constraints and those of complex constraints, stored within the `TBEDirectivesContainer`, are added to the query template.

```
<RM sig="entity">
   <AI att="entity" id="ENT_ID"/>
   <AI att="name" id="ENT"/>
</RM>
```

```
<ResultVariable name="ENT_ID" domain="Entity">
<ResultVariable name="ENT " domain="Name">
```

Figure 6.28: Generating variables.

EXAMPLE 6.15: *The upper part of* FIGURE *6.29 depicts the relation member representing the connection of entity type* `ENT_ID` *to its name* `ENT`*. When this relation member is processed, variables* `ENT_ID` *and* `ENT` *are generated. Thereby, both variables are result variables since corresponding TBE-directives are specified, as shown in the previous section. Further result variable* `ENT_ID` *has domain* `Entity`*. The domain of a result variable is determined analogous to the determination of the universe when a universe member is reconstructed. The reconstruction of universe members by means of analyzing the specification of the logical representation of schemes has been explained in the* SECTION *6.4.1.*

```
<RM sig="entity">
   <AI att="entity" id="ENT_ID"/>
   <AI att="name" id="ENT"/>
</RM>
```

```
<MembershipConstraint sig="Entity">
   <Assignment att="entity" var="ENT_ID"/>
   <Assignment att="name" var="ENT"/>
</MembershipConstraint>
```

Figure 6.29: Generating membership constraints.

EXAMPLE 6.16: *The upper part of* FIGURE *6.29 depicts a relation member. The lower part of this figure depicts the membership constraint that is generated according to the particular relation member.*

6.4.2.5.2 *Generating the generative template*

The generation of the generative template is analogous to the generation of the query template. The difference is, that instead of result variables and non-result variables, parameter variables and new-element variables are generated, respectively. Further, instead of membership constraints, relation constructors are generated. FIGURE 6.30 depicts a relation constructor in the lower part that is generated according to the relation member depicted in the upper part.

```
<RM sig="page">
   <AI att="page" id="PC_ID"/>
   <AI att="name" id="PC"/>
</RM>
```

```
<RelationConstructor sig="page">
   <Assignment att="page" var="PC_ID"/>
   <Assignment att="name" var="PC"/>
</MembershipConstraint>
```

Figure 6.30: Generating relation constructors.

For generating new-element variables, the corresponding construction expressions have to be generated too. EXAMPLE 6.17 illustrates the generation of new-element variables.

```
<RM sig="page">
   <AI att="page" id="PC_ID"/>
   <AI att="name" id="PC"/>
</RM>
```

```
<NewElementVariable name="PC_ID" domain="Page">
   </New>
</NewElementVariable>

<NewElementVariable name="PC " domain="Name">
   <Function name="concat">
      <Argument type="ParameterVariable" val="ENT"/>
      <Argument type="LiteralValue" val="Page"/>
   </Function>
</NewElementVariable>
```

Figure 6.31: Generating new-element variables.

***EXAMPLE 6.17**: The lower part of FIGURE 6.31 depicts new-element variables that are generated on basis of the relation member depicted in the upper part of FIGURE 6.31. Thereby, for new-element variable `PC_ID` a new construction expression is implicitly generated, since no other construction expression is specified by means of TBE-directives. In contrast, for new-element variable `PC` a function construction* *expression is specified by means of TBE-directives. Therefore, this function construction expression is attached to new-element variable* `PC`.

6.4.3 Processes for applying transformers

This section describes the implementation of processes for performing transformer applications. Processes `to logical representation` and `extract TBE directives` have already been described in the previous section. Therefore, this section only explains processes `individualize transformer`, `apply transformer` and `to native representation`.

6.4.3.1 Individualizing the transformer definition

This section describes process `individualize transformer`. The input for this process is a `Transformer Definition` in terms of TBE and a `TBEDirectives-Container` that stores the JAVA representations of application specific-constraints and application-specific construction expressions. The application-specific constraints and application-specific construction expressions are inserted into the transformer definition and afterwards passed to process `apply transformer`.

```
<TransformerDefinition name="IndexPCForET">
  <QueryTemplate>
    ...
    <ComplexConstraint logCon="or">
      <ComparisonConstraint var="ENT" litVal="Paper" op="not equals"/>
      <ComparisonConstraint var="ENT" litVal="Author" op="not equals"/>
    </ComplexConstraint>
  </QueryTemplate>
  ...
</TransformerDefinition>
```

Figure 6.32: Individualized definition of transformer `IndexPCForET`.

***EXAMPLE 6.18**: The application of transformer* `IndexPCForET` *to the* `CMA` *scheme is individualized by application-specific constraint* `or((ENT != 'Paper'), (ENT != 'Author'))`. *This application-specific constraint is extracted by process* `Extract TBE Directives` *and passed to process* `Individualize Transformer` *in terms of*

JAVA objects stored within the `TBEDirectivesContainer`. FIGURE 6.32 illustrates the definition of transformer `IndexPCForET` after this individualization.

6.4.3.2 Performing the transformer application

The TBE-engine performs the application of a transformer by generating an XQuery statement on basis of an individualized transformer definition in terms of TBE. The choice of XQuery, as the language for expressing transformer definitions, has been discussed in SECTION 6.1.2. Then, the TBE-engine executes this XQuery statement on the input scheme's logical representation. Thus, the input of process `apply transformer` is (i) an individualized transformer definition in terms of TBE represented by specific JAVA objects and (ii) the input scheme in logical representation in terms of XML. The output of process apply transformer is the logical representation of the output scheme, again in terms of XML.

The remainder of this section illustrates the XQuery statement that expresses the individualized definition of transformer `IndexPCForET`, which is simply referred to as *the XQuery statement* subsequently. Of course, the result of the execution of this XQuery statement to the logical representation of the CMA scheme, i.e. the output of process `apply transformer`, is illustrated as well.

The XQuery statement is basically separated into the *`Helper Variables Section`*, the *`Query Template Section`* and the *`Generative Template Section`*, as illustrated in FIGURE 6.33. Each of these sections is explained in the following.

The `helper variables section` is used for declaring XQuery-variables that are required in either of the subsequent sections. Since the input scheme's logical representation is compressed, i.e. universe members are not separately represented, the universe members, which are required in the `query template section` or the `generative template section` need to be reconstructed on basis of the relation members, comprised within the input scheme's logical representation. Therefore, one helper XQuery-variable is declared for each sort of universe members, which is required within either of the subsequent sections, that represents all universe members of the respective sort.

Helper Variables Section

```
let  $inputLR :=  input()/LR,
     $inputRM :=  $inputLR/RM,
     $rEntity := $inputRM[@sig eq 'Entity'],
     $uEntity := distinct-values(
        $inputRM[@sig eq 'Entity']/AI[@att eq 'entity']/@id union
        $inputRM[@sig eq 'Attribute']/AI[@att eq 'definedAt']/@id ...),
     $uPage := ...,
     $uName := ...,
```

Query Template Section

```
     $QTRes := (
        for      $ENT    in $uName,
                 $ENT_ID in $uEntity
        where    (exists(
          for    $rEntity_ in $rEntity
          where  $rEntity_/AI[@att eq 'entity' and @id eq $ENT_ID] and
                 $rEntity_/AI[@att eq 'name' and @id eq $ENT]
          return $rEntity_)
                 ) and
                 (($ENT eq 'Paper') or ($ENT eq 'Author'))
        return          <tuple>
                              <ENT>{$ENT}</ENT>
                              <ENT_ID>{$ENT_ID}</ENT_ID>
                        </tuple>
     )
```

Generative Template Section

```
return <LR> {
   let  $hPage :=max(   for $id in $uPage
                        return substring-after($id, 'page')),
        $GTRes := (
        for  $counter in 1 to count($QTRes/tuple)
        let  $ENT    := data(item-at($QTRes/tuple,$counter)/ENT),
             $ENT_ID := data(item-at($QTRes/tuple,$counter)/ENT_ID),
             $PC_ID  := concat('page', string($hPage+1+(1*($counter-1)))),
             $PCN    := concat($ENT ,'Page'),
          return <RM sig="Page">
                        <AI att="page" id="{$PC_ID}"/>
                        <AI att="name" id="{$PC}"/>
                 </RM>
        )
   return $GTRes/RM union $inputRM
}</LR>
```

Figure 6.33: Partial XQuery statement representing transformer `IndexPCForET`.

```
<REL sig="Entity">
   <ATT att="entity" universe="Entity"/>
   ...
</REL>
```

Figure 6.34: Partial signature of relation `Entity`.

EXAMPLE 6.19: *XPath-expressions collect universe members that are referenced from attribute instances. The XPath-expression depicted in LINE 5 and LINE 6 of FIGURE 6.33 retrieves all attribute instances, which reference universe members of sort entity type. Universe members of sort entity type are represented by helper XQuery-variable `$uEntity`, which is depicted in LINE 3 of FIGURE 6.33. The information, which particular attribute instances have to be considered by the XPath-expression is captured within the specification of the logical representation of WebML schemes. FIGURE 6.34 depicts the signature of relation `Entity` partially. Since attribute `entity` denotes to represent entity types, it is known that the XPath-expression depicted in LINE 5 and LINE 6 of FIGURE 6.33 has to consider instances of attribute `entity` of relation `Entity`.*

The `query template section` is used for computing the tuples of result variables, which fulfill all specified constraints. The tuples of result variables are represented in a particular XQuery-variable called `QTRes`, which is depicted in LINE 8 of FIGURE 6.33. The `query template section` expresses the evaluation of a query template, which has been conceptually described in SECTION 3.2.1.3. For explanative purpose we abstract here from query optimization. The optimized query is listed in APPENDIX G.

The `query template section` is structured as follows. Within a for-loop all combinations of result variables are computed, as illustrated in LINE 10 of FIGURE 6.33. Within each iteration of this for-loop, conditions that express the constraints of the query template are checked by a where-clause. This where-clause is depicted in LINE 11 of FIGURE 6.33. Finally, those tuples of result variables, which fulfill all specified constraints, are returned, as illustrated in LINE 18 of FIGURE 6.33. EXAMPLE 6.20, EXAMPLE 6.21 and EXAMPLE 6.22 illustrate how the query template of the individualized definition of transformer `IndexPCForET` is expressed in terms of XQuery. Again an XML representation of relevant parts of this transformer definition is used for illustration. Note, however, that the TBE-engine uses JAVA representations of transformer definitions in order to generate the corresponding XQuery.

```
<ResultVariable name="ENT_ID" domain="Entity"/>
<ResultVariable name="ENT" domain="Name"/>
```

Figure 6.35: XML representation of result variable `ENT_ID` and `ENT`.

EXAMPLE 6.20: *FIGURE 6.35 depicts result variables `ENT_ID` and `ENT` of transformer `IndexPCForET`. These result variables are expressed in terms of XQuery within the for-loop of the query template section, in order to compute all combinations of result variables. This is depicted in LINE 9 and LINE 10 of FIGURE 6.33. Further, these result variables are considered within the return-clause as depicted in LINE 19 and LINE 20 of FIGURE 6.33.*

```
<MembershipConstraint sig="Entity">
   <Assignment att="entity" var="ENT_ID"/>
   <Assignment att="name" var="ENT"/>
</MembershipConstraint>
```

Figure 6.36: XML representation of a membership constraint of transformer IndexPCForET.

EXAMPLE 6.21: *The membership constraint depicted in FIGURE 6.36 determines, that only such combinations of result variables are returned by the `query template section`, where the name represented by variable `ENT` belongs to the entity type represented by variable `ENT_ID`. Membership constraints are expressed in terms of XQuery by means of exists-conditions that check whether a particular relation member exists. The exists-condition, which expresses the membership constraint depicted in FIGURE 6.36, is depicted in LINE 11 to LINE 15 of FIGURE 6.33.*

```
<ComplexConstraint logCon="or">
   <ComparisonConstraint var="ENT" litVal="Paper" op="not equals"/>
   <ComparisonConstraint var="ENT" litVal="Author" op="not equals"/>
</ComplexConstraint>
```

Figure 6.37: XML representation of a complex constraint of individualized transformer `IndexPCForET`.

EXAMPLE 6.22: *Complex constraints and comparison constraints are expressed in terms of XQuery by means of predefined XQuery-functions. For example, XQuery-function `eq` is used for comparing XQuery variabes on equality. The complex constraint depicted* *in FIGURE 6.37 is expressed in terms of XQuery as depicted in LINE 18 of FIGURE 6.33.*

The `generative template section` is used for generating new relation members and new universe members. Since a compressed logical representation of schemes is used within process `apply transformer`, newly generated universe members are implicitly represented by newly generated relation members. The `generative template section` expresses the instantiation of a generative template for each tuple of result variables returned by the respective `query template section`.

The `generative template section` works as follows. Each tuple of result variables is iterated within a for-loop, as depicted in LINE 26 of FIGURE 6.33. Within each iteration of this for-loop, XQuery-variables are defined, which represent parameter variables and new-element variables, within a let-clause as depicted from LINE 28 to LINE 31 in FIGURE 6.33. At the definition of XQuery variables, which represent new-element variables, the corresponding construction expression is considered. Further, new relation members are generated according to the respective relation constructors and returned within each iteration of the for-loop. The outcome of the `generative template section` is stored into XQuery variable `GTRes`, as depicted in LINE 26 of FIGURE 6.33. EXAMPLE 6.23, EXAMPLE 6.24 and EXAMPLE 6.25 illustrate how the generative template of the individualized definition of transformer `IndexPCForET` is expressed in terms of XQuery. Again, XML representations are used for illustrating relevant parts of this transformer definition. Although, the TBE-engine uses JAVA representations of transformer definitions for expressing transformer definitions in terms of XQuery.

```
<ParameterVariable name="ENT_ID" domain="Entity"/>
<ParameterVariable name="ENT" domain="Name"/>
```

Figure 6.38: XML representation of parameter variables `ENT_ID` and `ENT`.

EXAMPLE 6.23*: FIGURE 6.38 depicts parameter variables `ENT_ID` and `ENT` in terms of XML. The corresponding XQuery-variables are depicted in LINE 28 and LINE 29 of FIGURE 6.33, respectively. Thereby, the actual value of a parameter variable is given by the value of the corresponding result-variable, which is passed by the `query template section`. The XPath-expression depicted in LINE 28 and LINE 29 of FIGURE 6.33 retrieves the values of the XQuery-variables representing parameter variable `ENT` and `ENT_ID`, respectively.*

```
<NewElementVariable name="PC_ID" domain="Page">
   </New>
</NewElementVariable>
<NewElementVariable name="PC " domain="Name">
   <Function name="concat">
      <Argument type="ParameterVariable" val="ENT"/>
      <Argument type="LiteralValue" val="Page"/>
   </Function>
</NewElementVariable>
```

Figure 6.39: XML representation of new-element variables `PC_ID` and `PC`.

EXAMPLE 6.24: *FIGURE 6.39 depicts new-element variables `PC_ID` and `PC` in terms of XML and their attached construction expressions. The corresponding XQuery-variables are depicted in LINE 30 and LINE 31 of FIGURE 6.33, respectively. The values of these XQuery-variables are computed by XQuery-function calls, expressing the construction expressions attached to the respective new-element variables. The call of XQuery-function `concat($ENT, 'Page')`, for example, expresses the function construction expression of new-element variable `PC`.*

The XQuery-function call depicted in LINE 30 of FIGURE 6.33 expresses the new construction expression attached to new-element variable `PC_ID`. Its semantics is to increment the highest existing identifier of page classes, comprised within the input scheme, and to concatenate the prefix of identifiers of page classes, i.e. `"page"`, to this incremented identifier. Therefore, identifiers matching the general syntax of identifiers, which has been specified in SECTION 6.3.1.1 are generated, e.g. `"page5"`. Thereby, the highest existing identifier of page classes is represented by XQuery-variable `$hPage`. Further, the prefix of identifiers of members of a particular universe is determined by the specification of the logical representation of WebML schemes.

```
<RelationConstructor sig="Page">
   <Assignment att="page" var="PC_ID"/>
   <Assignment att="name" var="PC"/>
</RelationConstructor>
```

Figure 6.40: XML representation of a relation constructor of transformer `IndexPCForET's` generative template.

EXAMPLE 6.25: *Relation constructors are expressed in terms of XQuery by means of returning XML elements, which represent the relation member to be generated according to the respective relation constructor.* FIGURE *6.41 depicts the XML elements that are generated according to the relation constructor depicted in* FIGURE *6.33, starting at* LINE *32 up to* LINE *35. Thereby, the identifiers of universe members that are referenced from the generated attribute instances are the values of corresponding new-element variables.*

Relation members that are generated by the `generative template section` are represented by XQuery-variable `GTRes`. Finally, the union of all newly generated relation members and all relation members comprised within the input scheme's logical representation is computed. The computation of the union of relation members is depicted in LINE 39 of FIGURE 6.33. The resulting set of relation members is the overall result of the execution of the XQuery statement and in turn the output of process `apply transformer`.

```
<RM sig="Entity">
   <AI att="entity" id="ent1"/>
   <AI att="name" id="Paper"/>
</RM>
<RM sig="Page">
   <AI att="page" id="page1"/>
   <AI att="name" id="PaperPage"/>
</RM>
```

Figure 6.41: Fragments of the output of process `apply transformer`.

EXAMPLE 6.26: FIGURE *6.41 depicts fragments of the output of process* `apply transformer` *resulting from applying transformer* `IndexPCForET` *to the* CMA *scheme. The relation member depicted in the upper part of* FIGURE *6.41, represents entity type* `Paper` *and the connection of this entity type to its name. The relation member depicted in the lower part of* FIGURE *6.41, represents newly generated page class* `PaperPage` *and the connection of this page class to its name. The second relation member was generated by executing the XQuery statement, whereas the first relation member was given by the input scheme, i.e. the* CMA *scheme.*

6.4.3.3 Mapping the output scheme to its native representation

This section describes process `to native representation`, i.e. it is described how the output scheme's logical representation is mapped to its representation in terms of WebML.

An XSLT stylesheet implements process `to native representation`, which is subsequently called `mapper`.

```
<RM sig="page">
   <AI att="page" id="page1"/>
   <AI att="name" id="Paper"/>
</RM>
```

```
<PAPER id="page1" name="Paper"/>
```

Figure 6.42: Fragment of the output scheme in notation of WebML (bottom) and its logical representation (top).

The `mapper` traverses the relation members of the output scheme and generates the represented WebML scheme elements. For example, relation members of signature `page` result in a corresponding page class. Therefore, the `mapper` comprises one template rule for each relation, i.e. sort of scheme elements.

***EXAMPLE 6.27**: The upper part of FIGURE 6.42 depicts the relation member representing the connection from page class `page1` to its name. The lower part of FIGURE 6.42 depicts the WebML scheme element resulting from applying the mapper. The actual mapping is self-explanatory.*

7 Related Work

Content

7.1 Approaches to scheme transformations 147

7.1.1 Scheme transformations in WebML 148
7.1.2 Scheme transformations in ARANEUS 148
7.1.3 Scheme transformations in OO-H 148
7.1.4 Comparison of approaches to scheme transformations 150

7.2 The graphical editor *TransEd* 152

7.2.1 Transformer definition mode 153
7.2.2 Scheme development mode 155

This chapter presents work related to the prototype TBE-engine developed in this diploma thesis. SECTION 7.1 compares TBE to other approaches to scheme transformations. SECTION 7.2 briefly describes the graphical editor TransEd that facilitates the definition of schemes and templates for WebML. This graphical editor is currently being developed within a related diploma thesis [Wab04] and can be pluged in seamlesly as graphical editor within the TBE-system developed in this diploma thesis. In Chapter 4 it has been described how WebRatio can be used off-the-shelf as graphical editor. Allthough this approach works it is quite inconvenient for modelleres to annotate TBE-directives in textual form. TransEd provides for a convenient, graphical specification of TBE-directives.

7.1 Approaches to scheme transformations

Approaches to scheme transformations are rare [CFB00, GCP01, MAM03] and, besides that of TBE, provided by model-driven development methods. The subsequent sections illustrate the respective approaches. Finally SECTION 7.1.4 compares the respective approaches.

7.1.1 Scheme transformations in WebML

The development method WebML [CFB00] provides the possibility of generating a default hypertext scheme out of a content scheme. This approach to scheme transformations is based on a fixed and predefined set of transformation rules. Each transformation rule defines the generation of hypertext scheme elements according to one content scheme element. Examples for such transformation rules are:

- "For each entity type, generate a data unit that contains all of the entity type's attributes" [LS04]. This rule aims at presenting details of a single member of a certain entity type.
- "For each entity type, generate an index unit" [LS04]. This rule aims at presenting all members of a certain entity type as a list.

However, the approach to scheme transformations defined in WebML is not implemented in the supporting CASE tool.

7.1.2 Scheme transformations in ARANEUS

The approach to scheme transformations defined in the development method ARANEUS [MAM03] is comparable to the one defined in WebML. Predefined scheme transformations are provided for enabling quick prototyping and achieved through a fixed and predefined set of transformation rules.

One such transformation rule is, for example, "PS-FROM-NE", which leads to the generation of a page class from an entity type. The name of the generated page class is thereby computed by concatenating string "Page" to the name of the entity type. Note that "PS" and "NE" is short for page scheme and navigational entity, which correspond to page classes and entity types, respectively. HOMER, the CASE tool supporting ARANEUS implements the defined approach to scheme transformations.

7.1.3 Scheme transformations in OO-H

The development method OO-H [GCP01] provides a set of patterns for performing scheme transformations. These patterns are textually described in the style suggested by Buschmann [BMRSS96] and grouped into three main categories as follows:

- *Information patterns:* for performing modelling tasks that originate from providing the user with application context information. One such pattern is, for example, the "Location" pattern where a page is refined in some way to provide the user with information about his or her current location context inside the application.
- *Interaction patterns:* for performing modelling tasks that originate from providing user interface communication issues. The "Index" pattern, for example, refines a page to list all members of a certain entity type.
- *User scheme evolution patterns:* for performing modelling tasks that originate from covering advanced structural features. The "Multiview" pattern, for example, lets the designer present various views, i.e. pages, of the same set of entity types.

Such patterns are implemented by one or more transformation rule instantiations. A transformation rule can be regarded as a template specifying criterions, which must be fulfilled by transformation rule instantiations in order of being a valid implementation for the particular pattern. Both, the transformation rule and the transformation rule instantiation are specified in an OCL like syntax [oql04]. The transformation rule instantiation specifies a sequence of scheme modification operations tailored to the internal representation of OO-H schemes. Thus, applying a particular pattern, i.e. a particular scheme transformation, is achieved through executing the sequence of scheme modification operations specified by the corresponding transformation rule instantiation.

```
[
landMarkPage = <APDScheme> -> select(name = <landMarkPageName>);
sourcePages = <APDScheme> -> select(type = 'Tstruct');
sourcePages -> AddLink(<landMarkPage>);
]
```

Figure 7.1: Transformation rule for the `Landmark` pattern.

A transformation rule and its possible instantiation is subsequently illustrated by the fictive example of a "Landmark" pattern. The "Landmark" pattern specifies that a certain hypertext page is to be reachable from every other hypertext page. Thus, every hypertext page has to define a link pointing to the landmark hypertext page.

***Example 7.1**: Figure 7.1 depicts the `Landmark` pattern's transformation rule. The first line specifies that a certain hypertext page (`landMarkPage`) is selected. In OO-H a hypertext scheme enhanced with presentational features is called abstract presentation diagram. The `landMarkPage` is identified by selecting the one hypertext page from the `APDScheme`, which is named `landMarkPageName`. The second line specifies the selection of all hypertext pages (sourcePages), to which a link to the `landMarkPage` will be added.*

In OO-H all scheme elements are typed. Thereby, scheme elements of type `Tstruct` correspond to hypertext pages. Thus, the set of `sourcePages` is identified by selecting all scheme elements from the APDScheme that are of type `Tstruct`. The last line specifies, that a link to the `landMarkPage` is added to all `sourcePages`.

```
landMarkPage = ConfScheme -> select(name = 'ConfPage');
sourcePages = ConfScheme -> select(type = 'Tstruct');
sourcePages -> AddLink(landMarkPage);
```

*Figure 7.2***:** A possible transformation rule instantiation for the "Landmark" pattern.

***Example 7.2**: A possible transformation rule instantiation, specifying that a hypertext page called `ConfPage` has to be used as landmark page, for the "Landmark" transformation rule is depicted in Figure 7.2. The interpretation of this example is straightforward. The first line selects the landmark page called `ConfPage` from the conference scheme (ConfScheme). The remaining lines are copied from the transformation rule and therefore not further explained.*

7.1.4 Comparison of approaches to scheme transformations

The introduced approaches to scheme transformations are compared using general requirements on the specification and application of transformers [Lec04]. These requirements are subsequently listed:

- *Easy to understand*: Transformers shall be easy to understand for average modellers. This is basically a matter of disciplined documentation, which in turn should be facilitated by the system with which transformers are defined.

- *Easy to define*: Transformers shall be easy to define for average modellers. This requires a proper representation of schemes and a proper language for manipulating these schemes.
- *Flexibility*: with each application of a transformer, it shall be possible to adapt its behaviour to the requirements of the particular application. This concerns two aspects, namely (1) to localize the part of a scheme to be considered/affected, and (2) to adapt the way new scheme elements are generated.
- *Proper expressive power*: Transformer shall provide enough expressive power such that their applicability is not restricted to some selected, simple modelling tasks.
- *Independency of particular modelling languages*: As there is a large number of different modelling languages, it would be beneficial, if the approach to the definition and application of transformers could be followed in any modelling language. This requirement is orthogonally to the before mentioned ones.

The results of the comparison of the respective approaches to scheme transformations are summarized in TABLE 7.1. Thereby, Y denotes that a distinctive approach satisfies a certain requirement and N denotes the opposite. Subsequently for each approach it is explained which requirements are satisfied and which not.

Requirement	*Approach*			
	WebML	*ARANEUS*	*OO-H*	*TBE*
Easy to understand	Y	Y	Y	Y
Easy to define	N	N	N	Y
Proper expressive power	N	N	N	Y
Independency	N	N	N	Y
Flexibility	N	N	Y	Y
Implementation	N	Y	Y	N

Table 7.1: Comparison of approaches to scheme transformations.

The approaches of WebML and ARANEUS both target on enabling quick prototyping. Although their scheme transformations are easy to understand, they have the disadvantages of being fixed and predefined. Thus it is neither in WebML nor in ARANEUS possible to define individual schema transformations or to adapt schema transformations at application time. The expressive power of these approaches is relatively small, as just a fixed set of scheme transformations aiming at quick prototyping are supported. Further, these approaches cannot be adapted to cooperate with other modelling languages, as they are tailored to the scheme elements provided by the particular modelling language. Besides those common disadvantages the approach of WebML further lacks of an implementation.

The approach offered by OO-H enables the flexible use of scheme transformations, called transformation rules, as multiple implementations, called transformation rule instantiations, are possible. Furthermore, the description of schema transformations, called patterns, in the style suggested by Buschmann [BMRSS96] makes them easy to be understood. Still this approach has major disadvantages. First of all the definition of scheme transformations, i.e. transformation rules, is inconvenient, as a special OCL [oql04] like syntax has to be used. Second, there is a lack of proper expressive power, as only scheme transformations can be defined, that manipulate hypertext- and presentation schemes, which are melted to one scheme, called abstract presentation diagram. As scheme transformations are defined as a sequence of scheme modification operations over an internal representation of OO-H schemes, it is not possible to adopt the approach to cooperate with other modelling languages.

The approach offered by TBE, which is described in CHAPTER 2, for performing scheme transformations fits the identified requirements best. First, by-example transformers are easy to understand, as their behaviour is graphically described in terms of the particular modelling language used for design. Second, the by-example approach for defining transformer allows users to easily define scheme transformations, as the modelling language used for defining schemes is also used for defining transformers and only a few textual instructions, called directives, are additionally needed. Third, transformers may be flexibly used, as their behaviour can be easily modified at application time. Last, the expressive power of by-example transformers is adequate, because all scheme elements, a particular modelling language defines, can be affected by a scheme transformation.

7.2 The graphical editor *TransEd*

This chapter describes TransEd, which is a graphical editor for specifying WebML schemes and furthermore definitions and applications of transformers for WebML schemes. WebML schemes edited within TransEd respectively WebRatio, which is the

CASE tool support for WebML, are interchangeable, since both editors enable the specification of WebML schemes on the basis of the normative WebML DTD.

The combination of our TBE-engine and TransEd builds the prototype implementation of the TBE-system. Thereby, transformers that are graphically specified within TransEd are compiled into transformer definitions in terms of TBE by the TBE-engine. Further, transformer applications, which are graphically specified within TransEd, are performed by the TBE-engine. The outcome of such transformer applications, i.e. a WebML scheme, is again visualized by TransEd.

TransEd offers one mode for specifying WebML schemes and applications of transformers to such schemes, which is called *scheme development mode*. Another mode, which is called *transformer definition mode*, offered by TransEd provides for defining transformers. SECTION 7.2.1 illustrates the transformer definition mode of TransEd. SECTION 7.2.2 illustrates the scheme development mode of TransEd.

7.2.1 Transformer definition mode

The transformer definition mode of TransEd provides for defining query templates and generative templates in an overall scheme view. Therefore, it is possible to define templates that require scheme elements of different sub-schemes, like, for example, the content scheme or the hypertext scheme, in one view.

Example 7.3: *FIGURE 7.3 depicts the definition of transformer DemoTrans. In the upper part of FIGURE 7.3 the definition of the query template is shown by means of an overall scheme view. Therefore, it is possible to define the page classes and the entity types of the query template in one view. The definition of the generative template is shown in the lower part of FIGURE 7.3. The transformer itself is only for illustration purpose and therefore not explained.*

TransEd supports the definition of variables by providing distinctive forms for each type of scheme elements. For example, the right part of FIGURE 7.4 illustrates forms for entering the properties of scheme elements of type "page class". Please ignore section "Visibility" for the moment. Thereby, it is provided for entering variable names, specifying the kind of the variable, and attaching construction expressions, if desired. In contrast, the left part of FIGURE 7.4 shows the form for editing properties of page classes in the scheme development mode, where obviously the fields required for specifying variables are not shown. Besides the "Visibility" section, this form corresponds to a property from as usually provided by WebRatio.

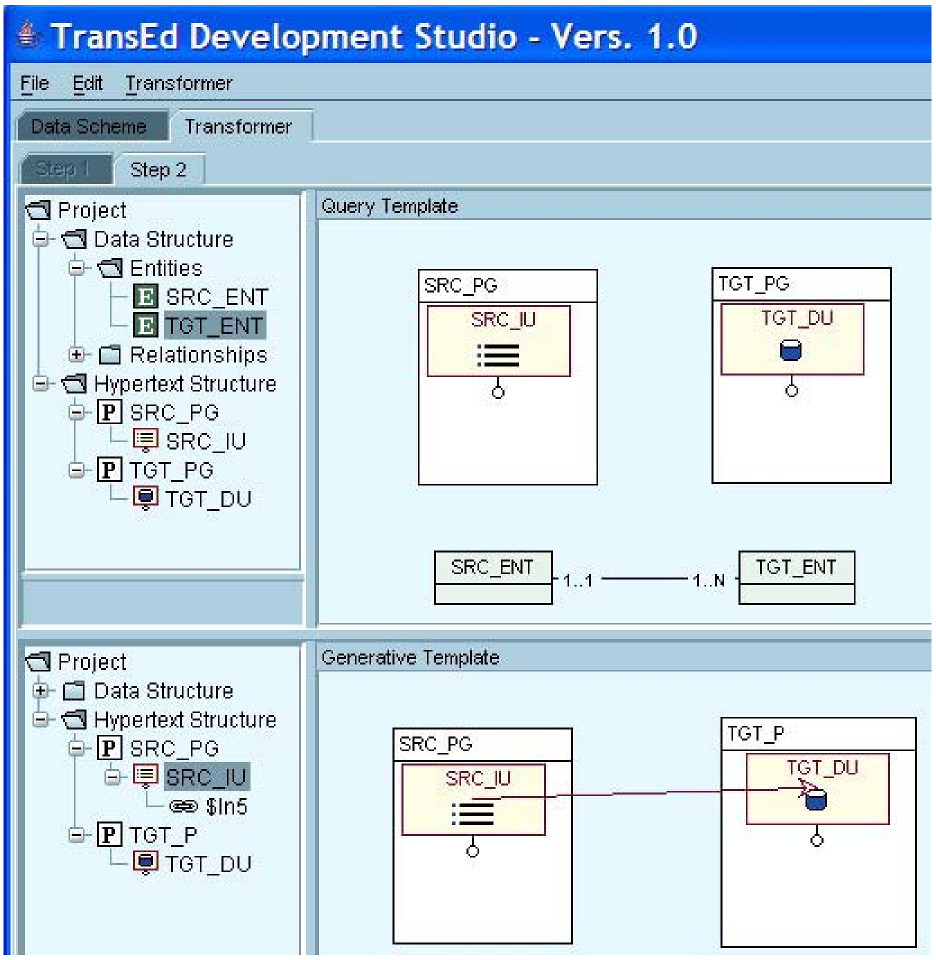
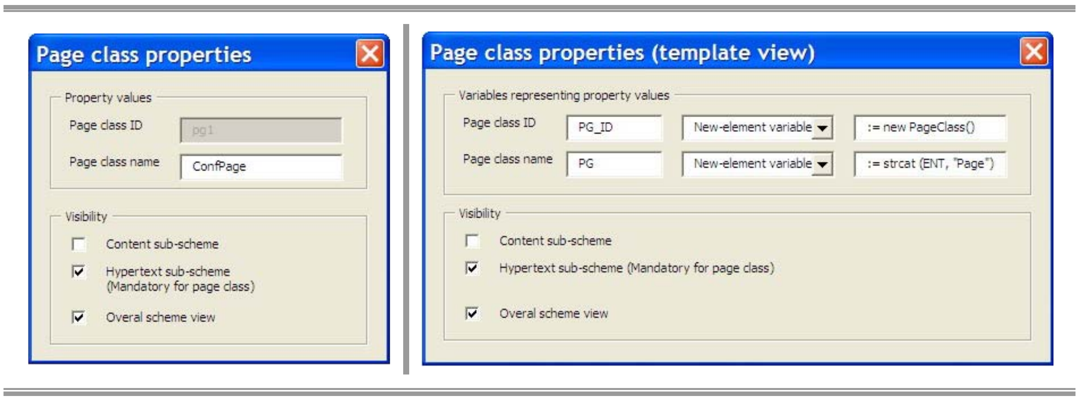


Figure 7.3: Overall scheme view of a transformer definition in TransEd

Furthermore, TransEd is aware of the syntax and semantics of transformers and can thereby assist modellers in defining transformers. Besides alerting errors, the system highlights interdependencies between different parts of a transformer. For example, if the mouse pointer is over a parameter variable of the generative template, the corresponding result variable of the query template is highlighted.



Page class properties

Property values

Page class ID	pg1
Page class name	ConfPage

Visibility

☐ Content sub-scheme
☑ Hypertext sub-scheme (Mandatory for page class)
☑ Overal scheme view

Page class properties (template view)

Variables representing property values

Page class ID	PG_ID	New-element variable	:= new PageClass()
Page class name	PG	New-element variable	:= strcat (ENT, "Page")

Visibility

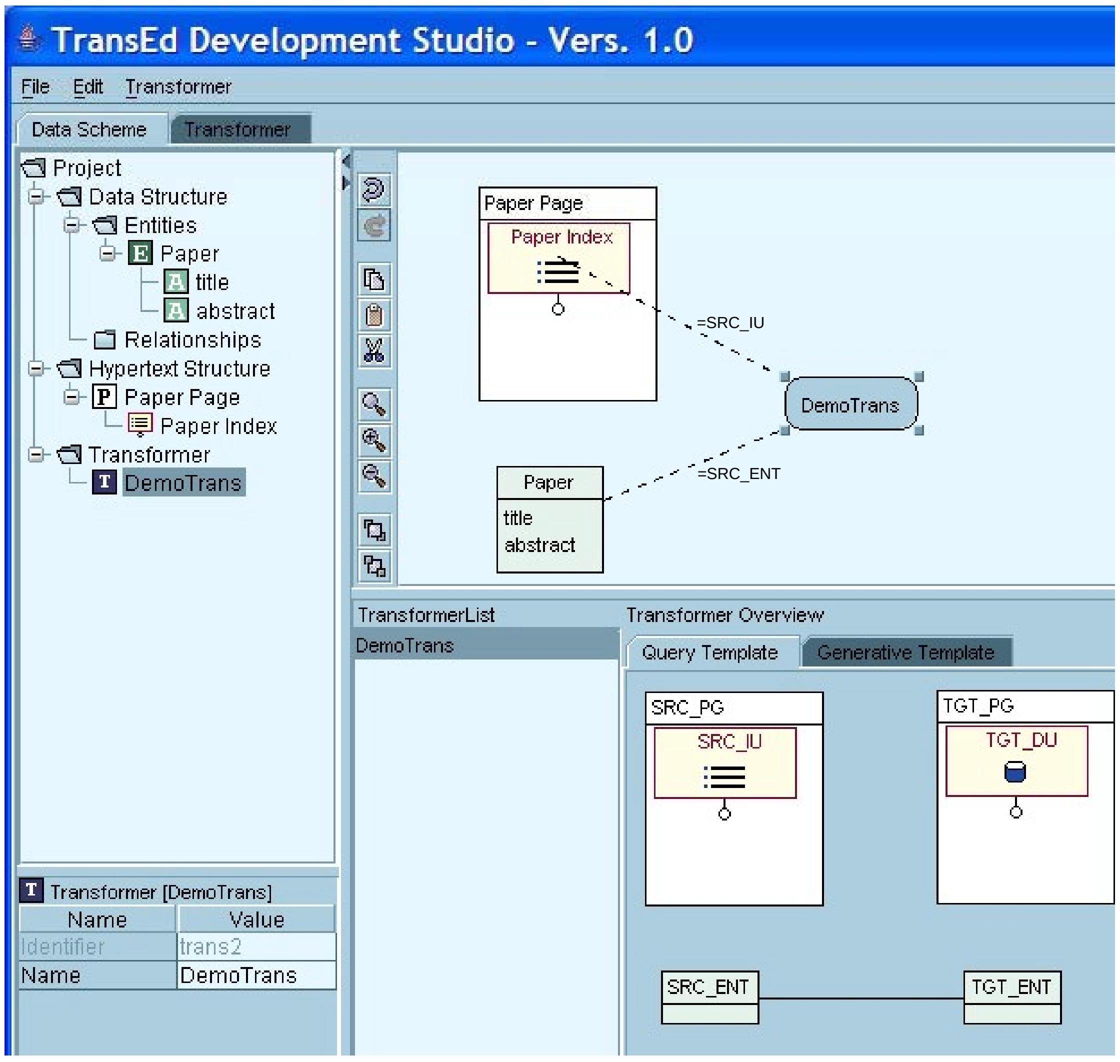
☐ Content sub-scheme
☑ Hypertext sub-scheme (Mandatory for page class)
☑ Overal scheme view

Figure 7.4: Forms for editing properties of page classes in the scheme development mode (left) and the transformer definition mode (right).

7.2.2 Scheme development mode

The scheme development mode of TransEd provides for specifying WebML schemes as well as it provides for specifying graphical applications of transformers. In order to facilitate graphical applications of transformers whose templates mix scheme elements, i.e. variables, of different sub-schemes, TransEd also provides an overall scheme view in the scheme development mode. In order to express in which view a particular scheme element shall be presented, one can individually specify the visibility in the scheme element's property form, which is illustrated in FIGURE 7.4.

For selecting a transformer to be applied TransEd visualizes all transformers currently stored into its transformer repository as a list. When a transformer is selected, it is visualized at the overall scheme view. For individualizing the transformer application TransEd provides for connecting scheme elements to the graphical representation of the transformer via drawing dashed lines. Although, the scheme elements, which are addressed by individualizing a transformer can be specified by drawing such lines it is not possible to determine the required variables of the transformer, since they are not visualized within the graphical representation of the transformer. Therefore, the names of variables have to be explicitly added to such connection lines.



Transformer [DemoTrans]

Name	Value
Identifier	trans2
Name	DemoTrans

Figure 7.5: Graphical application of a transformer DemoTrans in TransEd.

Example 7.4: *FIGURE 7.5 illustrates the graphical application of transformer DemoTrans, which is represented by a rounded rectangle including the transformer's name. The transfomer's definition is not repeated in this graphical shape but is shown in a separate window below. Application-specific constraints can be expressed by lines connecting scheme elements to the graphical application, and the transformer variable to be addressed is notated at the line. For the graphical application depicted in FIGURE 7.5, entity type Paper as well as index unit Paper Index are to be attached to the graphical application of transformer DemoTrans. For that purpose, both scheme elements are shown in the overall scheme view.*

8 Conclusion

We have presented a prototype implementation of the basic concepts of Transformers By-Example (TBE), which is a language for defining and applying scheme transformers. A TBE-system comprises two building blocks: The first building block is the *TBE-engine* that provides for compiling native transformer definitions into transformer definitions in terms of TBE *and* for performing transformer applications. The second building block is the *graphical editor* that facilitates the definition of schemes and templates.

We have achieved that the implementation of the graphical editor is exchangeable without requiring adaptations of the TBE-engine and vice versa since their interface is the internal representation of schemes of the respective modelling language, which is typically normative. For example, the internal representation of WebML schemes is an XML representation that adheres to the WebML DTD.

The implemented TBE-engine comprises model-dependent and model-independent components, where the latter make up the *TBE-framework*. The development of the TBE-framework has been focused within this diploma thesis. The independency of the TBE-framework from a particular modelling language `L` has been fundamentally achieved by processing the logical representation of schemes instead of processing the internal representation of schemes of modelling language `L` directly.

The major components of the TBE-framework are the `Generator` and the `Applicator`, which provide for generating transformer definitions in terms of TBE and provide for performing the application of such transformers definitions to input schemes in logical representation, respectively. We have demonstrated the implementation of an applicator that uses an XQuery statement, expressing the respective transformer definition in terms of TBE, for transforming an input scheme in logical representation. Because of the distinctive design of interfaces, representing the components of the TBE-framework, we have achieved that other applicators can be plugged in the TBE-framework easily as well.

We have demonstrated the application of the TBE-framework to modelling language WebML, i.e. we have implemented the few model-dependent components of the TBE-engine as required by WebML.

Further, we have demonstrated how WebRatio, which is the CASE-tool for WebML, can be used off-the-shelf as graphical editor for defining and applying transformers. In order to specify TBE-directives within WebRatio modellers annotate such directives in textual form. We have developed a syntax specifying this textual representation of TBE-directives.

List of Figures

Figure 1.1: WebML scheme of the CMA: content scheme (left) and hypertext scheme (right). 15
Figure 1.2: By-example definition of transformer `IndexPCForET` *in notation of WebML.* 17
Figure 1.3: Building blocks of a TBE-system - Graphical editor and TBE-engine. 20
Figure 1.4: Components of a model-specific TBE-engine. 25
Figure 1.5: Components of a generic TBE-engine. 26
Figure 1.6: Framework for TBE-engines and the WebML TBE-engine. 28
Figure 2.1: Phases in the WebML development process. 32
Figure 2.2: Extended content scheme of the CMA. 35
Figure 2.3: Extended hypertext scheme of the CMA. 36
Figure 2.4: Sub DTDs comprised within the WebML DTD. 38
Figure 2.5: Structure DTD fragment declaring scheme elements of sort entity type. 38
Figure 2.6: XML representation of entity type Author. 39
Figure 2.7: Screnn shot of the user interface of WebRatio. 41
Figure 3.1: Precise graphical notation of transformer `IndexPCForET´s` *query template.* 48
Figure 3.2: Transformer `IndexPCForET´s` *query template in terms of DRC.* 50
Figure 3.3: Precise graphical notation of transformer `IndexPCForET's` *generative template.* 54
Figure 3.4: Sample input for transformer `IndexPCForET´s` *generative template.* 57
Figure 3.5: Sample output of transformer `IndexPCForET´s` *generative template.* 57
Figure 3.6: Transformer `IndexPCForET.` 58
Figure 4.1: General syntax of directives in textual representation. 63
Figure 4.2: Annotating directives to WebML schemes. 63
Figure 4.3: Syntax of TBE-directives result variable and parameter variable in textual representation. 64
Figure 4.4: Graphical representation (left) and textual representation (right) of TBE-directive `tag result variable`*.* 64
Figure 4.5: Syntax of TBE-directive `constraint` *in textual representation.* 65
Figure 4.6: Syntax of TBE-directive `construction expression` *in textual representation.* 66
Figure 4.7: Syntax of customized-directive `alias`*.* 67
Figure 4.8: Specifying customized-directive `alias`*.* 67
Figure 4.9: Syntax of customized-directive `anchor`*.* 69
Figure 4.10: WorkArea of WebRatio when a new project is opened. 70
Figure 4.11: A newly added entity type (right) and its properties (left). 70
Figure 4.12: Result variable `ENT` *(right) and its properties (left).* 71
Figure 4.13: Definition of directive `alias: Identifier - $ENT_ID;`*.* 71
Figure 4.14: Definition of directive `alias: Identifier - ATT_ID;`*.* 72
Figure 4.15 New-element variable `SV_ID` *and corresponding directives.* 72
Figure 4.16: New-element variables `PC` *and* `PC_ID` *and corresponding directives.* 73
Figure 4.17 New-element variables `IU` *and* `IU_ID` *and corresponding directives.* 73

Figure 4.18: Customized-directive `anchor`*.* 74
Figure 4.19: Content scheme of the CMA web application. 75
Figure 4.20: Individualizing the application of transformer `IndexPCForET`*.* 76
Figure 4.21: Result of the individualized application of transformer `IndexPCForET` *to the* `CMA` *scheme.* 77
Figure 5.1: Base architecture of the TBE-engine. 80
Figure 5.2: Processes for compiling transformer definitions and their inputs and outputs. 82
Figure 5.3: Processes and their inputs and outputs required for applying transformers. 85
Figure 5.4: Datastructures `LogicalRepresentation` *and* `LogicalRepresentationSpecification`*.* 88
Figure 5.5: Overview of datastructure `TransformerDefinition` *(top) and required OCL expressions (bottom).* 91
Figure 5.6: Type hierarchy of constraints. 92
Figure 5.7: Structure of complex constraints. 92
Figure 5.8: Structure of membership constraints. 93
Figure 5.9: Structure of comparison constraints. 94
Figure 5.10: Structure of relation constructors. 95
Figure 5.11: Structure of new-element variables. 96
Figure 5.12 Structure of function construction expressions. 96
Figure 6.1: Technologies for implementing components for compiling transformer definitions. 102
Figure 6.2: Technologies for implementing components for applying transformers. 106
Figure 6.3: Allocation of process-implementations. 108
Figure 6.4: Interface `Mapper`*.* 110
Figure 6.5: Interface `Applicator`*.* 111
Figure 6.6: Abstract class `XQueryApplicator`*.* 111
Figure 6.7: Interface `Generator`*.* 112
Figure 6.8: Abstract class `Generator`*.* 113
Figure 6.9: Allocation of datastructure-implementations. 114
Figure 6.10: Syntax of identifiers. 116
Figure 6.11: Refined datastructure `LogicalRepresentationSpecification`*.* 116
Figure 6.12: Illustration of class `LogicalRepresentationSpecification`*.* 117
Figure 6.13: XML-schema specifying datastructure `LogicalRepresentationSpecification`*.* 118
Figure 6.14: Fragment of the specification of the logical representation of WebML schemes in terms of XML. 122
Figure 6.15: XML-schema specifying datastructure `Logical Representation`*.* 123
Figure 6.16: Illustration of class `LogicalRepresentation`*.* 123
Figure 6.17: Illustration of class `TransformerDefinition`*.* 124
Figure 6.18: Illustration of class `TBEDirectivesContainer`*.* 125
Figure 6.19: Illustration of the compressed logical representations of schemes. 126
Figure 6.20: Reconstructing universe members. 127
Figure 6.21: Variables ENT, ENT_ID and ATT_ID in notation of WebML (top) and resolved aliases (bottom). 128
Figure 6.22: Fragment of the query template of transformer IndexPCForET in notation of WebML (top) and its logical representation (bottom). 129

Figure 6.23: TBE-directives in notation of WebML (top) and their textual representation (bottom) 129
Figure 6.24: Customized-directive anchor in notation of WebML (top) and its textual representation (bottom). 130
Figure 6.25: New-element variables and construction expressions for generating coordinates of new scheme elements. 132
Figure 6.26: Preparing the generation of variables `ANCH_X` and `ANCH_Y`. 133
Figure 6.27: Preparation the generation of new-element variables for coordinates. 133
Figure 6.28: Generating variables. 135
Figure 6.29: Generating membership constraints. 135
Figure 6.30: Generating relation constructors. 136
Figure 6.31: Generating new-element variables. 136
Figure 6.32: Individualized definition of transformer `IndexPCForET`. 137
Figure 6.33: Partial XQuery statement representing transformer `IndexPCForET`. 139
Figure 6.34: Partial signature of relation `Entity`. 140
Figure 6.35: XML representation of result variable `ENT_ID` and `ENT`. 141
Figure 6.36: XML representation of a membership constraint of transformer IndexPCForET. 141
Figure 6.37: XML representation of a complex constraint of individualized transformer `IndexPCForET`. 141
Figure 6.38: XML representation of parameter variables `ENT_ID` and `ENT`. 142
Figure 6.39: XML representation of new-element variables `PC_ID` and `PC`. 143
Figure 6.40: XML representation of a relation constructor of transformer `IndexPCForET's` generative template. 143
Figure 6.41: Fragments of the output of process `apply transformer`. 144
Figure 6.42: Fragment of the output scheme in notation of WebML (bottom) and its logical representation (top). 145
Figure 7.1: Transformation rule for the `Landmark` pattern. *149*
Figure 7.2: A possible transformation rule instantiation for the "Landmark" pattern. *150*
Figure 7.3: Overall scheme view of a transformer definition in TransEd 154
Figure 7.4: Forms for editing properties of page classes in the scheme development mode (left) and the transformer definition mode (right). 155
Figure 7.5: Graphical application of a transformer DemoTrans in TransEd. 156

Bibliography

[antlr04] AntLR Parser Generator. http://www.antlr.org/, [September 2004].

[BMRSS96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. A System of Patterns, volume 1 of Pattern-Oriented Software Architecture. Wiley & Sons, New York, hardcover edition, 1996.

[Boe85] B. Boehm. A spiral model of software development and enhancement. In J. Wileden and M. Dowson, editors, Proceedings of the 2nd International Software Process Workshop, pages 22-42, March 1985.

[CCP01] Jaime Comez, Christina Cachero, and Oscar Pastor. Conceptual modeling of device-independent web applications. IEEE Web Engineering, 8(2):26-39, 2001.

[CFB00] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web modeling language (webml): a modeling language for designing web sites. In Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking, pages 137-157. North-Holland Publishing Co., 2000.

[Che76] Peter Pin-Shan Chen. The entity-relationship modeltoward a unified view of data. ACM Trans. Database Syst., 1(1):9-36, 1976.

[Con99] Jim Conallen. Modeling web application architectures with uml. Commun. ACM, 42(10):63-70, 1999.

[Dro97] Offer Drori. Hypertext implications for case environments. SIGSOFT Softw. Eng. Notes, 22(4):35-38, 1997.

[Fra99] Piero Fraternali. Tools and approaches for developing data-intensive web applications: a survey. ACM Comput. Surv., 31(3):227-263, 1999.

[Lec04] Stephan Lechner. Transformers-by-example. Dissertation, Johannes Kepler University Linz, Departement for Data & Knowledge Engineering, 2004.

[LS03] S. Lechner, M. Schrefl. Defining web schema transformers by example. In Proceedings of DEXA'03. Springer, 2003.

[LS04] Stephan Lechner and Michael Schrefl. Trasformers-by-example: pushing reuse in conceptual web application modelling. In Proceedings of the 2004 ACM symposium on Applied computing, pages 1654-1661. ACM Press, 2004.

[MAM03] Paolo Merialdo, Paolo Atzeni, and Giansalvatore Mecca. Design and development of data-intensive web sites: The araneus approach. ACM Trans. Inter. Tech., 3(1):49-92, 2003.

[MMCF03] M. Matera, A. Maurino, S. Ceri, and P. Fraternali. Model-driven design of collaborative web applications. Softw. Pract. Exper., 33(8):701-732, 2003.

[oopsla04] Conference on Object Oriented Programming Systems Languages and Applications. http://www.oopsla.org, [April 2004].

[oql04] The Object Query Language. http://www.odmg.org, [September 2004].

[RG00] R. Ramakrishnan, J. Gehrke. Database management systems. McGraw-Hill, 2000.

[RLS99] Gustavo Rossi, Fernando Daniel Lyardet, and Daniel Schwabe. Developing hypermedia applications with methods and patterns. ACM Comput. Surv., 31(4es):8, 1999.

[sax04] SAXON - The XSLT and XQuery Processor. http://saxon.sourceforge.net/, [September 2004].

[uml04] The Unified Modelling Language. http://www.omg.org, [September 2004].

[Wab04] A. Wabro. Editor für ein Template-basiertes Navigationsmodell. Diploma thesis, Johannes Kepler University Linz, Departement for Data & Knowledge Engineering, 2004.

[webml04] WebML Users Guide 3.0. http://www.webml.org, [April 2004].

[Wir77] Niklaus Wirth: What Can We Do about the Unnecessary Diversity of Notation for Syntactic Definitions? Commun. ACM 20 (11): 822-823 (1977)

[WK98] J. Warmer and A. Kleppe. The Object Constraint Language: Precise Modeling with UML. Addison-Wesley, paperback edition, 1998.

[xal04] Xalan-Java version 2.6.0. http://xml.apache.org/xalan-j/, [September 2004].

[xml04] The Extensible Markup Language. http://www.w3.org/xml, [September 2004].

WebML DTD			Logical Representation	
			Relation	Universe
<!ATTLIST ENTITY			Entity (	
id	ID	#REQUIRED	entity	Entity
name	CDATA	#IMPLIED	name	Name
superEntity	CDATA	#IMPLIED	superEntity	Entity
value	CDATA	#IMPLIED	value	Dummy
>			)	
<!ELEMENT ENTITY (ATTRIBUTE*, RELATIONSHIP*, PROPERTY*, COMMENT?)>				

WebML DTD			Logical Representation	
			Relation	Universe
<!ATTLIST ATTRIBUTE			Attribute (	
id	ID	#REQUIRED	attribute	Attribute
name	CDATA	#IMPLIED	name	Name
type	%WebMLTypes%	#IMPLIED	type	Dummy
contentType	CDATA	#IMPLIED	contentType	Dummy
userType	IDREF	#IMPLIED	userType	Dummy
value	CDATA	#IMPLIED	value	Dummy
			definedAt	Entity
>			)	
<!ELEMENT ATTRIBUTE (PROPERTY*, COMMENT?)>				

WebML DTD			Logical Representation	
			Relation	Universe
<!ATTLIST RELATIONSHIP			RelRole (	
id	ID	#REQUIRED	relRole	RelRole
name	CDATA	#IMPLIED	relShipName	Name
roleName	CDATA	#IMPLIED	name	Name
to	IDREF	#REQUIRED	to	Entity
inverse	IDREF	#REQUIRED	inverse	RelRole
minCard	CDATA	#REQUIRED	minCard	Cardinality
maxCard	CDATA	#REQUIRED	maxCard	Cardinality
value	CDATA	#IMPLIED	value	Dummy
			from	Entity
>			)	
<!ELEMENT RELATIONSHIP (PROPERTY*, COMMENT?)>				

WebML DTD			Logical Representation	
			Relation	Universe
<!ATTLIST graph:Node			EntityPos (	
id	ID	#REQUIRED	pos	EntityPos
x	NUMBER	#REQUIRED	xValue	XPosValue
y	NUMBER	#REQUIRED	yValue	YPosValue
element	IDREF	#REQUIRED	element	Entity
>			)	
<!ELEMENT graph:Node EMPTY>				

WebML DTD			**Logical Representation**	
			Relation	**Universe**
<!ATTLIST graph:Connection			RelRolePos (	
id	ID	#REQUIRED	pos	RelRolePos
x	NUMBER	#REQUIRED	xValue	XPosValue
y	NUMBER	#REQUIRED	yValue	YPosValue
element	IDREF	#REQUIRED	element	RelRole
>			)	
<!ELEMENT graph:Connection EMPTY>				

WebML DTD			**Logical Representation**	
			Relation	**Universe**
<!ATTLIST SITEVIEW			SiteView (	
id	ID	#REQUIRED	siteView	SiteView
name	CDATA	#IMPLIED	name	Name
protected	(yes|no)	'no'	protected	Dummy
homePage	IDREF	#IMPLIED	homePage	Dummy
>			)	
<!ELEMENT SITEVIEW (AREA*, PAGE*, OPERATIONUNITS, GLOBALPARAMETER*, COMMENT?)>				

WebML DTD			**Logical Representation**	
			Relation	**Universe**
<!ATTLIST PAGE			Page (	
id	ID	#REQUIRED	page	Page
name	CDATA	#IMPLIED	name	Name
landmark	(yes|no)	'no'	landmark	Dummy
			definedAt	SiteView
>			)	
<!ELEMENT PAGE (CONTENTUNITS, PAGE*, ALTERNATIVE*, LINK*, PROPERTY*)>				

WebML DTD			Logical Representation	
			Relation	Universe
<!ATTLIST LINK			Link (	
id	ID	#REQUIRED	link	Link
name	CDATA	#IMPLIED	name	Name
to	IDREFS	#REQUIRED	destPage	Page
			destDataUnit	DataUnit
			destIndexUnit	IndexUnit
			destScrollerUnit	ScrollerUnit
			destDeleteUnit	DeleteUnit
			destModifyUnit	ModifyUnit
			destCreateUnit	CreateUnit
type	(normal|automatic|transport)	'normal'	type	Dummy
newWindow	(yes|no)	'no'	newWindow	Dummy
			startDataUnit	DataUnit
			startIndexUnit	IndexUnit
			startScrollerUnit	ScrollerUnit
			startDeleteUnit	DeleteUnit
			startModifyUnit	ModifyUnit
			startCreateUnit	CreateUnit
			startPage	Page
>			)	
<!ELEMENT LINK (LINKPARAMETER*, PROPERTY*, COMMENT?)>				

WebML DTD			Logical Representation	
			Relation	Universe
<!ATTLIST DATAUNIT			DataUnit (	
id	ID	#REQUIRED	dataUnit	DataUnit
name	CDATA	#IMPLIED	name	Name
entity	IDREF	#IMPLIED	entity	Entity
			definedAt	Page
>			)	
<!ELEMENT DATAUNIT (SELECTOR?, (DISPLAYALL|DISPLAYATTRIBUTE*), LINK*, COMMENT?)>				

WebML DTD			Logical Representation	
			Relation	Universe
<!ATTLIST INDEXUNIT			IndexUnit (	
id	ID	#REQUIRED	indexUnit	IndexUnit
name	CDATA	#IMPLIED	name	Name
entity	IDREF	#IMPLIED	entity	Entity
distinct	(yes|no)	'no'	distinct	Dummy
			definedAt	Page
>			)	
<!ELEMENT INDEXUNIT (SELECTOR?, DISPLAYATTRIBUTE*, SORTATTRIBUTE*, LINK*, PROPERTY*)>				

WebML DTD			Logical Representation	
			Relation	Universe
<!ATTLIST ENTRYUNIT			EntryUnit (	
id	ID	#REQUIRED	entryUnit	EntryUnit
name	CDATA	#IMPLIED	name	Name
			definedAt	Page
>			)	
<!ELEMENT ENTRYUNIT (FIELD*, SELECTIONFIELD*, LINK*, PROPERTY*, COMMENT?)>				

WebML DTD			Logical Representation	
			Relation	Universe
<!ATTLIST SCROLLERUNIT			ScrollerUnit (	
id	ID	#REQUIRED	scrollerUnit	ScrollerUnit
name	CDATA	#IMPLIED	name	Name
entity	IDREF	#IMPLIED	entity	Entity
blockFactor	CDATA	#IMPLIED	blockFactor	Dummy
			definedAt	Page
>			)	
<!ELEMENT SCROLLERUNIT (SELECTOR?, SORTATTRIBUTE*, LINK, PROPERTY*, COMMENT?)>				

WebML DTD			Logical Representation	
			Relation	Universe
<!ATTLIST CREATEUNIT			CreateUnit (	
id	ID	#REQUIRED	createUnit	CreateUnit
name	CDATA	#IMPLIED	name	Name
entity	IDREF	#REQUIRED	entity	Entity
			definedAt	SiteView
>			)	
<!ELEMENT CREATEUNIT (LINK*, OK-LINK*, KO-LINK*, PROPERTY*, COMMENT?)>				

WebML DTD			Logical Representation	
			Relation	Universe
<!ATTLIST DELETEUNIT			DeleteUnit (	
id	ID	#REQUIRED	deleteUnit	DeleteUnit
name	CDATA	#IMPLIED	name	Name
entity	IDREF	#REQUIRED	entity	Entity
			definedAt	SiteView
>			)	
<!ELEMENT DELETEUNIT (SELECTOR?, LINK*, OK-LINK*, KO-LINK*, PROPERTY*, COMMENT?)>				

WebML DTD			Logical Representation	
			Relation	Universe
<!ATTLIST MODIFYUNIT			ModifyUnit (	
id	ID	#REQUIRED	modifyUnit	ModifyUnit
name	CDATA	#IMPLIED	name	Name
entity	IDREF	#REQUIRED	entity	Entity
			definedAt	SiteView
>			)	
<!ELEMENT MODIFYUNIT (LINK*, OK-LINK*, KO-LINK*, PROPERTY*, COMMENT?)>				

WebML DTD			Logical Representation	
			Relation	Universe
<!ATTLIST graph:Node			PagePos (	
id	ID	#REQUIRED	pos	PagePos
x	NUMBER	#REQUIRED	xValue	XPosValue
y	NUMBER	#REQUIRED	yValue	YPosValue
element	IDREF	#REQUIRED	element	Page
>			)	
<!ELEMENT graph:Node EMPTY>				

WebML DTD			Logical Representation	
			Relation	Universe
<!ATTLIST graph:Connection			LinkPos (	
id	ID	#REQUIRED	pos	LinkPos
x	NUMBER	#REQUIRED	xValue	XPosValue
y	NUMBER	#REQUIRED	yValue	YPosValue
element	IDREF	#REQUIRED	element	Link
>			)	
<!ELEMENT graph:Node EMPTY>				

WebML DTD			Logical Representation	
			Relation	Universe
<!ATTLIST graph:Node			IndexUnitPos (	
id	ID	#REQUIRED	pos	IndexUnitPos
x	NUMBER	#REQUIRED	xValue	XPosValue
y	NUMBER	#REQUIRED	yValue	YPosValue
element	IDREF	#REQUIRED	element	IndexUnit
>			)	
<!ELEMENT graph:Node EMPTY>				

WebML DTD			Logical Representation	
			Relation	Universe
<!ATTLIST graph:Node			DataUnitPos (	
id	ID	#REQUIRED	pos	DataUnitPos
x	NUMBER	#REQUIRED	xValue	XPosValue
y	NUMBER	#REQUIRED	yValue	YPosValue
element	IDREF	#REQUIRED	element	DataUnit
>			)	
<!ELEMENT graph:Node EMPTY>				

WebRatio Representation			Logical Representation	
			Relation	Universe
<!ATTLIST graph:Node			EntryUnitPos (	
id	ID	#REQUIRED	pos	EntryUnitPos
x	NUMBER	#REQUIRED	xValue	XPosValue
y	NUMBER	#REQUIRED	yValue	YPosValue
element	IDREF	#REQUIRED	element	EntryUnit
>			)	
<!ELEMENT graph:Node EMPTY>				

WebRatio Representation			Logical Representation	
			Relation	Universe
<!ATTLIST graph:Node			ScrollerUnitPos (	
id	ID	#REQUIRED	pos	ScrollerUnitPos
x	NUMBER	#REQUIRED	xValue	XPosValue
y	NUMBER	#REQUIRED	yValue	YPosValue
element	IDREF	#REQUIRED	element	ScrollerUnit
>			)	
<!ELEMENT graph:Node EMPTY>				

WebRatio Representation			Logical Representation	
			Relation	Universe
<!ATTLIST graph:Node			CreateUnitPos (	
id	ID	#REQUIRED	pos	CreateUnitPos
x	NUMBER	#REQUIRED	xValue	XPosValue
y	NUMBER	#REQUIRED	yValue	YPosValue
element	IDREF	#REQUIRED	element	CreateUnit
>			)	
<!ELEMENT graph:Node EMPTY>				

WebRatio Representation			Logical Representation	
			Relation	Universe
<!ATTLIST graph:Node			DeleteUnitPos (	
id	ID	#REQUIRED	pos	DeleteUnitPos
x	NUMBER	#REQUIRED	xValue	XPosValue
y	NUMBER	#REQUIRED	yValue	YPosValue
element	IDREF	#REQUIRED	element	DeleteUnit
>			)	
<!ELEMENT graph:Node EMPTY>				

WebRatio Representation			Logical Representation	
			Relation	Universe
<!ATTLIST graph:Node			ModifyUnitPos (	
id	ID	#REQUIRED	pos	ModifyUnitPos
x	NUMBER	#REQUIRED	xValue	XPosValue
y	NUMBER	#REQUIRED	yValue	YPosValue
element	IDREF	#REQUIRED	element	ModifyUnit
>			)	
<!ELEMENT graph:Node EMPTY>				

```xml
<?xml version="1.0" encoding="UTF-8"?>

<LogicalRepresentationSpecification
        xmlns="http://www.dke.jku.at/tbe/data/logrepspec"
        xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.dke.jku.at/tbe/data/logrepspec
                                LogicalRepresentationSpecification.xsd"
        name="WebML">

    <Universe name="Name"/>
    <Universe name="Cardinality"/>
    <Universe name="XPosValue"/>
    <Universe name="YPosValue"/>
    <Universe name="Entity" prefix="ent"/>
    <Universe name="Attribute" prefix="att"/>
    <Universe name="RelRole" prefix="rel"/>
    <Universe name="SiteView" prefix="sv"/>
    <Universe name="Page" prefix="page"/>
    <Universe name="Link" prefix="ln"/>
    <Universe name="DataUnit" prefix="dau"/>
    <Universe name="IndexUnit" prefix="inu"/>
    <Universe name="EntryUnit" prefix="flu"/>
    <Universe name="ScrollerUnit" prefix="scu"/>
    <Universe name="CreateUnit" prefix="cru"/>
    <Universe name="DeleteUnit" prefix="dlu"/>
    <Universe name="ModifyUnit" prefix="mfu"/>
    <Universe name="EntityPos" prefix="ent" suffix="_go"/>
    <Universe name="RelRolePos" prefix="rel" suffix="_go"/>
    <Universe name="PagePos" prefix="page" suffix="_go"/>
    <Universe name="LinkPos" prefix="ln" suffix="_go"/>
    <Universe name="DataUnitPos" prefix="dau" suffix="_go"/>
    <Universe name="IndexUnitPos" prefix="inu" suffix="_go"/>
    <Universe name="EntryUnitPos" prefix="flu" suffix="_go"/>
    <Universe name="ScrollerUnitPos" prefix="scu" suffix="_go"/>
    <Universe name="CreateUnitPos" prefix="cru" suffix="_go"/>
    <Universe name="DeleteUnitPos" prefix="dlu" suffix="_go"/>
    <Universe name="ModifyUnitPos" prefix="mfu" suffix="_go"/>

    <Relation name="Entity">
        <Attribute name="entity" universe="Entity"/>
        <Attribute name="name" universe="Name"/>
    </Relation>
    <Relation name="Attribute">
        <Attribute name="attribute" universe="Attribute"/>
        <Attribute name="name" universe="Name"/>
        <Attribute name="definedAt" universe="Entity"/>
    </Relation>
    <Relation name="RelRole">
        <Attribute name="relRole" universe="RelRole"/>
        <Attribute name="relShipName" universe="Name"/>
        <Attribute name="name" universe="Name"/>
        <Attribute name="to" universe="Entity"/>
        <Attribute name="inverse" universe="RelRole"/>
        <Attribute name="minCard" universe="Cardinality"/>
        <Attribute name="maxCard" universe="Cardinality"/>
        <Attribute name="from" universe="Entity"/>
    </Relation>
    <Relation name="EntityPos">
```

```
        <Attribute name="pos" universe="EntityPos"/>
        <Attribute name="element" universe="Entity"/>
        <Attribute name="xValue" universe="XPosValue"/>
        <Attribute name="yValue" universe="YPosValue"/>
    </Relation>
    <Relation name="RelRolePos">
        <Attribute name="pos" universe="RelRolePos"/>
        <Attribute name="element" universe="RelRole"/>
        <Attribute name="xValue" universe="XPosValue"/>
        <Attribute name="yValue" universe="YPosValue"/>
    </Relation>
    <Relation name="SiteView">
        <Attribute name="siteView" universe="SiteView"/>
        <Attribute name="name" universe="Name"/>
    </Relation>
    <Relation name="Page">
        <Attribute name="page" universe="Page"/>
        <Attribute name="name" universe="Name"/>
        <Attribute name="definedAt" universe="SiteView"/>
    </Relation>
    <Relation name="Link">
        <Attribute name="link" universe="Link"/>
        <Attribute name="name" universe="Name"/>
        <Attribute name="destPage" universe="Page"/>
        <Attribute name="destDataUnit" universe="DataUnit"/>
        <Attribute name="destIndexUnit" universe="IndexUnit"/>
        <Attribute name="destScrollerUnit" universe="ScrollerUnit"/>
        <Attribute name="destDeleteUnit" universe="DeleteUnit"/>
        <Attribute name="destModifyUnit" universe="ModifyUnit"/>
        <Attribute name="destCreatUnit" universe="CreateUnit"/>
        <Attribute name="startPage" universe="Page"/>
        <Attribute name="startDataUnit" universe="DataUnit"/>
        <Attribute name="startIndexUnit" universe="IndexUnit"/>
        <Attribute name="startScrollerUnit" universe="ScrollerUnit"/>
        <Attribute name="startDeleteUnit" universe="DeleteUnit"/>
        <Attribute name="startModifyUnit" universe="ModifyUnit"/>
        <Attribute name="startCreatUnit" universe="CreateUnit"/>
    </Relation>
    <Relation name="DataUnit">
        <Attribute name="dataUnit" universe="DataUnit"/>
        <Attribute name="name" universe="Name"/>
        <Attribute name="entity" universe="Entity"/>
        <Attribute name="definedAt" universe="Page"/>
    </Relation>
    <Relation name="IndexUnit">
        <Attribute name="indexUnit" universe="IndexUnit"/>
        <Attribute name="name" universe="Name"/>
        <Attribute name="entity" universe="Entity"/>
        <Attribute name="definedAt" universe="Page"/>
    </Relation>
    <Relation name="EntryUnit">
        <Attribute name="entryUnit" universe="EntryUnit"/>
        <Attribute name="name" universe="Name"/>
        <Attribute name="definedAt" universe="Page"/>
    </Relation>
    <Relation name="ScrollerUnit">
        <Attribute name="scrollerUnit" universe="ScrollerUnit"/>
        <Attribute name="name" universe="Name"/>
    <Attribute name="entity" universe="Entity"/>
    <Attribute name="definedAt" universe="Page"/>
</Relation>
<Relation name="CreateUnit">
    <Attribute name="createUnit" universe="CreateUnit"/>
    <Attribute name="name" universe="Name"/>
    <Attribute name="entity" universe="Entity"/>
    <Attribute name="definedAt" universe="SiteView"/>
</Relation>
<Relation name="DeleteUnit">
    <Attribute name="deleteUnit" universe="DeleteUnit"/>
    <Attribute name="name" universe="Name"/>
    <Attribute name="entity" universe="Entity"/>
    <Attribute name="definedAt" universe="SiteView"/>
</Relation>
<Relation name="ModifyUnit">
    <Attribute name="modifyUnit" universe="ModifyUnit"/>
    <Attribute name="name" universe="Name"/>
    <Attribute name="entity" universe="Entity"/>
    <Attribute name="definedAt" universe="SiteView"/>
</Relation>
<Relation name="PagePos">
    <Attribute name="pos" universe="PagePos"/>
    <Attribute name="element" universe="Page"/>
    <Attribute name="xValue" universe="XPosValue"/>
    <Attribute name="yValue" universe="YPosValue"/>
</Relation>
<Relation name="DataUnitPos">
    <Attribute name="pos" universe="DataUnitPos"/>
    <Attribute name="element" universe="DataUnit"/>
    <Attribute name="xValue" universe="XPosValue"/>
    <Attribute name="yValue" universe="YPosValue"/>
</Relation>
<Relation name="IndexUnitPos">
    <Attribute name="pos" universe="IndexUnitPos"/>
    <Attribute name="element" universe="IndexUnit"/>
    <Attribute name="xValue" universe="XPosValue"/>
    <Attribute name="yValue" universe="YPosValue"/>
</Relation>
<Relation name="EntryUnitPos">
    <Attribute name="pos" universe="EntryUnitPos"/>
    <Attribute name="element" universe="EntryUnit"/>
    <Attribute name="xValue" universe="XPosValue"/>
    <Attribute name="yValue" universe="YPosValue"/>
</Relation>
<Relation name="ScrollerUnitPos">
    <Attribute name="pos" universe="ScrollerUnitPos"/>
    <Attribute name="element" universe="ScrollerUnit"/>
    <Attribute name="xValue" universe="XPosValue"/>
    <Attribute name="yValue" universe="YPosValue"/>
</Relation>
<Relation name="CreateUnitPos">
    <Attribute name="pos" universe="CreateUnitPos"/>
    <Attribute name="element" universe="CreateUnit"/>
    <Attribute name="xValue" universe="XPosValue"/>
    <Attribute name="yValue" universe="YPosValue"/>
</Relation>
```

```
    <Relation name="DeleteUnitPos">
        <Attribute name="pos" universe="DeleteUnitPos"/>
        <Attribute name="element" universe="DeleteUnit"/>
        <Attribute name="xValue" universe="XPosValue"/>
        <Attribute name="yValue" universe="YPosValue"/>
    </Relation>
    <Relation name="ModifyUnitPos">
        <Attribute name="pos" universe="ModifyUnitPos"/>
        <Attribute name="element" universe="ModifyUnit"/>
        <Attribute name="xValue" universe="XPosValue"/>
        <Attribute name="yValue" universe="YPosValue"/>
    </Relation>
    <Relation name="LinkPos">
        <Attribute name="pos" universe="LinkPos"/>
        <Attribute name="element" universe="Link"/>
        <Attribute name="xValue" universe="XPosValue"/>
        <Attribute name="yValue" universe="YPosValue"/>
    </Relation>
</LogicalRepresentationSpecification>
```

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE WebML SYSTEM "WebML.dtd">

<WebML xmlns:auxiliary="http://www.webml.org/auxiliary"
    xmlns:graphmetadata="http://www.webml.org/graphmetadata"
    xmlns:presentation="http://www.webml.org/presentation" auxiliary:compileJavaFiles="yes"
    auxiliary:deploy-with-names="no" auxiliary:http-port="8080" auxiliary:https-port="8443"
    auxiliary:layoutUseUnderscore="no" auxiliary:scramble-url="no" auxiliary:secure-url="no"
    auxiliary:structured-deploy="no" siteName="Untitled" version="3.0.14">
  <Structure graphmetadata:go="Structure_go" id="Structure">
    <ENTITY auxiliary:attributesVisible="true" auxiliary:testCaseCount="20"
              graphmetadata:go="User_go" id="User" name="User">
      <ATTRIBUTE id="userOID" name="OID" type="OID"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="userName" name="UserName"
                    type="String"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="password" name="Password"
                    type="Password"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="email" name="EMail" type="String"/>
      <RELATIONSHIP auxiliary:testCaseCount="20" graphmetadata:go="User2Group_go"
            id="User2Group" inverse="Group2User" maxCard="N" minCard="1"
            name="User_Group" roleName="User2Group" to="Group"/>
      <RELATIONSHIP auxiliary:testCaseCount="20"
            graphmetadata:go="User2DefaultGroup_go" id="User2DefaultGroup"
            inverse="DefaultGroup2User" maxCard="1" minCard="1" name="User_DefaultGroup"
            roleName="User2DefaultGroup" to="Group"/>
    </ENTITY>
    <ENTITY auxiliary:attributesVisible="true" auxiliary:testCaseCount="20"
            graphmetadata:go="Group_go" id="Group" name="Group">
      <ATTRIBUTE id="groupOID" name="OID" type="OID"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="groupName" name="GroupName"
              type="String"/>
      <RELATIONSHIP auxiliary:testCaseCount="20" graphmetadata:go="" id="Group2User"
            inverse="User2Group" maxCard="N" minCard="1" name="User_Group"
            roleName="Group2User" to="User"/>
      <RELATIONSHIP auxiliary:testCaseCount="20" graphmetadata:go=""
            id="DefaultGroup2User" inverse="User2DefaultGroup" maxCard="N" minCard="0"
            name="User_DefaultGroup" roleName="DefaultGroup2User" to="User"/>
      <RELATIONSHIP auxiliary:testCaseCount="20" graphmetadata:go="Group2SiteView_go"
              id="Group2SiteView" inverse="SiteView2Group" maxCard="1" minCard="1"
              name="Group_SiteView" roleName="Group2SiteView" to="SiteView"/>
    </ENTITY>
    <ENTITY auxiliary:attributesVisible="true" auxiliary:testCaseCount="20"
            graphmetadata:go="SiteView_go" id="SiteView" name="SiteView">
      <ATTRIBUTE id="siteViewOID" name="OID" type="OID"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="siteViewID" name="SiteViewID"
              type="String"/>
      <RELATIONSHIP auxiliary:testCaseCount="20" graphmetadata:go="" id="SiteView2Group"
              inverse="Group2SiteView" maxCard="N" minCard="1" name="Group_SiteView"
              roleName="SiteView2Group" to="Group"/>
    </ENTITY>
    <ENTITY auxiliary:attributesVisible="true" graphmetadata:go="ent1_go" id="ent1"
            name="$ENT">
      <ATTRIBUTE id="att1" name="OID" type="OID"/>
      <ATTRIBUTE id="att2" name="" type="String">
        <PROPERTY id="prop2" name="Identifier - ATT_ID;" value="alias:"/>
      </ATTRIBUTE>
      <PROPERTY id="prop1" name="Identifier - $ENT_ID;" value="alias:"/>
    </ENTITY>
```

```
</Structure>
<MetaStructure graphmetadata:go="MetaStructure_go" id="MetaStructure">
   <DOMAIN id="meta$LogPriority" name="Log Priority">
      <DOMAINVALUE value="ERROR"/>
      <DOMAINVALUE value="WARN"/>
      <DOMAINVALUE value="INFO"/>
      <DOMAINVALUE value="DEBUG"/>
   </DOMAIN>
   <DOMAIN id="meta$RTServiceType" name="RTService Type">
      <DOMAINVALUE value="READ"/>
      <DOMAINVALUE value="WRITE"/>
      <DOMAINVALUE value="LINK"/>
      <DOMAINVALUE value="PERMISSION"/>
      <DOMAINVALUE value="OTHER"/>
   </DOMAIN>
   <ENTITY auxiliary:attributesVisible="false" auxiliary:testCaseCount="20"
               graphmetadata:go="meta$LogEvent_go" id="meta$LogEvent" name="LogEvent">
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="meta$LogEvent$index" name="index"
               type="Integer"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="meta$LogEvent$timestamp"
               name="timestamp" type="String"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="meta$LogEvent$priority"
               name="priority" userType="meta$LogPriority"/>
      <ATTRIBUTE id="meta$LogEvent$rtServiceID" name="rtServiceID" type="String"
               value="Self.meta$LogEvent$RelatedTo.meta$RTService$identifier"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="meta$LogEvent$message"
               name="message" type="String"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="meta$LogEvent$throwable"
               name="throwable" type="Text"/>
      <RELATIONSHIP auxiliary:testCaseCount="20"
               graphmetadata:go="meta$LogEvent$RelatedTo_go"
               id="meta$LogEvent$RelatedTo" inverse="meta$RTService$LoggedEvents"
               maxCard="1" minCard="1" name="LogEvent_RTService" roleName="RelatedTo"
               to="meta$RTService"/>
   </ENTITY>
   <ENTITY auxiliary:attributesVisible="false" auxiliary:testCaseCount="20"
            graphmetadata:go="meta$RTService_go" id="meta$RTService" name="RTService">
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="meta$RTService$identifier"
               name="identifier" type="String"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="meta$RTService$type" name="type"
               userType="meta$RTServiceType"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="meta$RTService$hitCount"
               name="hitCount" type="Integer"/>
      <RELATIONSHIP auxiliary:testCaseCount="20" graphmetadata:go=""
               id="meta$RTService$LoggedEvents" inverse="meta$LogEvent$RelatedTo"
               maxCard="N" minCard="0" name="LogEvent_RTService"
               roleName="LoggedEvents" to="meta$LogEvent"/>
   </ENTITY>
</MetaStructure>
<Navigation>
   <GLOBALPARAMETER duration="session" entity="User" id="UserCtxParam"
               name="UserCtxParam"/>
   <GLOBALPARAMETER duration="session" entity="Group" id="GroupCtxParam"
               name="GroupCtxParam"/>
</Navigation>
<Mapping>
   <rdbms:RDBMSMapping xmlns:rdbms="http://www.webml.org/mapping/rdbms"/>
</Mapping>
```

```
    <auxiliary:GraphMetaData>
        <graphmetadata:Drawing element="Structure" id="Structure_go" scale="1.0" x="-116.0"
                    y="-314.5"/>
        <graphmetadata:Node element="User" id="User_go" x="-382.0" y="-370.5"/>
        <graphmetadata:Node element="Group" id="Group_go" x="-295.0" y="-377.0"/>
        <graphmetadata:Node element="SiteView" id="SiteView_go" x="-214.5" y="-377.0"/>
        <graphmetadata:Node element="ent1" id="ent1_go" x="-379.0" y="-282.0"/>
        <graphmetadata:Connection element="User2Group" id="User2Group_go" x="-340.5"
                    y="-353.5"/>
        <graphmetadata:Connection element="User2DefaultGroup" id="User2DefaultGroup_go"
                x="-341.0" y="-401.0"/>
        <graphmetadata:Connection element="Group2SiteView" id="Group2SiteView_go" x="" y=""/>
    </auxiliary:GraphMetaData>
    <auxiliary:ProjectDependentOptions>
        <auxiliary:Option name="SHOW_CARDINALITY" type="BOOLEAN" value="false"/>
        <auxiliary:Option name="SHOW_MASTER_OBJECT" type="BOOLEAN" value="true"/>
        <auxiliary:Option name="SHOW_ROLES" type="BOOLEAN" value="true"/>
        <auxiliary:Option name="SHOW_PARAMETER_LINK_SYMBOL" type="BOOLEAN"
                    value="true"/>
        <auxiliary:Option name="SHOW_CARDINALITY_UML_STYLE" type="BOOLEAN"
                    value="true"/>
        <auxiliary:Option name="SHOW_ATTRIBUTES_INSIDE_ENTITIES" type="BOOLEAN"
                    value="true"/>
    </auxiliary:ProjectDependentOptions>
</WebML>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE WebML SYSTEM "WebML.dtd">

<WebML xmlns:auxiliary="http://www.webml.org/auxiliary"
    xmlns:graphmetadata="http://www.webml.org/graphmetadata"
    xmlns:presentation="http://www.webml.org/presentation" auxiliary:compileJavaFiles="yes"
    auxiliary:deploy-with-names="no" auxiliary:http-port="8080" auxiliary:https-port="8443"
    auxiliary:layoutUseUnderscore="no" auxiliary:scramble-url="no" auxiliary:secure-url="no"
    auxiliary:structured-deploy="no" siteName="Untitled" version="3.0.14">
  <Structure graphmetadata:go="Structure_go" id="Structure">
    <ENTITY auxiliary:attributesVisible="true" auxiliary:testCaseCount="20"
          graphmetadata:go="User_go" id="User" name="User">
      <ATTRIBUTE id="userOID" name="OID" type="OID"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="userName" name="UserName"
            type="String"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="password" name="Password"
            type="Password"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="email" name="EMail" type="String"/>
      <RELATIONSHIP auxiliary:testCaseCount="20" graphmetadata:go="User2Group_go"
            id="User2Group" inverse="Group2User" maxCard="N" minCard="1"
            name="User_Group" roleName="User2Group" to="Group"/>
      <RELATIONSHIP auxiliary:testCaseCount="20"
            graphmetadata:go="User2DefaultGroup_go" id="User2DefaultGroup"
            inverse="DefaultGroup2User" maxCard="1" minCard="1" name="User_DefaultGroup"
            roleName="User2DefaultGroup" to="Group"/>
    </ENTITY>
    <ENTITY auxiliary:attributesVisible="true" auxiliary:testCaseCount="20"
            graphmetadata:go="Group_go" id="Group" name="Group">
      <ATTRIBUTE id="groupOID" name="OID" type="OID"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="groupName" name="GroupName"
            type="String"/>
      <RELATIONSHIP auxiliary:testCaseCount="20" graphmetadata:go="" id="Group2User"
            inverse="User2Group" maxCard="N" minCard="1" name="User_Group"
            roleName="Group2User" to="User"/>
      <RELATIONSHIP auxiliary:testCaseCount="20" graphmetadata:go=""
            id="DefaultGroup2User" inverse="User2DefaultGroup" maxCard="N" minCard="0"
            name="User_DefaultGroup" roleName="DefaultGroup2User" to="User"/>
      <RELATIONSHIP auxiliary:testCaseCount="20" graphmetadata:go="Group2SiteView_go"
            id="Group2SiteView" inverse="SiteView2Group" maxCard="1" minCard="1"
            name="Group_SiteView" roleName="Group2SiteView" to="SiteView"/>
    </ENTITY>
    <ENTITY auxiliary:attributesVisible="true" auxiliary:testCaseCount="20"
              graphmetadata:go="SiteView_go" id="SiteView" name="SiteView">
      <ATTRIBUTE id="siteViewOID" name="OID" type="OID"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="siteViewID" name="SiteViewID"
              type="String"/>
      <RELATIONSHIP auxiliary:testCaseCount="20" graphmetadata:go="" id="SiteView2Group"
            inverse="Group2SiteView" maxCard="N" minCard="1" name="Group_SiteView"
            roleName="SiteView2Group" to="Group"/>
    </ENTITY>
    <ENTITY auxiliary:attributesVisible="true" graphmetadata:go="ent1_go" id="ent1"
            name="$ENT">
      <ATTRIBUTE id="att1" name="OID" type="OID"/>
      <PROPERTY id="prop1" name="Identifier - $ENT_ID;" value="alias:"/>
      <PROPERTY id="prop2" name="ENT_ID;" value="anchor:;"/>
    </ENTITY>
  </Structure>
  <MetaStructure graphmetadata:go="MetaStructure_go" id="MetaStructure">
```

```
    <DOMAIN id="meta$LogPriority" name="Log Priority">
      <DOMAINVALUE value="ERROR"/>
      <DOMAINVALUE value="WARN"/>
      <DOMAINVALUE value="INFO"/>
      <DOMAINVALUE value="DEBUG"/>
    </DOMAIN>
    <DOMAIN id="meta$RTServiceType" name="RTService Type">
      <DOMAINVALUE value="READ"/>
      <DOMAINVALUE value="WRITE"/>
      <DOMAINVALUE value="LINK"/>
      <DOMAINVALUE value="PERMISSION"/>
      <DOMAINVALUE value="OTHER"/>
    </DOMAIN>
    <ENTITY auxiliary:attributesVisible="false" auxiliary:testCaseCount="20"
            graphmetadata:go="meta$LogEvent_go" id="meta$LogEvent" name="LogEvent">
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="meta$LogEvent$index" name="index"
            type="Integer"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="meta$LogEvent$timestamp"
               name="timestamp" type="String"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="meta$LogEvent$priority"
               name="priority" userType="meta$LogPriority"/>
      <ATTRIBUTE id="meta$LogEvent$rtServiceID" name="rtServiceID" type="String"
               value="Self.meta$LogEvent$RelatedTo.meta$RTService$identifier"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="meta$LogEvent$message"
               name="message" type="String"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="meta$LogEvent$throwable"
               name="throwable" type="Text"/>
      <RELATIONSHIP auxiliary:testCaseCount="20"
            graphmetadata:go="meta$LogEvent$RelatedTo_go" id="meta$LogEvent$RelatedTo"
            inverse="meta$RTService$LoggedEvents" maxCard="1" minCard="1"
            name="LogEvent_RTService" roleName="RelatedTo" to="meta$RTService"/>
    </ENTITY>
    <ENTITY auxiliary:attributesVisible="false" auxiliary:testCaseCount="20"
            graphmetadata:go="meta$RTService_go" id="meta$RTService" name="RTService">
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="meta$RTService$identifier"
               name="identifier" type="String"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="meta$RTService$type" name="type"
               userType="meta$RTServiceType"/>
      <ATTRIBUTE auxiliary:testCaseFile="default.txt" id="meta$RTService$hitCount"
               name="hitCount" type="Integer"/>
      <RELATIONSHIP auxiliary:testCaseCount="20" graphmetadata:go=""
            id="meta$RTService$LoggedEvents" inverse="meta$LogEvent$RelatedTo"
            maxCard="N" minCard="0" name="LogEvent_RTService" roleName="LoggedEvents"
            to="meta$LogEvent"/>
    </ENTITY>
  </MetaStructure>
  <Navigation>
    <SITEVIEW graphmetadata:go="sv1_go" id="sv1" localize="no" name="" protected="no"
            secure="no">
      <OPERATIONUNITS/>
      <PAGE graphmetadata:go="page1_go" id="page1" landmark="no" localize="no"
               name="PC" secure="no">
        <CONTENTUNITS>
          <INDEXUNIT distinct="no" entity="ent1" graphmetadata:go="inu1_go" id="inu1"
                   name="IU">
            <PROPERTY id="prop4" name="Identifier - IU_ID;" value="alias:"/>
            <PROPERTY id="prop11" name="IU=concat( ENT,'List');" value="expression:"/>
          </INDEXUNIT>
```

```
            </CONTENTUNITS>
            <PROPERTY id="prop3" name="Identifier - PC_ID;" value="alias:"/>
            <PROPERTY id="prop5" name="PC=concat( ENT,'Page');" value="expression:"/>
            <presentation:grid colcount="3" rowcount="3">
               <presentation:row>
                  <presentation:cell/>
                  <presentation:cell/>
                  <presentation:cell/>
               </presentation:row>
               <presentation:row>
                  <presentation:cell/>
                  <presentation:cell/>
                  <presentation:cell/>
               </presentation:row>
               <presentation:row>
                  <presentation:cell/>
                  <presentation:cell/>
                  <presentation:cell/>
               </presentation:row>
            </presentation:grid>
         </PAGE>
         <PROPERTY id="prop8" name="Identifier - SV_ID;" value="alias:"/>
         <PROPERTY id="prop13" name="ENT_ID;" value="anchor:"/>
         <PROPERTY id="prop14" name="SV_ID='sv1';" value="expression:"/>
      </SITEVIEW>
      <GLOBALPARAMETER duration="session" entity="User" id="UserCtxParam"
                name="UserCtxParam"/>
      <GLOBALPARAMETER duration="session" entity="Group" id="GroupCtxParam"
                name="GroupCtxParam"/>
   </Navigation>
   <Mapping>
      <rdbms:RDBMSMapping xmlns:rdbms="http://www.webml.org/mapping/rdbms"/>
   </Mapping>
   <auxiliary:GraphMetaData>
      <graphmetadata:Drawing element="Structure" id="Structure_go" scale="0.5" x="34.0"
                y="35.0"/>
      <graphmetadata:Node element="User" id="User_go" x="34.0" y="35.0"/>
      <graphmetadata:Node element="Group" id="Group_go" x="188.0" y="25.0"/>
      <graphmetadata:Node element="SiteView" id="SiteView_go" x="188.0" y="103.0"/>
      <graphmetadata:Node element="ent1" id="ent1_go" x="-49.5" y="174.0"/>
      <graphmetadata:Connection element="User2Group" id="User2Group_go" x="106.0"
                y="48.5"/>
      <graphmetadata:Connection element="User2DefaultGroup" id="User2DefaultGroup_go"
                x="107.0" y="12.5"/>
      <graphmetadata:Connection element="Group2SiteView" id="Group2SiteView_go" x="" y=""/>
      <graphmetadata:Drawing element="sv1" id="sv1_go" scale="1.0" x="443.0" y="533.0"/>
      <graphmetadata:Node element="inu1" id="inu1_go" x="255.0" y="467.0"/>
      <graphmetadata:Node element="page1" id="page1_go" x="255.0" y="468.25"/>
   </auxiliary:GraphMetaData>
   <auxiliary:ProjectDependentOptions>
      <auxiliary:Option name="SHOW_CARDINALITY" type="BOOLEAN" value="true"/>
      <auxiliary:Option name="SHOW_MASTER_OBJECT" type="BOOLEAN" value="true"/>
      <auxiliary:Option name="SHOW_ROLES" type="BOOLEAN" value="true"/>
      <auxiliary:Option name="SHOW_PARAMETER_LINK_SYMBOL" type="BOOLEAN"
                    value="true"/>
      <auxiliary:Option name="SHOW_CARDINALITY_UML_STYLE" type="BOOLEAN"
                    value="true"/>
```

```
    <auxiliary:Option name="SHOW_ATTRIBUTES_INSIDE_ENTITIES" type="BOOLEAN"
                 value="true"/>
  </auxiliary:ProjectDependentOptions>
</WebML>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>

<LR logRepSpec="WebML">
    <RM relation="WebML">
        <AI name="id" value="gen-N400001"/>
        <AI name="auxiliary:compileJavaFiles" value="yes"/>
        <AI name="auxiliary:deploy-with-names" value="no"/>
        <AI name="auxiliary:http-port" value="8080"/>
        <AI name="auxiliary:https-port" value="8443"/>
        <AI name="auxiliary:layoutUseUnderscore" value="no"/>
        <AI name="auxiliary:scramble-url" value="no"/>
        <AI name="auxiliary:secure-url" value="no"/>
        <AI name="auxiliary:structured-deploy" value="no"/>
        <AI name="siteName" value="Untitled"/>
        <AI name="version" value="3.0.14"/>
    </RM>
    <RM relation="Structure">
        <AI name="parent" value="gen-N400001"/>
        <AI name="graphmetadata:go" value="Structure_go"/>
        <AI name="id" value="Structure"/>
    </RM>
    <RM relation="Entity">
        <AI name="auxiliary:attributesVisible" value="true"/>
        <AI name="auxiliary:testCaseCount" value="20"/>
        <AI name="graphmetadata:go" value="User_go"/>
        <AI name="entity" value="User"/>
        <AI name="name" value="User"/>
    </RM>
    <RM relation="Attribute">
        <AI name="definedAt" value="User"/>
        <AI name="attribute" value="userOID"/>
        <AI name="name" value="OID"/>
        <AI name="type" value="OID"/>
    </RM>
    <RM relation="Attribute">
        <AI name="definedAt" value="User"/>
        <AI name="auxiliary:testCaseFile" value="default.txt"/>
        <AI name="attribute" value="userName"/>
        <AI name="name" value="UserName"/>
        <AI name="type" value="String"/>
    </RM>
    <RM relation="Attribute">
        <AI name="definedAt" value="User"/>
        <AI name="auxiliary:testCaseFile" value="default.txt"/>
        <AI name="attribute" value="password"/>
        <AI name="name" value="Password"/>
        <AI name="type" value="Password"/>
    </RM>
    <RM relation="Attribute">
        <AI name="definedAt" value="User"/>
        <AI name="auxiliary:testCaseFile" value="default.txt"/>
        <AI name="attribute" value="email"/>
        <AI name="name" value="EMail"/>
        <AI name="type" value="String"/>
    </RM>
    <RM relation="RelRole">
        <AI name="from" value="User"/>
        <AI name="auxiliary:testCaseCount" value="20"/>
    <AI name="graphmetadata:go" value="User2Group_go"/>
    <AI name="relRole" value="User2Group"/>
    <AI name="inverse" value="Group2User"/>
    <AI name="maxCard" value="N"/>
    <AI name="minCard" value="1"/>
    <AI name="relShipName" value="User_Group"/>
    <AI name="relRoleName" value="User2Group"/>
    <AI name="to" value="Group"/>
</RM>
<RM relation="RelRole">
    <AI name="from" value="User"/>
    <AI name="auxiliary:testCaseCount" value="20"/>
    <AI name="graphmetadata:go" value="User2DefaultGroup_go"/>
    <AI name="relRole" value="User2DefaultGroup"/>
    <AI name="inverse" value="DefaultGroup2User"/>
    <AI name="maxCard" value="1"/>
    <AI name="minCard" value="1"/>
    <AI name="relShipName" value="User_DefaultGroup"/>
    <AI name="relRoleName" value="User2DefaultGroup"/>
    <AI name="to" value="Group"/>
</RM>
<RM relation="Entity">
    <AI name="auxiliary:attributesVisible" value="true"/>
    <AI name="auxiliary:testCaseCount" value="20"/>
    <AI name="graphmetadata:go" value="Group_go"/>
    <AI name="entity" value="Group"/>
    <AI name="name" value="Group"/>
</RM>
<RM relation="Attribute">
    <AI name="definedAt" value="Group"/>
    <AI name="attribute" value="groupOID"/>
    <AI name="name" value="OID"/>
    <AI name="type" value="OID"/>
</RM>
<RM relation="Attribute">
    <AI name="definedAt" value="Group"/>
    <AI name="auxiliary:testCaseFile" value="default.txt"/>
    <AI name="attribute" value="groupName"/>
    <AI name="name" value="GroupName"/>
    <AI name="type" value="String"/>
</RM>
<RM relation="RelRole">
    <AI name="from" value="Group"/>
    <AI name="auxiliary:testCaseCount" value="20"/>
    <AI name="relRole" value="Group2User"/>
    <AI name="inverse" value="User2Group"/>
    <AI name="maxCard" value="N"/>
    <AI name="minCard" value="1"/>
    <AI name="relShipName" value="User_Group"/>
    <AI name="relRoleName" value="Group2User"/>
    <AI name="to" value="User"/>
</RM>
<RM relation="RelRole">
    <AI name="from" value="Group"/>
    <AI name="auxiliary:testCaseCount" value="20"/>
    <AI name="relRole" value="DefaultGroup2User"/>
    <AI name="inverse" value="User2DefaultGroup"/>
    <AI name="maxCard" value="N"/>
```

```
        <AI name="minCard" value="0"/>
        <AI name="relShipName" value="User_DefaultGroup"/>
        <AI name="relRoleName" value="DefaultGroup2User"/>
        <AI name="to" value="User"/>
    </RM>
    <RM relation="RelRole">
        <AI name="from" value="Group"/>
        <AI name="auxiliary:testCaseCount" value="20"/>
        <AI name="graphmetadata:go" value="Group2SiteView_go"/>
        <AI name="relRole" value="Group2SiteView"/>
        <AI name="inverse" value="SiteView2Group"/>
        <AI name="maxCard" value="1"/>
        <AI name="minCard" value="1"/>
        <AI name="relShipName" value="Group_SiteView"/>
        <AI name="relRoleName" value="Group2SiteView"/>
        <AI name="to" value="SiteView"/>
    </RM>
    <RM relation="Entity">
        <AI name="auxiliary:attributesVisible" value="true"/>
        <AI name="auxiliary:testCaseCount" value="20"/>
        <AI name="graphmetadata:go" value="SiteView_go"/>
        <AI name="entity" value="SiteView"/>
        <AI name="name" value="SiteView"/>
    </RM>
    <RM relation="Attribute">
        <AI name="definedAt" value="SiteView"/>
        <AI name="attribute" value="siteViewOID"/>
        <AI name="name" value="OID"/>
        <AI name="type" value="OID"/>
    </RM>
    <RM relation="Attribute">
        <AI name="definedAt" value="SiteView"/>
        <AI name="auxiliary:testCaseFile" value="default.txt"/>
        <AI name="attribute" value="siteViewID"/>
        <AI name="name" value="SiteViewID"/>
        <AI name="type" value="String"/>
    </RM>
    <RM relation="RelRole">
        <AI name="from" value="SiteView"/>
        <AI name="auxiliary:testCaseCount" value="20"/>
        <AI name="relRole" value="SiteView2Group"/>
        <AI name="inverse" value="Group2SiteView"/>
        <AI name="maxCard" value="N"/>
        <AI name="minCard" value="1"/>
        <AI name="relShipName" value="Group_SiteView"/>
        <AI name="relRoleName" value="SiteView2Group"/>
        <AI name="to" value="Group"/>
    </RM>
    <RM relation="Entity">
        <AI name="auxiliary:attributesVisible" value="true"/>
        <AI name="graphmetadata:go" value="ent1_go"/>
        <AI name="entity" value="ENT_ID"/>
        <AI name="name" value="ENT"/>
    </RM>
    <RM relation="Attribute">
        <AI name="definedAt" value="ENT_ID"/>
        <AI name="attribute" value="att1"/>
        <AI name="name" value="OID"/>
        <AI name="type" value="OID"/>
    </RM>
    <RM relation="PROPERTY">
        <AI name="parent" value="ENT_ID"/>
        <AI name="id" value="prop1"/>
        <AI name="name" value="Identifier - $ENT_ID;"/>
        <AI name="value" value="alias:"/>
    </RM>
    <RM relation="PROPERTY">
        <AI name="parent" value="ENT_ID"/>
        <AI name="id" value="prop2"/>
        <AI name="name" value="ENT_ID;"/>
        <AI name="value" value="anchor:;"/>
    </RM>
    <RM relation="MetaStructure">
        <AI name="parent" value="gen-N400001"/>
        <AI name="graphmetadata:go" value="MetaStructure_go"/>
        <AI name="id" value="MetaStructure"/>
    </RM>
    <RM relation="DOMAIN">
        <AI name="parent" value="MetaStructure"/>
        <AI name="id" value="meta$LogPriority"/>
        <AI name="name" value="Log Priority"/>
    </RM>
    <RM relation="DOMAINVALUE">
        <AI name="parent" value="meta$LogPriority"/>
        <AI name="id" value="gen-N4000BB"/>
        <AI name="value" value="ERROR"/>
    </RM>
    <RM relation="DOMAINVALUE">
        <AI name="parent" value="meta$LogPriority"/>
        <AI name="id" value="gen-N4000BE"/>
        <AI name="value" value="WARN"/>
    </RM>
    <RM relation="DOMAINVALUE">
        <AI name="parent" value="meta$LogPriority"/>
        <AI name="id" value="gen-N4000C1"/>
        <AI name="value" value="INFO"/>
    </RM>
    <RM relation="DOMAINVALUE">
        <AI name="parent" value="meta$LogPriority"/>
        <AI name="id" value="gen-N4000C4"/>
        <AI name="value" value="DEBUG"/>
    </RM>
    <RM relation="DOMAIN">
        <AI name="parent" value="MetaStructure"/>
        <AI name="id" value="meta$RTServiceType"/>
        <AI name="name" value="RTService Type"/>
    </RM>
    <RM relation="DOMAINVALUE">
        <AI name="parent" value="meta$RTServiceType"/>
        <AI name="id" value="gen-N4000CC"/>
        <AI name="value" value="READ"/>
    </RM>
    <RM relation="DOMAINVALUE">
        <AI name="parent" value="meta$RTServiceType"/>
        <AI name="id" value="gen-N4000CF"/>
    <AI name="value" value="WRITE"/>
</RM>
<RM relation="DOMAINVALUE">
    <AI name="parent" value="meta$RTServiceType"/>
    <AI name="id" value="gen-N4000D2"/>
    <AI name="value" value="LINK"/>
</RM>
<RM relation="DOMAINVALUE">
    <AI name="parent" value="meta$RTServiceType"/>
    <AI name="id" value="gen-N4000D5"/>
    <AI name="value" value="PERMISSION"/>
</RM>
<RM relation="DOMAINVALUE">
    <AI name="parent" value="meta$RTServiceType"/>
    <AI name="id" value="gen-N4000D8"/>
    <AI name="value" value="OTHER"/>
</RM>
<RM relation="ENTITY">
    <AI name="parent" value="MetaStructure"/>
    <AI name="auxiliary:attributesVisible" value="false"/>
    <AI name="auxiliary:testCaseCount" value="20"/>
    <AI name="graphmetadata:go" value="meta$LogEvent_go"/>
    <AI name="id" value="meta$LogEvent"/>
    <AI name="name" value="LogEvent"/>
</RM>
<RM relation="ATTRIBUTE">
    <AI name="parent" value="meta$LogEvent"/>
    <AI name="auxiliary:testCaseFile" value="default.txt"/>
    <AI name="id" value="meta$LogEvent$index"/>
    <AI name="name" value="index"/>
    <AI name="type" value="Integer"/>
</RM>
<RM relation="ATTRIBUTE">
    <AI name="parent" value="meta$LogEvent"/>
    <AI name="auxiliary:testCaseFile" value="default.txt"/>
    <AI name="id" value="meta$LogEvent$timestamp"/>
    <AI name="name" value="timestamp"/>
    <AI name="type" value="String"/>
</RM>
<RM relation="ATTRIBUTE">
    <AI name="parent" value="meta$LogEvent"/>
    <AI name="auxiliary:testCaseFile" value="default.txt"/>
    <AI name="id" value="meta$LogEvent$priority"/>
    <AI name="name" value="priority"/>
    <AI name="userType" value="meta$LogPriority"/>
</RM>
<RM relation="ATTRIBUTE">
    <AI name="parent" value="meta$LogEvent"/>
    <AI name="id" value="meta$LogEvent$rtServiceID"/>
    <AI name="name" value="rtServiceID"/>
    <AI name="type" value="String"/>
    <AI name="value" value="Self.meta$LogEvent$RelatedTo.meta$RTService$identifier"/>
</RM>
<RM relation="ATTRIBUTE">
    <AI name="parent" value="meta$LogEvent"/>
    <AI name="auxiliary:testCaseFile" value="default.txt"/>
    <AI name="id" value="meta$LogEvent$message"/>
```

```
        <AI name="name" value="message"/>
        <AI name="type" value="String"/>
    </RM>
    <RM relation="ATTRIBUTE">
        <AI name="parent" value="meta$LogEvent"/>
        <AI name="auxiliary:testCaseFile" value="default.txt"/>
        <AI name="id" value="meta$LogEvent$throwable"/>
        <AI name="name" value="throwable"/>
        <AI name="type" value="Text"/>
    </RM>
    <RM relation="RELATIONSHIP">
        <AI name="parent" value="meta$LogEvent"/>
        <AI name="auxiliary:testCaseCount" value="20"/>
        <AI name="graphmetadata:go" value="meta$LogEvent$RelatedTo_go"/>
        <AI name="id" value="meta$LogEvent$RelatedTo"/>
        <AI name="inverse" value="meta$RTService$LoggedEvents"/>
        <AI name="maxCard" value="1"/>
        <AI name="minCard" value="1"/>
        <AI name="name" value="LogEvent_RTService"/>
        <AI name="roleName" value="RelatedTo"/>
        <AI name="to" value="meta$RTService"/>
    </RM>
    <RM relation="ENTITY">
        <AI name="parent" value="MetaStructure"/>
        <AI name="auxiliary:attributesVisible" value="false"/>
        <AI name="auxiliary:testCaseCount" value="20"/>
        <AI name="graphmetadata:go" value="meta$RTService_go"/>
        <AI name="id" value="meta$RTService"/>
        <AI name="name" value="RTService"/>
    </RM>
    <RM relation="ATTRIBUTE">
        <AI name="parent" value="meta$RTService"/>
        <AI name="auxiliary:testCaseFile" value="default.txt"/>
        <AI name="id" value="meta$RTService$identifier"/>
        <AI name="name" value="identifier"/>
        <AI name="type" value="String"/>
    </RM>
    <RM relation="ATTRIBUTE">
        <AI name="parent" value="meta$RTService"/>
        <AI name="auxiliary:testCaseFile" value="default.txt"/>
        <AI name="id" value="meta$RTService$type"/>
        <AI name="name" value="type"/>
        <AI name="userType" value="meta$RTServiceType"/>
    </RM>
    <RM relation="ATTRIBUTE">
        <AI name="parent" value="meta$RTService"/>
        <AI name="auxiliary:testCaseFile" value="default.txt"/>
        <AI name="id" value="meta$RTService$hitCount"/>
        <AI name="name" value="hitCount"/>
        <AI name="type" value="Integer"/>
    </RM>
    <RM relation="RELATIONSHIP">
        <AI name="parent" value="meta$RTService"/>
        <AI name="auxiliary:testCaseCount" value="20"/>
        <AI name="id" value="meta$RTService$LoggedEvents"/>
        <AI name="inverse" value="meta$LogEvent$RelatedTo"/>
        <AI name="maxCard" value="N"/>
        <AI name="minCard" value="0"/>
        <AI name="name" value="LogEvent_RTService"/>
        <AI name="roleName" value="LoggedEvents"/>
        <AI name="to" value="meta$LogEvent"/>
    </RM>
    <RM relation="Navigation">
        <AI name="parent" value="gen-N400001"/>
        <AI name="id" value="gen-N400139"/>
    </RM>
    <RM relation="SiteView">
        <AI name="graphmetadata:go" value="sv1_go"/>
        <AI name="siteView" value="SV_ID"/>
        <AI name="localize" value="no"/>
        <AI name="protected" value="no"/>
        <AI name="secure" value="no"/>
    </RM>
    <RM relation="OPERATIONUNITS">
        <AI name="parent" value="SV_ID"/>
        <AI name="id" value="gen-N400143"/>
    </RM>
    <RM relation="Page">
        <AI name="definedAt" value="SV_ID"/>
        <AI name="graphmetadata:go" value="page1_go"/>
        <AI name="page" value="PC_ID"/>
        <AI name="landmark" value="no"/>
        <AI name="localize" value="no"/>
        <AI name="name" value="PC"/>
        <AI name="secure" value="no"/>
    </RM>
    <RM relation="CONTENTUNITS">
        <AI name="parent" value="PC_ID"/>
        <AI name="id" value="gen-N40014D"/>
    </RM>
    <RM relation="IndexUnit">
        <AI name="definedAt" value="PC_ID"/>
        <AI name="distinct" value="no"/>
        <AI name="entity" value="ENT_ID"/>
        <AI name="graphmetadata:go" value="inu1_go"/>
        <AI name="indexUnit" value="IU_ID"/>
        <AI name="name" value="IU"/>
    </RM>
    <RM relation="PROPERTY">
        <AI name="parent" value="IU_ID"/>
        <AI name="id" value="prop4"/>
        <AI name="name" value="Identifier - IU_ID;"/>
        <AI name="value" value="alias:"/>
    </RM>
    <RM relation="PROPERTY">
        <AI name="parent" value="IU_ID"/>
        <AI name="id" value="prop11"/>
        <AI name="name" value="IU=concat( ENT,'List');"/>
        <AI name="value" value="expression:"/>
    </RM>
    <RM relation="PROPERTY">
        <AI name="parent" value="PC_ID"/>
        <AI name="id" value="prop3"/>
        <AI name="name" value="Identifier - PC_ID;"/>
        <AI name="value" value="alias:"/>
```

```
</RM>
<RM relation="PROPERTY">
    <AI name="parent" value="PC_ID"/>
    <AI name="id" value="prop5"/>
    <AI name="name" value="PC=concat( ENT,'Page');"/>
    <AI name="value" value="expression:"/>
</RM>
<RM relation="presentation:grid">
    <AI name="parent" value="PC_ID"/>
    <AI name="id" value="gen-N40016C"/>
    <AI name="colcount" value="3"/>
    <AI name="rowcount" value="3"/>
</RM>
<RM relation="presentation:row">
    <AI name="parent" value="gen-N40016C"/>
    <AI name="id" value="gen-N400170"/>
</RM>
<RM relation="presentation:cell">
    <AI name="parent" value="gen-N400170"/>
    <AI name="id" value="gen-N400172"/>
</RM>
<RM relation="presentation:cell">
    <AI name="parent" value="gen-N400170"/>
    <AI name="id" value="gen-N400174"/>
</RM>
<RM relation="presentation:cell">
    <AI name="parent" value="gen-N400170"/>
    <AI name="id" value="gen-N400176"/>
</RM>
<RM relation="presentation:row">
    <AI name="parent" value="gen-N40016C"/>
    <AI name="id" value="gen-N400179"/>
</RM>
<RM relation="presentation:cell">
    <AI name="parent" value="gen-N400179"/>
    <AI name="id" value="gen-N40017B"/>
</RM>
<RM relation="presentation:cell">
    <AI name="parent" value="gen-N400179"/>
    <AI name="id" value="gen-N40017D"/>
</RM>
<RM relation="presentation:cell">
    <AI name="parent" value="gen-N400179"/>
    <AI name="id" value="gen-N40017F"/>
</RM>
<RM relation="presentation:row">
    <AI name="parent" value="gen-N40016C"/>
    <AI name="id" value="gen-N400182"/>
</RM>
<RM relation="presentation:cell">
    <AI name="parent" value="gen-N400182"/>
    <AI name="id" value="gen-N400184"/>
</RM>
<RM relation="presentation:cell">
    <AI name="parent" value="gen-N400182"/>
    <AI name="id" value="gen-N400186"/>
</RM>
<RM relation="presentation:cell">
```

```
        <AI name="parent" value="gen-N400182"/>
        <AI name="id" value="gen-N400188"/>
    </RM>
    <RM relation="PROPERTY">
        <AI name="parent" value="SV_ID"/>
        <AI name="id" value="prop8"/>
        <AI name="name" value="Identifier - SV_ID;"/>
        <AI name="value" value="alias:"/>
    </RM>
    <RM relation="PROPERTY">
        <AI name="parent" value="SV_ID"/>
        <AI name="id" value="prop13"/>
        <AI name="name" value="ENT_ID;"/>
        <AI name="value" value="anchor:"/>
    </RM>
    <RM relation="PROPERTY">
        <AI name="parent" value="SV_ID"/>
        <AI name="id" value="prop14"/>
        <AI name="name" value="SV_ID='sv1';"/>
        <AI name="value" value="expression:"/>
    </RM>
    <RM relation="GLOBALPARAMETER">
        <AI name="parent" value="gen-N400139"/>
        <AI name="duration" value="session"/>
        <AI name="entity" value="User"/>
        <AI name="id" value="UserCtxParam"/>
        <AI name="name" value="UserCtxParam"/>
    </RM>
    <RM relation="GLOBALPARAMETER">
        <AI name="parent" value="gen-N400139"/>
        <AI name="duration" value="session"/>
        <AI name="entity" value="Group"/>
        <AI name="id" value="GroupCtxParam"/>
        <AI name="name" value="GroupCtxParam"/>
    </RM>
    <RM relation="Mapping">
        <AI name="parent" value="gen-N400001"/>
        <AI name="id" value="gen-N4001AA"/>
    </RM>
    <RM relation="rdbms:RDBMSMapping">
        <AI name="parent" value="gen-N4001AA"/>
        <AI name="id" value="gen-N4001AC"/>
    </RM>
    <RM relation="auxiliary:GraphMetaData">
        <AI name="parent" value="gen-N400001"/>
        <AI name="id" value="gen-N4001B0"/>
    </RM>
    <RM relation="graphmetadata:Drawing">
        <AI name="parent" value="gen-N4001B0"/>
        <AI name="element" value="Structure"/>
        <AI name="id" value="Structure_go"/>
        <AI name="scale" value="0.5"/>
        <AI name="x" value="34.0"/>
        <AI name="y" value="35.0"/>
    </RM>
    <RM relation="graphmetadata:Node">
        <AI name="parent" value="gen-N4001B0"/>
        <AI name="element" value="User"/>
        <AI name="id" value="User_go"/>
        <AI name="x" value="34.0"/>
        <AI name="y" value="35.0"/>
    </RM>
    <RM relation="graphmetadata:Node">
        <AI name="parent" value="gen-N4001B0"/>
        <AI name="element" value="Group"/>
        <AI name="id" value="Group_go"/>
        <AI name="x" value="188.0"/>
        <AI name="y" value="25.0"/>
    </RM>
    <RM relation="graphmetadata:Node">
        <AI name="parent" value="gen-N4001B0"/>
        <AI name="element" value="SiteView"/>
        <AI name="id" value="SiteView_go"/>
        <AI name="x" value="188.0"/>
        <AI name="y" value="103.0"/>
    </RM>
    <RM relation="EntityPos">
        <AI name="element" value="ENT_ID"/>
        <AI name="pos" value="ent1_go"/>
        <AI name="xValue" value="-49.5"/>
        <AI name="yValue" value="174.0"/>
    </RM>
    <RM relation="graphmetadata:Connection">
        <AI name="parent" value="gen-N4001B0"/>
        <AI name="element" value="User2Group"/>
        <AI name="id" value="User2Group_go"/>
        <AI name="x" value="106.0"/>
        <AI name="y" value="48.5"/>
    </RM>
    <RM relation="graphmetadata:Connection">
        <AI name="parent" value="gen-N4001B0"/>
        <AI name="element" value="User2DefaultGroup"/>
        <AI name="id" value="User2DefaultGroup_go"/>
        <AI name="x" value="107.0"/>
        <AI name="y" value="12.5"/>
    </RM>
    <RM relation="graphmetadata:Connection">
        <AI name="parent" value="gen-N4001B0"/>
        <AI name="element" value="Group2SiteView"/>
        <AI name="id" value="Group2SiteView_go"/>
    </RM>
    <RM relation="graphmetadata:Drawing">
        <AI name="parent" value="gen-N4001B0"/>
        <AI name="element" value="SV_ID"/>
        <AI name="id" value="sv1_go"/>
        <AI name="scale" value="1.0"/>
        <AI name="x" value="443.0"/>
        <AI name="y" value="533.0"/>
    </RM>
    <RM relation="IndexUnitPos">
        <AI name="element" value="IU_ID"/>
        <AI name="pos" value="inu1_go"/>
        <AI name="xValue" value="255.0"/>
        <AI name="yValue" value="467.0"/>
    </RM>
    <RM relation="PagePos">
        <AI name="element" value="PC_ID"/>
        <AI name="pos" value="page1_go"/>
        <AI name="xValue" value="255.0"/>
        <AI name="yValue" value="468.25"/>
    </RM>
    <RM relation="auxiliary:ProjectDependentOptions">
        <AI name="parent" value="gen-N400001"/>
        <AI name="id" value="gen-N4001F7"/>
    </RM>
    <RM relation="auxiliary:Option">
        <AI name="parent" value="gen-N4001F7"/>
        <AI name="id" value="gen-N4001F9"/>
        <AI name="name" value="SHOW_CARDINALITY"/>
        <AI name="type" value="BOOLEAN"/>
        <AI name="value" value="true"/>
    </RM>
    <RM relation="auxiliary:Option">
        <AI name="parent" value="gen-N4001F7"/>
        <AI name="id" value="gen-N4001FE"/>
        <AI name="name" value="SHOW_MASTER_OBJECT"/>
        <AI name="type" value="BOOLEAN"/>
        <AI name="value" value="true"/>
    </RM>
    <RM relation="auxiliary:Option">
        <AI name="parent" value="gen-N4001F7"/>
        <AI name="id" value="gen-N400203"/>
        <AI name="name" value="SHOW_ROLES"/>
        <AI name="type" value="BOOLEAN"/>
        <AI name="value" value="true"/>
    </RM>
    <RM relation="auxiliary:Option">
        <AI name="parent" value="gen-N4001F7"/>
        <AI name="id" value="gen-N400208"/>
        <AI name="name" value="SHOW_PARAMETER_LINK_SYMBOL"/>
        <AI name="type" value="BOOLEAN"/>
        <AI name="value" value="true"/>
    </RM>
    <RM relation="auxiliary:Option">
        <AI name="parent" value="gen-N4001F7"/>
        <AI name="id" value="gen-N40020D"/>
        <AI name="name" value="SHOW_CARDINALITY_UML_STYLE"/>
        <AI name="type" value="BOOLEAN"/>
        <AI name="value" value="true"/>
    </RM>
    <RM relation="auxiliary:Option">
        <AI name="parent" value="gen-N4001F7"/>
        <AI name="id" value="gen-N400212"/>
        <AI name="name" value="SHOW_ATTRIBUTES_INSIDE_ENTITIES"/>
        <AI name="type" value="BOOLEAN"/>
        <AI name="value" value="true"/>
    </RM>
</LR>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>

<LR logRepSpec="WebML">
    <RM relation="WebML">
        <AI name="id" value="gen-N400001"/>
        <AI name="auxiliary:compileJavaFiles" value="yes"/>
        <AI name="auxiliary:deploy-with-names" value="no"/>
        <AI name="auxiliary:http-port" value="8080"/>
        <AI name="auxiliary:https-port" value="8443"/>
        <AI name="auxiliary:layoutUseUnderscore" value="no"/>
        <AI name="auxiliary:scramble-url" value="no"/>
        <AI name="auxiliary:secure-url" value="no"/>
        <AI name="auxiliary:structured-deploy" value="no"/>
        <AI name="siteName" value="Untitled"/>
        <AI name="version" value="3.0.14"/>
    </RM>
    <RM relation="Structure">
        <AI name="parent" value="gen-N400001"/>
        <AI name="graphmetadata:go" value="Structure_go"/>
        <AI name="id" value="Structure"/>
    </RM>
    <RM relation="Entity">
        <AI name="auxiliary:attributesVisible" value="true"/>
        <AI name="auxiliary:testCaseCount" value="20"/>
        <AI name="graphmetadata:go" value="User_go"/>
        <AI name="entity" value="User"/>
        <AI name="name" value="User"/>
    </RM>
    <RM relation="Attribute">
        <AI name="definedAt" value="User"/>
        <AI name="attribute" value="userOID"/>
        <AI name="name" value="OID"/>
        <AI name="type" value="OID"/>
    </RM>
    <RM relation="Attribute">
        <AI name="definedAt" value="User"/>
        <AI name="auxiliary:testCaseFile" value="default.txt"/>
        <AI name="attribute" value="userName"/>
        <AI name="name" value="UserName"/>
        <AI name="type" value="String"/>
    </RM>
    <RM relation="Attribute">
        <AI name="definedAt" value="User"/>
        <AI name="auxiliary:testCaseFile" value="default.txt"/>
        <AI name="attribute" value="password"/>
        <AI name="name" value="Password"/>
        <AI name="type" value="Password"/>
    </RM>
    <RM relation="Attribute">
        <AI name="definedAt" value="User"/>
        <AI name="auxiliary:testCaseFile" value="default.txt"/>
        <AI name="attribute" value="email"/>
        <AI name="name" value="EMail"/>
        <AI name="type" value="String"/>
    </RM>
    <RM relation="RelRole">
        <AI name="from" value="User"/>
        <AI name="auxiliary:testCaseCount" value="20"/>
    <AI name="graphmetadata:go" value="User2Group_go"/>
    <AI name="relRole" value="User2Group"/>
    <AI name="inverse" value="Group2User"/>
    <AI name="maxCard" value="N"/>
    <AI name="minCard" value="1"/>
    <AI name="relShipName" value="User_Group"/>
    <AI name="relRoleName" value="User2Group"/>
    <AI name="to" value="Group"/>
</RM>
<RM relation="RelRole">
    <AI name="from" value="User"/>
    <AI name="auxiliary:testCaseCount" value="20"/>
    <AI name="graphmetadata:go" value="User2DefaultGroup_go"/>
    <AI name="relRole" value="User2DefaultGroup"/>
    <AI name="inverse" value="DefaultGroup2User"/>
    <AI name="maxCard" value="1"/>
    <AI name="minCard" value="1"/>
    <AI name="relShipName" value="User_DefaultGroup"/>
    <AI name="relRoleName" value="User2DefaultGroup"/>
    <AI name="to" value="Group"/>
</RM>
<RM relation="Entity">
    <AI name="auxiliary:attributesVisible" value="true"/>
    <AI name="auxiliary:testCaseCount" value="20"/>
    <AI name="graphmetadata:go" value="Group_go"/>
    <AI name="entity" value="Group"/>
    <AI name="name" value="Group"/>
</RM>
<RM relation="Attribute">
    <AI name="definedAt" value="Group"/>
    <AI name="attribute" value="groupOID"/>
    <AI name="name" value="OID"/>
    <AI name="type" value="OID"/>
</RM>
<RM relation="Attribute">
    <AI name="definedAt" value="Group"/>
    <AI name="auxiliary:testCaseFile" value="default.txt"/>
    <AI name="attribute" value="groupName"/>
    <AI name="name" value="GroupName"/>
    <AI name="type" value="String"/>
</RM>
<RM relation="RelRole">
    <AI name="from" value="Group"/>
    <AI name="auxiliary:testCaseCount" value="20"/>
    <AI name="relRole" value="Group2User"/>
    <AI name="inverse" value="User2Group"/>
    <AI name="maxCard" value="N"/>
    <AI name="minCard" value="1"/>
    <AI name="relShipName" value="User_Group"/>
    <AI name="relRoleName" value="Group2User"/>
    <AI name="to" value="User"/>
</RM>
<RM relation="RelRole">
    <AI name="from" value="Group"/>
    <AI name="auxiliary:testCaseCount" value="20"/>
    <AI name="relRole" value="DefaultGroup2User"/>
    <AI name="inverse" value="User2DefaultGroup"/>
    <AI name="maxCard" value="N"/>
```

```
        <AI name="minCard" value="0"/>
        <AI name="relShipName" value="User_DefaultGroup"/>
        <AI name="relRoleName" value="DefaultGroup2User"/>
        <AI name="to" value="User"/>
    </RM>
    <RM relation="RelRole">
        <AI name="from" value="Group"/>
        <AI name="auxiliary:testCaseCount" value="20"/>
        <AI name="graphmetadata:go" value="Group2SiteView_go"/>
        <AI name="relRole" value="Group2SiteView"/>
        <AI name="inverse" value="SiteView2Group"/>
        <AI name="maxCard" value="1"/>
        <AI name="minCard" value="1"/>
        <AI name="relShipName" value="Group_SiteView"/>
        <AI name="relRoleName" value="Group2SiteView"/>
        <AI name="to" value="SiteView"/>
    </RM>
    <RM relation="Entity">
        <AI name="auxiliary:attributesVisible" value="true"/>
        <AI name="auxiliary:testCaseCount" value="20"/>
        <AI name="graphmetadata:go" value="SiteView_go"/>
        <AI name="entity" value="SiteView"/>
        <AI name="name" value="SiteView"/>
    </RM>
    <RM relation="Attribute">
        <AI name="definedAt" value="SiteView"/>
        <AI name="attribute" value="siteViewOID"/>
        <AI name="name" value="OID"/>
        <AI name="type" value="OID"/>
    </RM>
    <RM relation="Attribute">
        <AI name="definedAt" value="SiteView"/>
        <AI name="auxiliary:testCaseFile" value="default.txt"/>
        <AI name="attribute" value="siteViewID"/>
        <AI name="name" value="SiteViewID"/>
        <AI name="type" value="String"/>
    </RM>
    <RM relation="RelRole">
        <AI name="from" value="SiteView"/>
        <AI name="auxiliary:testCaseCount" value="20"/>
        <AI name="relRole" value="SiteView2Group"/>
        <AI name="inverse" value="Group2SiteView"/>
        <AI name="maxCard" value="N"/>
        <AI name="minCard" value="1"/>
        <AI name="relShipName" value="Group_SiteView"/>
        <AI name="relRoleName" value="SiteView2Group"/>
        <AI name="to" value="Group"/>
    </RM>
    <RM relation="Entity">
        <AI name="auxiliary:attributesVisible" value="true"/>
        <AI name="graphmetadata:go" value="ent1_go"/>
        <AI name="entity" value="ENT_ID"/>
        <AI name="name" value="ENT"/>
    </RM>
    <RM relation="Attribute">
        <AI name="definedAt" value="ENT_ID"/>
        <AI name="attribute" value="att1"/>
        <AI name="name" value="OID"/>
        <AI name="type" value="OID"/>
    </RM>
    <RM relation="Attribute">
        <AI name="definedAt" value="ENT_ID"/>
        <AI name="attribute" value="ATT_ID"/>
        <AI name="type" value="String"/>
    </RM>
    <RM relation="PROPERTY">
        <AI name="parent" value="ATT_ID"/>
        <AI name="id" value="prop2"/>
        <AI name="name" value="Identifier - ATT_ID;"/>
        <AI name="value" value="alias:"/>
    </RM>
    <RM relation="PROPERTY">
        <AI name="parent" value="ENT_ID"/>
        <AI name="id" value="prop1"/>
        <AI name="name" value="Identifier - $ENT_ID;"/>
        <AI name="value" value="alias:"/>
    </RM>
    <RM relation="MetaStructure">
        <AI name="parent" value="gen-N400001"/>
        <AI name="graphmetadata:go" value="MetaStructure_go"/>
        <AI name="id" value="MetaStructure"/>
    </RM>
    <RM relation="DOMAIN">
        <AI name="parent" value="MetaStructure"/>
        <AI name="id" value="meta$LogPriority"/>
        <AI name="name" value="Log Priority"/>
    </RM>
    <RM relation="DOMAINVALUE">
        <AI name="parent" value="meta$LogPriority"/>
        <AI name="id" value="gen-N4000C1"/>
        <AI name="value" value="ERROR"/>
    </RM>
    <RM relation="DOMAINVALUE">
        <AI name="parent" value="meta$LogPriority"/>
        <AI name="id" value="gen-N4000C4"/>
        <AI name="value" value="WARN"/>
    </RM>
    <RM relation="DOMAINVALUE">
        <AI name="parent" value="meta$LogPriority"/>
        <AI name="id" value="gen-N4000C7"/>
        <AI name="value" value="INFO"/>
    </RM>
    <RM relation="DOMAINVALUE">
        <AI name="parent" value="meta$LogPriority"/>
        <AI name="id" value="gen-N4000CA"/>
        <AI name="value" value="DEBUG"/>
    </RM>
    <RM relation="DOMAIN">
        <AI name="parent" value="MetaStructure"/>
        <AI name="id" value="meta$RTServiceType"/>
        <AI name="name" value="RTService Type"/>
    </RM>
    <RM relation="DOMAINVALUE">
        <AI name="parent" value="meta$RTServiceType"/>
        <AI name="id" value="gen-N4000D2"/>
    <AI name="value" value="READ"/>
</RM>
<RM relation="DOMAINVALUE">
    <AI name="parent" value="meta$RTServiceType"/>
    <AI name="id" value="gen-N4000D5"/>
    <AI name="value" value="WRITE"/>
</RM>
<RM relation="DOMAINVALUE">
    <AI name="parent" value="meta$RTServiceType"/>
    <AI name="id" value="gen-N4000D8"/>
    <AI name="value" value="LINK"/>
</RM>
<RM relation="DOMAINVALUE">
    <AI name="parent" value="meta$RTServiceType"/>
    <AI name="id" value="gen-N4000DB"/>
    <AI name="value" value="PERMISSION"/>
</RM>
<RM relation="DOMAINVALUE">
    <AI name="parent" value="meta$RTServiceType"/>
    <AI name="id" value="gen-N4000DE"/>
    <AI name="value" value="OTHER"/>
</RM>
<RM relation="ENTITY">
    <AI name="parent" value="MetaStructure"/>
    <AI name="auxiliary:attributesVisible" value="false"/>
    <AI name="auxiliary:testCaseCount" value="20"/>
    <AI name="graphmetadata:go" value="meta$LogEvent_go"/>
    <AI name="id" value="meta$LogEvent"/>
    <AI name="name" value="LogEvent"/>
</RM>
<RM relation="ATTRIBUTE">
    <AI name="parent" value="meta$LogEvent"/>
    <AI name="auxiliary:testCaseFile" value="default.txt"/>
    <AI name="id" value="meta$LogEvent$index"/>
    <AI name="name" value="index"/>
    <AI name="type" value="Integer"/>
</RM>
<RM relation="ATTRIBUTE">
    <AI name="parent" value="meta$LogEvent"/>
    <AI name="auxiliary:testCaseFile" value="default.txt"/>
    <AI name="id" value="meta$LogEvent$timestamp"/>
    <AI name="name" value="timestamp"/>
    <AI name="type" value="String"/>
</RM>
<RM relation="ATTRIBUTE">
    <AI name="parent" value="meta$LogEvent"/>
    <AI name="auxiliary:testCaseFile" value="default.txt"/>
    <AI name="id" value="meta$LogEvent$priority"/>
    <AI name="name" value="priority"/>
    <AI name="userType" value="meta$LogPriority"/>
</RM>
<RM relation="ATTRIBUTE">
    <AI name="parent" value="meta$LogEvent"/>
    <AI name="id" value="meta$LogEvent$rtServiceID"/>
    <AI name="name" value="rtServiceID"/>
    <AI name="type" value="String"/>
    <AI name="value" value="Self.meta$LogEvent$RelatedTo.meta$RTService$identifier"/>
```

```
</RM>
<RM relation="ATTRIBUTE">
    <AI name="parent" value="meta$LogEvent"/>
    <AI name="auxiliary:testCaseFile" value="default.txt"/>
    <AI name="id" value="meta$LogEvent$message"/>
    <AI name="name" value="message"/>
    <AI name="type" value="String"/>
</RM>
<RM relation="ATTRIBUTE">
    <AI name="parent" value="meta$LogEvent"/>
    <AI name="auxiliary:testCaseFile" value="default.txt"/>
    <AI name="id" value="meta$LogEvent$throwable"/>
    <AI name="name" value="throwable"/>
    <AI name="type" value="Text"/>
</RM>
<RM relation="RELATIONSHIP">
    <AI name="parent" value="meta$LogEvent"/>
    <AI name="auxiliary:testCaseCount" value="20"/>
    <AI name="graphmetadata:go" value="meta$LogEvent$RelatedTo_go"/>
    <AI name="id" value="meta$LogEvent$RelatedTo"/>
    <AI name="inverse" value="meta$RTService$LoggedEvents"/>
    <AI name="maxCard" value="1"/>
    <AI name="minCard" value="1"/>
    <AI name="name" value="LogEvent_RTService"/>
    <AI name="roleName" value="RelatedTo"/>
    <AI name="to" value="meta$RTService"/>
</RM>
<RM relation="ENTITY">
    <AI name="parent" value="MetaStructure"/>
    <AI name="auxiliary:attributesVisible" value="false"/>
    <AI name="auxiliary:testCaseCount" value="20"/>
    <AI name="graphmetadata:go" value="meta$RTService_go"/>
    <AI name="id" value="meta$RTService"/>
    <AI name="name" value="RTService"/>
</RM>
<RM relation="ATTRIBUTE">
    <AI name="parent" value="meta$RTService"/>
    <AI name="auxiliary:testCaseFile" value="default.txt"/>
    <AI name="id" value="meta$RTService$identifier"/>
    <AI name="name" value="identifier"/>
    <AI name="type" value="String"/>
</RM>
<RM relation="ATTRIBUTE">
    <AI name="parent" value="meta$RTService"/>
    <AI name="auxiliary:testCaseFile" value="default.txt"/>
    <AI name="id" value="meta$RTService$type"/>
    <AI name="name" value="type"/>
    <AI name="userType" value="meta$RTServiceType"/>
</RM>
<RM relation="ATTRIBUTE">
    <AI name="parent" value="meta$RTService"/>
    <AI name="auxiliary:testCaseFile" value="default.txt"/>
    <AI name="id" value="meta$RTService$hitCount"/>
    <AI name="name" value="hitCount"/>
    <AI name="type" value="Integer"/>
</RM>
<RM relation="RELATIONSHIP">
    <AI name="parent" value="meta$RTService"/>
```

```
        <AI name="auxiliary:testCaseCount" value="20"/>
        <AI name="id" value="meta$RTService$LoggedEvents"/>
        <AI name="inverse" value="meta$LogEvent$RelatedTo"/>
        <AI name="maxCard" value="N"/>
        <AI name="minCard" value="0"/>
        <AI name="name" value="LogEvent_RTService"/>
        <AI name="roleName" value="LoggedEvents"/>
        <AI name="to" value="meta$LogEvent"/>
    </RM>
    <RM relation="Navigation">
        <AI name="parent" value="gen-N400001"/>
        <AI name="id" value="gen-N40013F"/>
    </RM>
    <RM relation="GLOBALPARAMETER">
        <AI name="parent" value="gen-N40013F"/>
        <AI name="duration" value="session"/>
        <AI name="entity" value="User"/>
        <AI name="id" value="UserCtxParam"/>
        <AI name="name" value="UserCtxParam"/>
    </RM>
    <RM relation="GLOBALPARAMETER">
        <AI name="parent" value="gen-N40013F"/>
        <AI name="duration" value="session"/>
        <AI name="entity" value="Group"/>
        <AI name="id" value="GroupCtxParam"/>
        <AI name="name" value="GroupCtxParam"/>
    </RM>
    <RM relation="Mapping">
        <AI name="parent" value="gen-N400001"/>
        <AI name="id" value="gen-N40014E"/>
    </RM>
    <RM relation="rdbms:RDBMSMapping">
        <AI name="parent" value="gen-N40014E"/>
        <AI name="id" value="gen-N400150"/>
    </RM>
    <RM relation="auxiliary:GraphMetaData">
        <AI name="parent" value="gen-N400001"/>
        <AI name="id" value="gen-N400154"/>
    </RM>
    <RM relation="graphmetadata:Drawing">
        <AI name="parent" value="gen-N400154"/>
        <AI name="element" value="Structure"/>
        <AI name="id" value="Structure_go"/>
        <AI name="scale" value="1.0"/>
        <AI name="x" value="-116.0"/>
        <AI name="y" value="-314.5"/>
    </RM>
    <RM relation="graphmetadata:Node">
        <AI name="parent" value="gen-N400154"/>
        <AI name="element" value="User"/>
        <AI name="id" value="User_go"/>
        <AI name="x" value="-382.0"/>
        <AI name="y" value="-370.5"/>
    </RM>
    <RM relation="graphmetadata:Node">
        <AI name="parent" value="gen-N400154"/>
        <AI name="element" value="Group"/>
        <AI name="id" value="Group_go"/>
        <AI name="x" value="-295.0"/>
        <AI name="y" value="-377.0"/>
    </RM>
    <RM relation="graphmetadata:Node">
        <AI name="parent" value="gen-N400154"/>
        <AI name="element" value="SiteView"/>
        <AI name="id" value="SiteView_go"/>
        <AI name="x" value="-214.5"/>
        <AI name="y" value="-377.0"/>
    </RM>
    <RM relation="EntityPos">
        <AI name="element" value="ENT_ID"/>
        <AI name="pos" value="ent1_go"/>
        <AI name="xValue" value="-379.0"/>
        <AI name="yValue" value="-282.0"/>
    </RM>
    <RM relation="graphmetadata:Connection">
        <AI name="parent" value="gen-N400154"/>
        <AI name="element" value="User2Group"/>
        <AI name="id" value="User2Group_go"/>
        <AI name="x" value="-340.5"/>
        <AI name="y" value="-353.5"/>
    </RM>
    <RM relation="graphmetadata:Connection">
        <AI name="parent" value="gen-N400154"/>
        <AI name="element" value="User2DefaultGroup"/>
        <AI name="id" value="User2DefaultGroup_go"/>
        <AI name="x" value="-341.0"/>
        <AI name="y" value="-401.0"/>
    </RM>
    <RM relation="graphmetadata:Connection">
        <AI name="parent" value="gen-N400154"/>
        <AI name="element" value="Group2SiteView"/>
        <AI name="id" value="Group2SiteView_go"/>
    </RM>
    <RM relation="auxiliary:ProjectDependentOptions">
        <AI name="parent" value="gen-N400001"/>
        <AI name="id" value="gen-N400188"/>
    </RM>
    <RM relation="auxiliary:Option">
        <AI name="parent" value="gen-N400188"/>
        <AI name="id" value="gen-N40018A"/>
        <AI name="name" value="SHOW_CARDINALITY"/>
        <AI name="type" value="BOOLEAN"/>
        <AI name="value" value="false"/>
    </RM>
    <RM relation="auxiliary:Option">
        <AI name="parent" value="gen-N400188"/>
        <AI name="id" value="gen-N40018F"/>
        <AI name="name" value="SHOW_MASTER_OBJECT"/>
        <AI name="type" value="BOOLEAN"/>
        <AI name="value" value="true"/>
    </RM>
    <RM relation="auxiliary:Option">
        <AI name="parent" value="gen-N400188"/>
        <AI name="id" value="gen-N400194"/>
        <AI name="name" value="SHOW_ROLES"/>
        <AI name="type" value="BOOLEAN"/>
        <AI name="value" value="true"/>
    </RM>
    <RM relation="auxiliary:Option">
        <AI name="parent" value="gen-N400188"/>
        <AI name="id" value="gen-N400199"/>
        <AI name="name" value="SHOW_PARAMETER_LINK_SYMBOL"/>
        <AI name="type" value="BOOLEAN"/>
        <AI name="value" value="true"/>
    </RM>
    <RM relation="auxiliary:Option">
        <AI name="parent" value="gen-N400188"/>
        <AI name="id" value="gen-N40019E"/>
        <AI name="name" value="SHOW_CARDINALITY_UML_STYLE"/>
        <AI name="type" value="BOOLEAN"/>
        <AI name="value" value="true"/>
    </RM>
    <RM relation="auxiliary:Option">
        <AI name="parent" value="gen-N400188"/>
        <AI name="id" value="gen-N4001A3"/>
        <AI name="name" value="SHOW_ATTRIBUTES_INSIDE_ENTITIES"/>
        <AI name="type" value="BOOLEAN"/>
        <AI name="value" value="true"/>
    </RM>
</LR>
```

```
let $schemaIns :=   input()/LRI,
    $relMems :=  $schemaIns/RM,
    $rAttribute := $relMems[@relation eq 'Attribute'],
    $rEntity := $relMems[@relation eq 'Entity'],
    $rEntityPos := $relMems[@relation eq 'EntityPos'],
    $uPagePos :=    distinct-values($relMems[@relation eq 'PagePos']/AI[@name eq
                     'pos']/@value),
    $uSiteView :=   distinct-values($relMems[@relation eq 'SiteView']/AI[@name eq
                    'siteView']/@value union
            $relMems[@relation eq 'Page']/AI[@name eq 'definedAt']/@value union
            $relMems[@relation eq 'CreateUnit']/AI[@name eq 'definedAt']/@value union
            $relMems[@relation eq 'DeleteUnit']/AI[@name eq 'definedAt']/@value union
            $relMems[@relation eq 'ModifyUnit']/AI[@name eq 'definedAt']/@value),
    $uAttribute := distinct-values($relMems[@relation eq 'Attribute']/AI[@name eq
        'attribute']/@value),
    $uXPosValue := distinct-values($relMems[@relation eq 'EntityPos']/AI[@name eq
            'xValue']/@value union
            $relMems[@relation eq 'RelRolePos']/AI[@name eq 'xValue']/@value union
            $relMems[@relation eq 'PagePos']/AI[@name eq 'xValue']/@value union
            $relMems[@relation eq 'DataUnitPos']/AI[@name eq 'xValue']/@value union
            $relMems[@relation eq 'IndexUnitPos']/AI[@name eq 'xValue']/@value union
            $relMems[@relation eq 'EntryUnitPos']/AI[@name eq 'xValue']/@value union
            $relMems[@relation eq 'ScrollerUnitPos']/AI[@name eq 'xValue']/@value union
            $relMems[@relation eq 'CreateUnitPos']/AI[@name eq 'xValue']/@value union
            $relMems[@relation eq 'DeleteUnitPos']/AI[@name eq 'xValue']/@value union
            $relMems[@relation eq 'ModifyUnitPos']/AI[@name eq 'xValue']/@value union
            $relMems[@relation eq 'LinkPos']/AI[@name eq 'xValue']/@value),
    $uIndexUnitPos :=   distinct-values($relMems[@relation eq 'IndexUnitPos']/AI[@name eq
            'pos']/@value),
    $uEntity := distinct-values($relMems[@relation eq 'Entity']/AI[@name eq 'entity']/@value union
            $relMems[@relation eq 'Attribute']/AI[@name eq 'definedAt']/@value union
            $relMems[@relation eq 'RelRole']/AI[@name eq 'to']/@value union
            $relMems[@relation eq 'RelRole']/AI[@name eq 'from']/@value union
            $relMems[@relation eq 'EntityPos']/AI[@name eq 'element']/@value union
            $relMems[@relation eq 'DataUnit']/AI[@name eq 'entity']/@value union
            $relMems[@relation eq 'IndexUnit']/AI[@name eq 'entity']/@value union
            $relMems[@relation eq 'ScrollerUnit']/AI[@name eq 'entity']/@value union
            $relMems[@relation eq 'CreateUnit']/AI[@name eq 'entity']/@value union
            $relMems[@relation eq 'DeleteUnit']/AI[@name eq 'entity']/@value union
            $relMems[@relation eq 'ModifyUnit']/AI[@name eq 'entity']/@value),
    $uEntityPos := distinct-values($relMems[@relation eq 'EntityPos']/AI[@name eq 'pos']/@value),
    $uIndexUnit :=  distinct-values($relMems[@relation eq 'Link']/AI[@name eq
                'destIndexUnit']/@value union
            $relMems[@relation eq 'Link']/AI[@name eq 'startIndexUnit']/@value union
            $relMems[@relation eq 'IndexUnit']/AI[@name eq 'indexUnit']/@value union
            $relMems[@relation eq 'IndexUnitPos']/AI[@name eq 'element']/@value),
    $uName := distinct-values($relMems[@relation eq 'Entity']/AI[@name eq 'name']/@value union
            $relMems[@relation eq 'Attribute']/AI[@name eq 'name']/@value union
            $relMems[@relation eq 'RelRole']/AI[@name eq 'relShipName']/@value union
            $relMems[@relation eq 'RelRole']/AI[@name eq 'name']/@value union
            $relMems[@relation eq 'SiteView']/AI[@name eq 'name']/@value union
            $relMems[@relation eq 'Page']/AI[@name eq 'name']/@value union
            $relMems[@relation eq 'Link']/AI[@name eq 'name']/@value union
            $relMems[@relation eq 'DataUnit']/AI[@name eq 'name']/@value union
            $relMems[@relation eq 'IndexUnit']/AI[@name eq 'name']/@value union
            $relMems[@relation eq 'EntryUnit']/AI[@name eq 'name']/@value union
            $relMems[@relation eq 'ScrollerUnit']/AI[@name eq 'name']/@value union
            $relMems[@relation eq 'CreateUnit']/AI[@name eq 'name']/@value union
        $relMems[@relation eq 'DeleteUnit']/AI[@name eq 'name']/@value union
        $relMems[@relation eq 'ModifyUnit']/AI[@name eq 'name']/@value),
$uYPosValue := distinct-values($relMems[@relation eq 'EntityPos']/AI[@name eq
        'yValue']/@value union
        $relMems[@relation eq 'RelRolePos']/AI[@name eq 'yValue']/@value union
        $relMems[@relation eq 'PagePos']/AI[@name eq 'yValue']/@value union
        $relMems[@relation eq 'DataUnitPos']/AI[@name eq 'yValue']/@value union
        $relMems[@relation eq 'IndexUnitPos']/AI[@name eq 'yValue']/@value union
        $relMems[@relation eq 'EntryUnitPos']/AI[@name eq 'yValue']/@value union
        $relMems[@relation eq 'ScrollerUnitPos']/AI[@name eq 'yValue']/@value union
        $relMems[@relation eq 'CreateUnitPos']/AI[@name eq 'yValue']/@value union
        $relMems[@relation eq 'DeleteUnitPos']/AI[@name eq 'yValue']/@value union
        $relMems[@relation eq 'ModifyUnitPos']/AI[@name eq 'yValue']/@value union
        $relMems[@relation eq 'LinkPos']/AI[@name eq 'yValue']/@value),
$uPage :=     distinct-values($relMems[@relation eq 'Page']/AI[@name eq 'page']/@value union
        $relMems[@relation eq 'Link']/AI[@name eq 'destPage']/@value union
        $relMems[@relation eq 'Link']/AI[@name eq 'startPage']/@value union
        $relMems[@relation eq 'DataUnit']/AI[@name eq 'definedAt']/@value union
        $relMems[@relation eq 'IndexUnit']/AI[@name eq 'definedAt']/@value union
        $relMems[@relation eq 'EntryUnit']/AI[@name eq 'definedAt']/@value union
        $relMems[@relation eq 'ScrollerUnit']/AI[@name eq 'definedAt']/@value union
        $relMems[@relation eq 'PagePos']/AI[@name eq 'element']/@value)
let $outerQTRes := (

    let $innerQTRes1 := (

    for $Y_ANCH in $uYPosValue,
        $X_ANCH in $uXPosValue
    where
        (exists(for$rEntityPos_ in $rEntityPos
            where $rEntityPos_/AI[@name eq 'xValue' and @value eq $X_ANCH] and
                $rEntityPos_/AI[@name eq 'yValue' and @value eq $Y_ANCH]
            return $rEntityPos_))
    return <tuple>
            <Y_ANCH>{$Y_ANCH}</Y_ANCH>
            <X_ANCH>{$X_ANCH}</X_ANCH>
        </tuple>
    )

return <outerQTRes>
    {
    for $innerQTRes1_ in $innerQTRes1,
        $ENT in $uName,
        $ENT_ID in $uEntity
    let $Y_ANCH := data($innerQTRes1_/Y_ANCH),
        $X_ANCH := data($innerQTRes1_/X_ANCH)
    where
        (exists(for$rEntity_ in $rEntity
            where $rEntity_/AI[@name eq 'entity' and @value eq $ENT_ID] and
                $rEntity_/AI[@name eq 'name' and @value eq $ENT]
            return $rEntity_)) and
        (exists(for$rAttribute_ in $rAttribute
            where $rAttribute_/AI[@name eq 'definedAt' and @value eq $ENT_ID]
            return $rAttribute_)) and
        (exists(for$rEntityPos_ in $rEntityPos
            where $rEntityPos_/AI[@name eq 'element' and @value eq $ENT_ID] and
                $rEntityPos_/AI[@name eq 'xValue' and @value eq $X_ANCH] and
                $rEntityPos_/AI[@name eq 'yValue' and @value eq $Y_ANCH]
                return $rEntityPos_)) and(
                (($ENT eq 'Author')) or
                (($ENT eq 'Paper'))
            )
        return <tuple>
                <Y_ANCH>{$Y_ANCH}</Y_ANCH>
                <X_ANCH>{$X_ANCH}</X_ANCH>
                <ENT>{$ENT}</ENT>
                <ENT_ID>{$ENT_ID}</ENT_ID>
            </tuple>
        }
        </outerQTRes>

    )

return <LRI>{
let $hPagePos :=    max(   for $id in $uPagePos
            where       starts-with($id, 'page') and ends-with($id, '_go')
            return substring-before(substring-after($id, 'page'), '_go')),
    $hIndexUnitPos :=   max(   for $id in $uIndexUnitPos
            where       starts-with($id, 'inu') and ends-with($id, '_go')
            return substring-before(substring-after($id, 'inu'), '_go')),
    $hIndexUnit :=  max(   for $id in $uIndexUnit
            where       starts-with($id, 'inu')
            return substring-after($id, 'inu')),
    $hPage :=   max(   for $id in $uPage
            where       starts-with($id, 'page')
            return substring-after($id, 'page')),
    $gtRes := (
for $counter in 1 to count($outerQTRes/tuple)
let $ENT_ID :=  data(item-at($outerQTRes/tuple,$counter)/ENT_ID),
    $ENT :=  data(item-at($outerQTRes/tuple,$counter)/ENT),
    $X_ANCH := data(item-at($outerQTRes/tuple,$counter)/X_ANCH),
    $Y_ANCH := data(item-at($outerQTRes/tuple,$counter)/Y_ANCH),
    $SV_ID :=   'sv1',
    $PC_ID :=   concat('page', string(xs:decimal(max((($hPage),(xs:string(0))))) + 1 + 1 * ($counter
                 -1))),
    $PC :=  concat($ENT ,'Page'),
    $IU_ID := concat('inu', string(xs:decimal(max((($hIndexUnit),(xs:string(0))))) + 1 + 1 * ($counter
               -1))),
    $IU := concat($ENT ,'List'),
    $POS_1 :=   concat(concat('inu', string(xs:decimal(max((($hIndexUnitPos),(xs:string(0))))) + 1 +
                1 * ($counter -1))), '_go'),
    $X_VAL_1 :=$X_ANCH + 634.0,
    $Y_VAL_1 :=$Y_ANCH + 749.0,
    $POS_2 :=   concat(concat('page', string(xs:decimal(max((($hPagePos),(xs:string(0))))) + 1 + 1
                * ($counter -1))), '_go'),
    $X_VAL_2 :=$X_ANCH + 634.0,
    $Y_VAL_2 :=$Y_ANCH + 750.25
return <tuple>
        <RM relation="SiteView">
            <AI name="siteView" value="{$SV_ID}"/>
        </RM>
        <RM relation="Page">
            <AI name="page" value="{$PC_ID}"/>
            <AI name="name" value="{$PC}"/>
            <AI name="definedAt" value="{$SV_ID}"/>
        </RM>
        <RM relation="IndexUnit">
            <AI name="indexUnit" value="{$IU_ID}"/>
            <AI name="name" value="{$IU}"/>
            <AI name="entity" value="{$ENT_ID}"/>
            <AI name="definedAt" value="{$PC_ID}"/>
        </RM>
        <RM relation="IndexUnitPos">
            <AI name="pos" value="{$POS_1}"/>
            <AI name="element" value="{$IU_ID}"/>
            <AI name="xValue" value="{$X_VAL_1}"/>
            <AI name="yValue" value="{$Y_VAL_1}"/>
        </RM>
        <RM relation="PagePos">
            <AI name="pos" value="{$POS_2}"/>
            <AI name="element" value="{$PC_ID}"/>
            <AI name="xValue" value="{$X_VAL_2}"/>
            <AI name="yValue" value="{$Y_VAL_2}"/>
        </RM>
    </tuple>)
return $gtRes/RM union $relMems
}</LRI>
```