Karlsruhe Institute of Technology

Software Design and Quality

# Integration of Triple Graph Grammars in Vitruvius

Proposal for a Master's Thesis of

Robin Schulz

At the KIT Department of Informatics
KASTEL – Institute of Information Security and Dependability

| | |
|---|---|
| First examiner: | Prof. Dr. Ralf Reussner |
| Second examiner: | Prof. Dr.-Ing. Anne Koziolek |
| First advisor: | Lars König, M.Sc. |
| Second advisor: | Thomas Weber, M.Sc. |

18 November 2024 – 18 May 2025

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

# Contents

# 1 Introduction

The development of many large modern IT systems and systems with physical aspects to them, which are called cyber-physical systems (CPS) is characterized by the use of different languages and tools to describe different parts and aspects of the system under development. Different modelling and programming languages are used to describe different aspects of the system, e.g. design, implementation and documentation and different sub-domains and sub-systems. The use of different languages produces different informational artifacts describing the system and dependencies between those artifacts.

As an approach to cope with increasingly complex conglomerates of different artifacts that all describe a single system, Model Driven Software Engineering (MDSE) has become a relevant strategy with industrial relevance. Developmental groups of stakeholders to such systems can form by assigning different roles to software developers or by choosing certain architectural patterns or development processes, e.g. software architects, system deployers and component developers. Systems that not only consist of software introduce further groups of developers, e.g. electrical or mechanical engineers, that also produce artifacts that describe the system from physical viewpoints. To account for different perspectives and concerns, in short: viewpoints, that different groups of developers have, the view-based paradigm has been established. There, various views are defined that represent partial information of a system that is relevant from the viewpoint of the roles which different developer groups assume. With the usage of multiple models that constitute a system, those models often share information which introduces redundancy and the question of how to prevent that redundancy or keep redundant information consistent arises. Klare et. al. [14] aim to answer that question by proposing the concept of Virtual Single Underlying Model metamodels (V-SUM metamodels) and presenting the Vitruvius approach. Aiming to combine "the advantages of synthetic and projective modeling", the projective concept of a view as a dynamically generated model is combined with the synthetic concept of describing the system with multiple interrelated models instead of one. Developers only use views to modify the system. Alongside view definition languages, languages which are specialized for consistency preservation play the key role in keeping consistency between the models which form the system description of a V-SUM.

While such languages exist and have been implemented for V-SUMs in the context of Vitruvius, there are also existing and mature [8, 6] general-purpose model-transformation languages which are able to express complex relations between models. One such concept is given by *Triple Graph Grammars* (TGGs) [17], which consist of context-sensitive graph production rule patterns that relate a pair of graphs by building a third graph representing the relation. These rules can be used for keeping two models consistent in an incremental manner via pattern matching.

TGGs allow for specifying descriptive rules that can concern any number of entities and the graph approach allows for graphical vizualization of these, potentially complex, relations. To to use that advantage for consistency preservation in V-SUMs and examine the resulting approach, the following research questions have to be answered:

1. Can TGG rule definitions be used for consistency keeping in V-SUMs?
2. What are the performance impacts of using TGGs to specify consistency relations compared to existing consistency preservation languages?
3. What kinds of consistency relations can be expressed?

To that end, the proposed thesis aims to investigate how TGGs can be used for consistency keeping in Vitruvius by researching how sequences of Vitruvius changes can be converted to sequences of TGG rule applications and vice versa.

In chapter 2 we give an overview over the foundations for this thesis. Chapter 3 outlines the proposed concept which is to be elaborated in the thesis and chapter 4 describes implementation matters to that concept. Chapter 5 shows how the implemented prototype is to be evaluated. In chapter 6 we describe existing approaches to the problem of defining transformations to preserve consistency. Chapter 7 concludes this document by describing organizational matters within and around the thesis.

# 2 Foundations

The proposed thesis aims to evaluate how TGGs can be used as a consistency perservation tool in V-SUMs and to that end, extend the Vitruvius toolset. Thus, this chapter provides an overview over the theoretical and practical foundations of V-SUMs, model based software engineering, model synchronization and Triple Graph Grammars.

## 2.1 Model-Driven Software Engineering

Model-driven software Engineering (MDSE), or sometimes called model-driven software development (MDSD) is an approach to software engineering which Kramer [15] describes as "a development paradigm in which models are used in an automated way for all development tasks". Well-known approaches to that paradigm are the Object Managment Group's Model-Driven Architecture (MDA) approach [16] and the Unified Modelling Language [10], the latter being a language in which models can be described.

Definitions of what a model is are numerous. Stachowiak [18] provides three properties that define a model:

- mapping property: models always are models of *something*, they represent originals which themselves can be models again.
- reduction property: models in general only comprise those attributes of the original that seemed relevant to model users or creators
- pragmatism property: models aren't unambiguously assigned to their originals. They fulfill a purpose for certain subjects, within certain time intervals and with limiation to certain conceptual or real operations.

Czarnecki and Helsen define the term model as "abstractions of a system or its environment, or both" [4]. Using this broad definition seems appropriate, since in the context of model-driven software engineering, program code is considered as models as well as formal descriptions of interaction like agent-based modelling or those of timing constraints (see [10]). A similarly broad definition is given by Caplat and Sourrouille, they define a model as "a representation of a system expressed in a given formalism or language" [3]. This formalism includes an abstract syntax, which is represented by one or multiple concrete syntaxes, and also semantics, which gives meaning to the abstract syntax and induces rules and conditions, sometimes called concrete semantics, which constrain what a well-formed model is and thus further restrict the syntax [9].

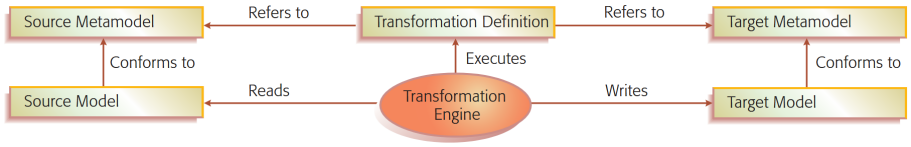


**Figure 2.1:** Schematic example of a model transformation with one source and one target model.[4]

### 2.1.1 Model Transformations

Model transformations play a key role in model-driven software engineering, as they enable specifying and automating relations between models. In general, model transformations map one or multiple source models to one or multiple target model. A transformation engine executes the transformation definition which refers to the sources' and targets' metamodels, which define their abstract syntax. This process, as described by Czarnecki and Helsen [4] is schematically depicted in Figure 2.1 using the example of one source and one target model.

Semantic properties often cannot be distinctly related to a single model if a system consists of multiple models that are developed together; however loosely coupled, a certain *semantic overlap* between those models occurs. Consistency preservation rules (CPRs) in the Vitruvius approach [14] realize some semantic considerations, in that case those about consistency, to relations between models, which are realized by model transformations.

### 2.1.2 Model Synchronization

In order to keep models that have a semantic overlap consistent, a change that occurs in one model has to be propagated to another model, if there are such relations between those models. This synchronization process can be realized with model transformations that have certain features. In their model transformation classification approach, Czarnecki and Helsen identify features that can be used to characterize different approaches to consistency preservation. [4] The incrementality features are necessary properties for model synchronization, while directionalitiy and tracing classify synchronization transformations.

**Incrementality** Incrementality comprises the features target incrementality, source incrementality and preservation of user edits in the target. *Target incrementality* describes whether a transformation is able to propagate changes in the source model to an existing target model. Since consistency preservation requires propagating consistency-breaking changes in one model to another model with which it semantically overlaps, that is a necessary property of consistency preservation transformations. A transformation has the feature of *source incrementality* if it "minimizes the amount of source that needs to be reexamined by a transformation". This property is desireable from a performance point of view and with regard to potential information loss. *Preservation of user edits in the target* describes the ability of a transformation to apply source model changes to the target model while preserving modifications in the target model. A transformation that is target incremental

but doesn't preserve user edits in the target is undesireable for consistency preservation purposes, since it may discard user edits and thus introduce information loss.

**Directionality** If a transformation can be executed in only one direction, it is called *unidirectional*, otherwise *multidirectional*. A *bidirectional* transformation is a kind of multidirectional transformation that can be executed in two directions. In the context of model synchronization, where forward and backward transformations are required if both models are subject to user change, bidirectionaly of a transformation reduces the definition effort of transformations and ensures that the forward transformation does the same as the backward transformation. If those are separate unidirectional transformations, ensuring that they apply the same definition of consistency has to be done elsewise, e.g. by the vigilance of the transformation developer or by deriving the unidirectional transformations from a shared consistency definition, like it is done in the Commonalities Language [13], which can thus be called bidirectional.

**Tracing** is described by Czarnecki and Helsen as a "runtime footprint of transformation execution". It is a relevant property for model synchronization transformations since it eases the identification of what model elements have to be changed in the target model.

## 2.2 Single Underlying Models

Single Underlying Models (SUMs) [1] are an approach to keeping a system consistent by using only one model to represent the system and defining views which are generated via model transformation and are used to access the model. This kind of approach can be called projective, since views are generated by "extraction from an underlying repository" (IEEE1471/ISO42010). This approach has the advantage, that no consistency-keeping has to be done between pairs of models because all information is contained in the SUM.

The profit of total consistency comes with the drawback of a SUM being monolithic, and thus in bigger projects, too complex to handle by the methodologist responsible for creating the metamodel and view transformations. Wanting to keep the benefit of consistency while breaking complexity by using multiple metamodels internally, Klare et. al proposed the concept of a *Virtual Single Underlying Metamodel* V-SUMM, with V-SUMs describing the respective instances of the V-SUMM [14]. A V-SUM internally consists of multiple models but behaves like a SUM to the users who work with the views, since it is also only accessed via views, as shown in Figure 2.2. The model instances of these metamodels that share common information have to be kept consistent via model transformations.

## 2.3 The Vitruvius Approach

As an approach to realize a V-SUM, Klare et al. developed the Vitruvius approach [14]. What exactly is to be kept consistent is abstractly defined by the concept of *consistency*

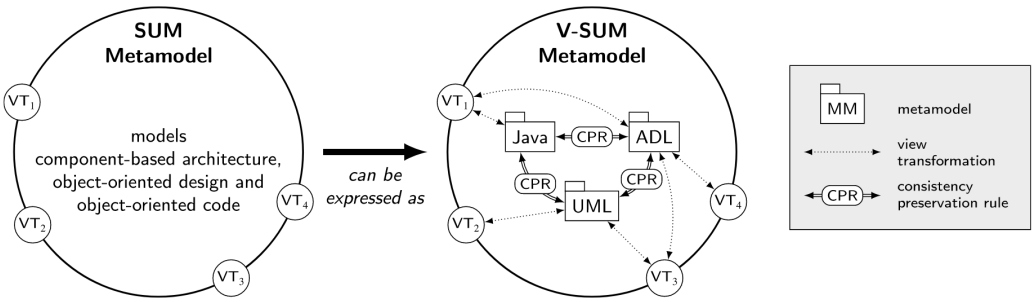


**Figure 2.2:** A V-SUM compared to a SUM. From [14]

*preservation rules*, which is reified by the usage of languages like the Reactions Language and the Commonalities Language. The Vitruvius approach keeps track of consistency relations by letting CPRs use a correspondence model that helps identifying what model elements are to be kept consistent.

### 2.3.1 Change Definition

Changes in Vitruvius are defined via a change metamodel, which itself is defined in Ecore, a meta-metamodel that closely resembles OMG's Essential Meta Object Facility (EMOF), which is supported by the Eclipse Modelling Framework (EMF) as an alternate serialization of Ecore [19, p. 39]. Ecore is supported for representing metamodels in Vitruvius [15].

This change metamodel consists of change meta-classes which concern atomic changes and compound changes which group atomic changes, marking that "these atomic changes occurred together" [15]. These compound changes are not explicitly modeled in Ecore, however, but represented in the implementation.

The Vitruvius change metamodel incorporates a class hierarchy with multiple subconcepts of what a Change can be. The root entity is an *EChange*. Multiple subclasses are additive or subtractive (by inheriting the respective abstract class), representing adding or deleting something from the model. There are kinds of changes for

- EObjects: abstract classes for deletion, creation and existence changing
- attribute and reference changes of EObjects: change classes that concern lists, single- or many-valued attributes and references.
- root EObjects that are not added by referencing another element

To map changes to model entities, the change classes are generically typed with EcoreClasses, which represent the model elements on which changes are referring to. As an example, a change to insert a new value into a many valued attribute is modeled like in Figure 2.4.

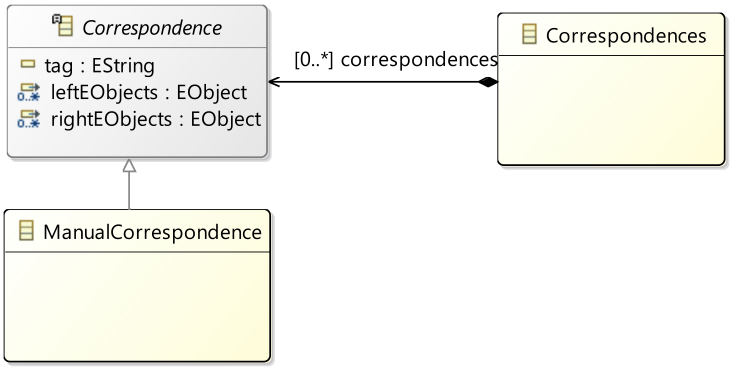


**Figure 2.3:** The VITRUVIUS correspondence model. Generated from the current model definition [20]

```
InsertEAttributeValue<Element extends EJavaObject,
                      Value extends EJavaObject>
    -> InsertInListEChange<Element, EAttribute, Value>,
       AdditiveAttributeEChange<Element, Value>
```

**Figure 2.4:** *InsertEAttributeValue* Signature. InsertEAttributeValue inherits *InsertInListEchange* and *AdditiveAttributeEChange*

### 2.3.2 Correspondence Model

To trace which elements are to be kept consistent, VITRUVIUS uses a so-called *correspondence model*. CPRs use that model to trace relations between elements which are to be kept consistent. This model basically consists of a simple abstract *Correspondence* class which has a tag for storing metadata to "distinguish different correspondences between the same element" [14] and two fields for left and right objects which are mapped to each other by being part of the same entity, as to be seen in Figure 2.3.

## 2.4 Triple Graph Grammars

Schürr[17], who introduced Triple Graph Grammars, describes them as graph grammars or graph rewriting systems, that rewrite three graphs in parallel while explicitly modelling inter-graph relationships in one of the graphs, the so called *correspondence graph*.

**Graph Grammars** or "graph rewriting systems", [17], are sets of rules which are semantically similar to productions known from formal language grammars. These rules work in the following way: An object diagram, called *left-hand side* represents a pattern, which has to be found in a graph so that the rule can be applied. The *right-hand side* of the rule represents how that subgraph looks like after transforming the left-hand side via application of the rule. If all elements on the left-hand side can be identified with an element on the right-hand

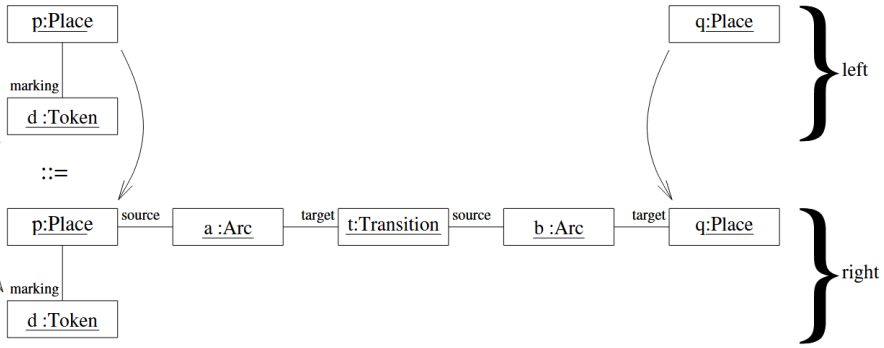


**Figure 2.5:** A non-deleting graph grammar rule. [12]

side, the rule is called a *non-deleting rule*. In such cases, each element on the left-hand side is identified with an element on the right hand side, in Figure 2.5 this is depicted by arrows in between the left- and right-hand sides. For deleting rules, only some elements on the left-hand side can be identified with elements on the right-hand side. To be able to maintain edges between the subgraph which is matched with the left-hand side and the rest of the graph, a context graph or glue graph is identified which contains the elements that are shared between the left-hand and the right-hand side and remain untouched by the transformation [5].

While Graph Grammars are described as "well-suited for the description of complex transformation or inference processes on complex data structures" [17], they are limited to specifying in-place modifications within only one graph.

**Triple Graph Grammars** Triple Graph Grammars (TGGs) were introduced by Schurr [17] to extend this capability to model relationships between two (or more) graphs thus allowing for simultaneous evolution of models while accounting for their relationship. This is achieved by an explicitly modeled correspondence graph and a ruleset which maps one model triplet, consisting of source, target, and correspondence graph to another triplet in a similar way as described for graph grammars in general. The first triplet again represents the pattern to be found in the graph which constitutes from the source, target and correspondence graphs, the left-hand side, and the second triplet represents that subgraph after transformation.

**Model Synchronization with TGGs** This way, TGG rules also can be used to describe what has to be changed in a target graph when a change in the source graph occurs. If such a change is matchable to the left-hand and right-hand side of a TGG *source rule*, meaning the part of the TGG rule that concerns the source graph, the respective target rule can be applied to the target graph. The correspondence graph directly references the subgraph in the target graph on which the target rule has to be applied. That process, from change occurring in the source to applying the matching change in a target graph represents the application of the TGG rule on the triple graph. Since the labelling of the source and target graphs as "source" and "target" is arbitrary, TGG rules can be described as bidirectional synchronization transformations.

Representing the deletion and modification of elements with TGGs can be done by inversely applying TGG rules, by rolling back rule applications, removing the rule application which created the element one wants to delete and re-applying rule applications. Both of these approaches can introduce information loss in the target model if a modification is represented. In the inversion case, a modification would be represented as an inverse application of a TGG rule followed by a re-application of the same rule, but not inversed and with other parameters, or another rule. Since the inversed application deletes model elements in the target model that might be enriched with other model elements that are not covered by the TGG rules, information loss can occur. The same problem occurs with the rollback approach. In certain situations, only inverting a rule application isn't sufficient to represent a deleting change, e.g. when two model elements were generated together, in the sense that both creations are represented by one rule application, and later, only one of these is to be deleted. Then, the one that is not to be deleted, hinders the mere application of the inverse rule. In that case, it either would have to be re-created after inversion, or the rollback approach would need to be done. So both approaches to handling deleting and modifying changes may introduce information loss in the target model Having reviewed different approaches for handling these situations, Fritsche et al. come to the conclusion, that an approach that "avoids unnecessary information loss, is proven to be correct and is efficiently implemented", is still missing [7]. They present short-cut rules [8, 7], which are an approach to cope with the problem of information loss by combining the inversed rule and the other rule. Nodes that would be deleted in the first and re-created in the second, are found out and retained in the left-hand and right-hand side of the resulting short-cut rule. While the original short-cut rules were created at compile time, Fritsche et al. recently presented higher-order short-cut rules [6] which are generated at runtime to cover more rule combinations and thus further decrease information loss in the target model. Short-cut rules can be viewed as an effort to improve incrementality features of TGGs for model synchronization, as described in subsection 2.1.2.

## 2.5 eMoflon

eMoflon [21] is a tool suite for TGGs which implements model generation, transformation and synchronization as well as consistency checking. It encompasses a strategy to allow for information-preserving model changes in many cases, which is non-trivial and reduces rule-writing overhead. This is done by synthesizing so-called short-cut-rules [7, 6] that combine a rule that is to be revoked with a rule that replaces that rule. The resulting rule preserves the entities that would otherwise have been deleted and re-created. Especially if the rule invocation has been done relatively early in the project, this becomes crucial if one doesn't want to lose information in the deleted-and-recreated entities that is not part of the TGG modelling and can thus not be recreated together with the entities.

Correspondences in eMoflon are modeled similarly but different to how Vitruvius models it. In eMoflon, the so-called schema defines a correspondence metamodel between two metamodels whose instances are to be kept consistent. This metamodel is then used in rule

definitions to model the actual correspondences between entities of source and target. The correspondences are modeled similarly to how Vitruvius does it, correspondence types are given a name and source and target EClasses. The difference here lies in the name, which is not an attribute but an alias for the type and might be identifiable with the tag attribute in Vitruvius. Another difference is, that in Vitruvius, the user doesn't pre-define an explicit correspondence metamodel which is used to manage correspondences in the CPRs but instantiates the existing correspondence metamodel, which is shown in Figure 2.3. In the reactions language, described in subsection 6.1.1, this metamodel is part of the language.

# 3 Concept

The proposed thesis is divided into the development of a conversion process between Vitruvius change sequences and TGG pattern application sequences as described in chapter 2, the development of a prototype and its evaluation.

In Vitruvius, all changes are atomic or composite, composite changes being a mere sequence of atomic changes. “atomic” is defined by Klare et al. [14] as affecting “only one element value”. TGG rule pattern applications differ from Vitruvius changes insofar that pattern applications can affect more than one model element and are intended to be to be able to express complex relationships in consistency definitions in the change model.

In the context of utilizing that ability of TGGs for Vitruvius, this gives rise to the question of how those atomic Vitruvius changes can be accurately mapped to complex TGG rule pattern applications, and the question of how TGG rule patterns can be mapped to atomic change sequence templates. The proposed thesis aims to investigate that by researching existing strategies for detecting complex patterns in atomic change sequences and applying them to the given use case and/or developing new strategies. It will also be necessary to develop a process to convert TGG rule applications back to Vitruvius change sequences. Figure 3.1 gives an overview of the conversion process concept. A sequence of atomic Vitruvius changes to a source model, recorded by a change monitor or derived from state differences, which both is existing Vitruvius functionality is given, as well as a source and a target model. The proposed conversion process transforms the given sequence to a sequence of TGG source rule applications. The source rule definitions are derived from TGG rule definitions. The source rule application sequence is used to synchronize the target model. Since TGG source rules map to target rules because both are derived from a single TGG rule, the source rule application sequence can be mapped to a target rule application sequence, which is then used by the TGG engine to synchronize the target model. Also, the target rule application sequence can be transformed to a sequence of atomic Vitruvius changes to the target model, which is not done by the TGG engine but by the proposed conversion process. That also can be used to synchronize the target model.

A strategy for converting atomic change sequences to TGG rule application sequences has to group changes together, so that a pattern may be applicable to the resulting composite change. Especially picking changes out-of-order can be necessary to identify patterns.

**Backward-conversion Pattern Matching** In this thesis, the main concept for solving the problem of generating TGG rule application sequences out of Vitruivus change sequences is to transform the problem to the Vitruvius change space, solve it there. We call this strategy backward-conversion pattern matching. It involves converting TGG patterns to template

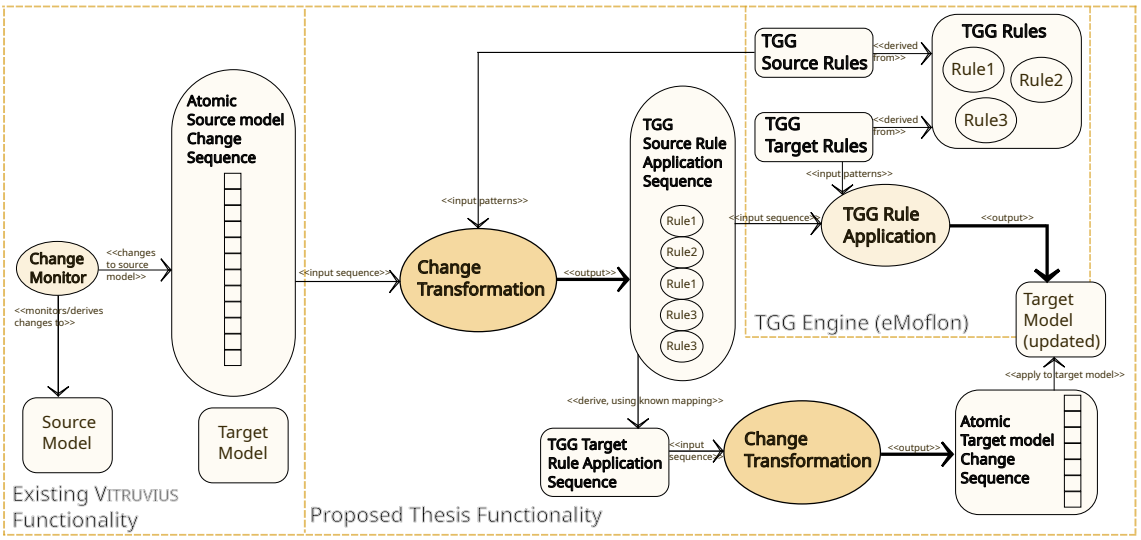


**Figure 3.1:** An Overview of the proposed concept.

sequences of atomic Vitruvius changes and matching those template sequences in a given change sequence given by Vitruvius. The matching is done with respect to the source model giving context to the matcher and using heuristics that rate complex change patterns based on their structure and on the given Vitruvius change sequence. After the sequence has been covered with such templates, the then matched Vitruvius change sequence is converted to a sequence of TGG rule applications. This process is depicted in Figure 3.2.

In general, the *backward conversion* maps TGG changes to a set of permutations of Vitruvius change sequences, because one pattern might be representable by several different permutations of the same set of atomic changes. That has to be taken into consideration. Further the possible permutations will have to be calculated based on interdependencies of the individual atomic changes within a pattern, for example adding a reference to a new object can only be done if the object has been created before, so in a pattern that encompasses both, *createObject* has to be done before *addReference*.

Figure 3.2 shows the workings of that strategy and how atomic changes might need to be picked out-of-order. This introduces the possiblity of conflicts, if several atomic changes whose mapped-to patterns overlap have a dependency, e.g. concern the same model entity. This invalidates a candidate so that the resulting algorithm will have to consider multiple sequence matchings to ensure that no possible rule application is overlooked.

Further, on the situation of having multiple sequence matchings that don't have dependency conflicts, a choice is to be made and to that end, it is favorable to determine which candidate most likely matches what actually was done by the user (who could only have done one or none of the matches). To be able to meet that choice, heuristics have to be applied and possibly developed. Khelladi et al. [11] propose an approach to a similar problem where different heuristics are applied to rate single matches of complex changes in the process of matching. From a performance point of view, this is preferable to calculating candidates and weighing them afterwards. They also formulate three heuristics of which

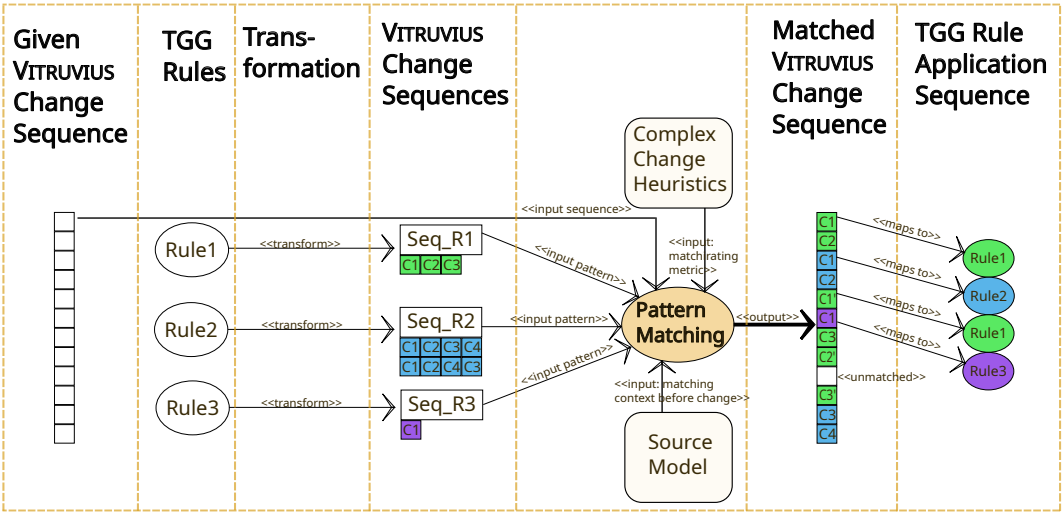


**Figure 3.2: Backward-conversion Pattern Matching:** Converting patterns into (sets of permutations of) change sequences and matching those to a given sequence to retrieve a possible TGG rule application sequence. Some changes don't have to be mapped as they don't concern elements which are deemed to be kept consistent.

two are relatively easily applicable to our scenario, the third, *Containment Level*, which ranks complex changes based on whether it is a superset of other complex changes and the depth of that containment, requires an extension of determining this containment level if it is to be applied in this thesis: Then, an algorithm needs to be developed, which calculates the ranking of

The Backward-conversion pattern matching includes the usage of the mapping of TGG rule patterns to sequences of VITRUVIUS changes. As mentioned before, there might be several possible permutations of such sequences which gives rise to the problem of taking a decision between those in an at least deterministic way.

# 4 Implementation

Based on existing implementations of Vitruvius and a TGG framework, we will implement a prototype.

After researching/developing a strategy how sequences of model changes can be converted to pattern applications, this strategy is to be integrated into Vitruvius. As part of that, and depending on the strategy, a pattern matcher has to be implemented or, if possible, extended.

It is to be found out how an existing TGG framework can be used to represent the TGG end of the implementation. and for the development of TGG rules which are to be applied to changes coming from the Vitruvius framework. This can encompass data structures, metamodels for TGG graphs and rules, algorithms, editors and visualization tools. The *eMoflon* framework implements state-of-the-art model synchronization strategies like short-cut rules [7] and higher-order short-cut-rules [6] from which the integration in Vitruvius would benefit, so this thesis relies on modification and/or usage of existing *eMoflon* engine parts.

A further obstacle might be handling data storage/ integrating the eMoflon project structure and the compatibility of the correspondence models of the eMoflon TGG implementation and the Vitruvius implementation. Both explicitly provide such a model, but a transformation might be necessary. The similarities and differences of both models are discussed in section 2.5, the Vitruvius correspondence model is described in subsection 2.3.2.

It is to be determined how situations, where consistency cannot be restored automatically , can be handled. Such a situation might occur because no matching mapping can be found although changes touch elements which have correspondences. We constrain our approach to not handle such situations and conceptually assume that the methodologist has to specify further rules so that such situations don't occur.

# 5 Evaluation

The purpose of the evaluation of the prototype is to empirically assess the capabilities of the developed apporach to pattern matching of TGG rules on atomic model changes.

Therefore, one concern is to determine the automatization degree in terms of consistency completeness i.e. the expressiveness for consistency conditions, the rule representation complexity, (the avoidance of data loss) and performance of TGG usage on available Vitruvius case studies like those used in the article describing the Vitruvius approach [14] and based on that, compare them with pre-existing consistency preservation languages. To that end, the case studies will have to be modified, existing rules are to be translated to TGG rules. It might be beneficial to extend the case studies so that the assumed advantage of TGG in defining more complex consistency relations can be accounted for.

A second concern, that of scalability is to be evaluated via not a case study but randomly generated data. The idea is to randomly generate models of various sizes and measure the asymptotical runtime performance of the prototype in its potentially different variations.

**GQM Plan** The evaluation is structured by following a Goal Question Metric (GQM) plan, as proposed by Basili et al.[2], to enable transparent methodical verifiability and to explicitly consider construct validity.

- G1 Ensuring correctness of the consistency-keeping process
  - Q1.1 Does the approach identify complex changes correctly?
    - M1.1.1 Number of incorrectly matched patterns in a rule sequence for a given atomic-change-sequence.
- G2 Preserving consistency completeness
  - Q2.1 Can realistic consistency relations be represented by the TGG approach?
    - M2.1.1 Number of rules that can be represented by TGG rules.
- G3 Preserving or improving performance of consistency-keeping
  - Q3.1 How does the TGG approach perform in comparison to existing approaches in (semi-) realistic scenarios
    - M3.1.1 runtime of consistency preservation algorithm on change sequences
  - Q3.2 How does the approach scale in comparison to existing approaches?

M3.2.1 runtime trend of consistency preservation algorithm on change sequences in dependence of input nodes and change sequence length.

# 6 Related Work

Other approaches to model synchronisation are presented in this chapter.

## 6.1 Model Synchronisation Languages

*Domain Specific Languages* (*DSLs*) that can be used for model synchronisation can be subdivided in unidirectional and bidirectional as well as in imperative and declarative approaches. TGGs are bidirectional and declarative, which comes with drawbacks and advantages in comparison to other approaches that have been applied or are applicable in the context of V-SUMs. The Vitruvius project currently uses two languages that are used for consistency preservation between models: the *Reactions* Language and the *Commonalities* Language.

### 6.1.1 Reactions Language

The Reactions language is an imperative and unidirectional language which allows for the definition of reactions that are executed on models when a change in one model that breaks consistency to another model occurs. Its advantage lies in its expressiveness as a "general-purpose programming language"[15]. In comparison to bidirectional approaches to model synchronization, unidirectional approaches have the disadvantage that transformations have to be written for both directions, which makes implementing them more time-consuming and possibly more error-prone. Further, using a graph-based language like TGGs, complex relationships are visualizable, which can prove user-friendly for methodologists that develop CPRs.

### 6.1.2 Commonalities Language

The Commonalities approach [13] is motivated by the idea of making common concepts shared between two metamodels explicit and thus making this redundant information visible. These shared concepts are called *Commonalities* and they are the core entities of a *concept metamodel* whose instances are what the user specifies by using the Commonalities Language.

It thus can be described as a declarative and bidirectional language which, in contrast to other declarative languages used for model synchronization, does not consist of consistency

constraints but of the explicit definition of what information is shared between metamodels. This definition can then be used to derive transformations either between the (then explicitly instantiated) Commonalities metamodel and the metamodels which are to be kept consistent to each other or between those metamodels directly. The Commonalities language implements the first option and generates transformations defined in the Reactions Language.

# 7 Process

## 7.1 Supervision

The thesis is proposed to be reviewed by Prof. Dr. Ralf Reussner, leader of the research group *Dependability of Software-intensive Systems* and Prof. Dr-Ing. Anne Koziolek, both at the KASTEL – Institute of Information Security and Dependability at Karlsruhe Institute of Technology. Advisors will be Lars König, M.Sc. and Thomas Weber, M.Sc.

## 7.2 Artifacts and Resources

**Artifacts** Artifacts will encompass

- An implementation of the prototype, in the form of a code extension of the Vitruivus framework, including
  - transformations and an algorithm for applying the consistency keeping based on TGG patterns
  - embedding of eMoflon
- Possibly code extensions to eMoflon
- Evaluation artifacts:
  - evaluation code for generating randomized test cases
  - modified case studies (with TGG CPRs)
- A report written in LaTeX

**Resources** Resources will include

- the eMoflon framework
- the Vitruvius framework
- a private laptop
- An IDE like eclipse
- pre-existing case studies for Vitruvius

## 7.3 Work packages

The thesis is structured incrementally, The following work packages are defined:

**Work Package 1: Literature research** Perform further literature research on the following topics:

- detecting complex change patterns in atomic change sequences
- heuristics to rate overlapping complex change patterns

**Work Package 2: Preparation**

- Determine interfaces in Vitruvius and eMoflon that are to be used and/or changed
- Define an architecture for the prototype implementing the concept
- Provide basic connection to those interfaces of the frameworks that are required for the prototype.

**Work Package 3: Concept elaboration and simple pattern detection** The first iteration will be to enable the ability to detect patterns in change sequences where all changes belonging to one pattern occur contiguously in the atomic change sequence. This will serve as a minimal working prototype. Heuristics for pattern match rating and subtractive changes need not be included at this point. This work package serves as a basic proof-of-concept. The following is reached after finishing this work package:

- TGG Patterns can be converted into Vitruvius atomic change sequence templates.
- Vitruvius atomic change sequences that fulfill the aforementioned connectedness constraint can be converted into TGG pattern application sequences.
- These TGG pattern application sequences can be converted to atomic change sequences to the target model.
- Atomic change sequences fulfilling the connectedness constraint can be propagated to the target model, enabling consistency keeping in these cases.

**Work Package 4: Uncontiguous pattern detection** This work package concerns improving the pattern matching algorithm. Its is to enable pattern matching where the atomic changes mapping to a pattern are not contiguous.

**Work Package 5: Heuristics inclusion and/or development** Also serving as an improvement to the pattern matcher, this work package includes developing and/or integrating complex change heuristics to rate matches where multiple patterns would match. Also, it is to be ensured that choosing a pattern does not reduce consistency completeness.

**Work Package 6: Include Usage of Short-cut rules** In this work package, it is to be figured out to make use of of short-cut rules and possibly higher-order short-cut rules, which are implemented in the eMoflon framework.

**Work Package 7: Evaluation** The following is done to complete the evaluation work package

- Prepare Evaluation by
  - Extending the case studies
  - Writing the evaluation code
- Conduct case-study based evaluation
- Conduct randomized scalability evaluation

**Work Package 8: Writing** This work package contains writing down what has been done in the thesis, including literature research, the elaborated concept, the prototype's implementation and evaluation.

## 7.4 Risk Management

**Risk 1: Underestimating complexity of the proposed concept** The proposed backward-conversion pattern matching strategy might prove to be more complex and thus time-consuming than estimated. This risk is mitigated by designing the work process iterative. That includes keeping the initial work package that concerns implementing the prototype (Work Package 3) as short as possible and defining the expected result of that package a minimal working protoype, while desirable properties like uncontiguous pattern detection and short-cut rules are put into separate work packages, sorted by dependence and importance. These later work packages can be dropped if time runs short.

**Risk 2: Underestimated complexity of work packages in general** To mitigate the risk of work packages proving to be more complex and/or time-consuming than expected, a buffer of one month is reserved at the end of the thesis processing period, see section 7.5.

**Risk 3: Problems with the environment** Working with the Vitruvius framework, the eclipse IDE and the eMoflon framework might introduce time overhead, since documentation is rare and complexity is high. Mitigating this risk is also done partly by the buffer, partly by explicitly reserving time for preparation (Work Package 1) and partly by frequent communication with the advisor, a weekly meeting is planned.

## 7.5 Time Scheduling

The time schedule for when to perform the planned work packages from section 7.3 is shown in Figure 7.1.

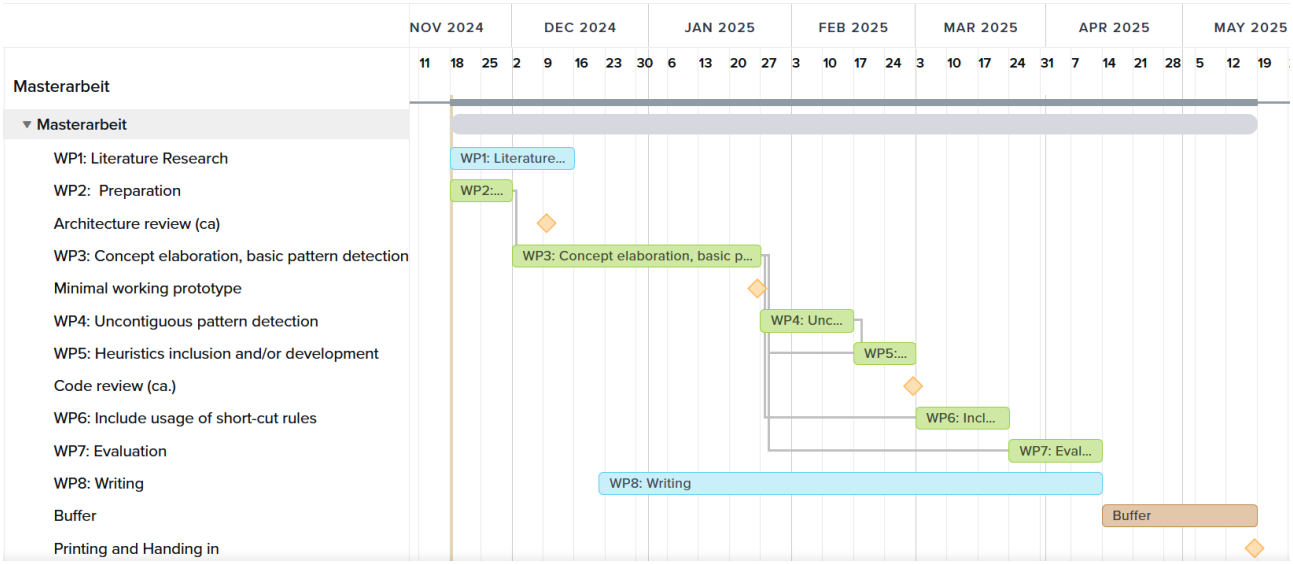


**Figure 7.1:** Coarse-grained time schedule for the thesis (Gantt chart)

# Bibliography


[1] Colin Atkinson, Dietmar Stoll, and Philipp Bostan. "Orthographic Software Modeling: A Practical Approach to View-Based Development". en. In: *Evaluation of Novel Approaches to Software Engineering*. Ed. by Leszek A. Maciaszek, César González-Pérez, and Stefan Jablonski. Berlin, Heidelberg: Springer, 2010, pp. 206–219. ISBN: 978-3-642-14819-4. DOI: 10.1007/978-3-642-14819-4_15.

[2] Victor R. Basili and David M. Weiss. "A Methodology for Collecting Valid Software Engineering Data". In: *IEEE Transactions on Software Engineering* SE-10.6 (Nov. 1984). Conference Name: IEEE Transactions on Software Engineering, pp. 728–738. ISSN: 1939-3520. DOI: 10.1109/TSE.1984.5010301. URL: https://ieeexplore.ieee.org/document/5010301?arnumber=5010301 (visited on 11/03/2024).

[3] Guy Caplat and Jean Louis Sourrouille. "Model mapping in MDA". In: *Workshop in Software Model Engineering (WISME2002)*. Vol. 196. Citeseer, 2002. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=59fec4d1eb7f618eaf1edbd10d3de2 (visited on 09/24/2024).

[4] K. Czarnecki and S. Helsen. "Feature-based survey of model transformation approaches". en. In: *IBM Systems Journal* 45.3 (2006), pp. 621–645. ISSN: 0018-8670. DOI: 10.1147/sj.453.0621. URL: http://ieeexplore.ieee.org/document/5386627/ (visited on 09/20/2024).

[5] Hartmut Ehrig, Annegret Habel, and Hans-Jörg Kreowski. "Introduction to graph grammars with applications to semantic networks". en. In: *Computers & Mathematics with Applications* 23.6-9 (Mar. 1992), pp. 557–572. ISSN: 08981221. DOI: 10.1016/0898-1221(92)90124-Z. URL: https://linkinghub.elsevier.com/retrieve/pii/089812219290124Z (visited on 10/30/2024).

[6] Lars Fritsche et al. "Advanced Consistency Restoration with Higher-Order Short-Cut Rules". en. In: *Graph Transformation*. Ed. by Maribel Fernández and Christopher M. Poskitt. Cham: Springer Nature Switzerland, 2023, pp. 184–203. ISBN: 978-3-031-36709-0. DOI: 10.1007/978-3-031-36709-0_10.

[7] Lars Fritsche et al. "Avoiding unnecessary information loss: correct and efficient model synchronization based on triple graph grammars". en. In: *International Journal on Software Tools for Technology Transfer* 23.3 (June 2021), pp. 335–368. ISSN: 1433-2787. DOI: 10.1007/s10009-020-00588-7. URL: https://doi.org/10.1007/s10009-020-00588-7 (visited on 09/14/2024).

[8] Lars Fritsche et al. "Short-Cut Rules". en. In: *Software Technologies: Applications and Foundations*. Ed. by Manuel Mazzara, Iulian Ober, and Gwen Salaün. Cham: Springer International Publishing, 2018, pp. 415–430. ISBN: 978-3-030-04771-9. DOI: 10.1007/978-3-030-04771-9_30.

[9] David Harel and Bernhard Rumpe. *Modeling languages: Syntax, semantics and all that stuff*. Tech. rep. Citeseer, 2000. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=072663058126978f4b6478121c71de30e404160c (visited on 09/24/2024).

[10] *https://www.omg.org/spec/UML/2.5.1/PDF*. URL: https://www.omg.org/spec/UML/2.5.1/PDF (visited on 09/24/2024).

[11] Djamel Eddine Khelladi et al. "Detecting Complex Changes During Metamodel Evolution". en. In: *Advanced Information Systems Engineering*. Ed. by Jelena Zdravkovic, Marite Kirikova, and Paul Johannesson. Cham: Springer International Publishing, 2015, pp. 263–278. ISBN: 978-3-319-19069-3. DOI: 10.1007/978-3-319-19069-3_17.

[12] Ekkart Kindler and Robert Wagner. "Triple Graph Grammars: Concepts, Extensions, Implementations, and Application Scenarios". en. In: ().

[13] Heiko Klare and Joshua Gleitze. "Commonalities for Preserving Consistency of Multiple Models". In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. Sept. 2019, pp. 371–378. DOI: 10.1109/MODELS-C.2019.00058. URL: https://ieeexplore.ieee.org/document/8904619/?arnumber=8904619 (visited on 09/17/2024).

[14] Heiko Klare et al. "Enabling consistency in view-based system development — The Vitruvius approach". In: *Journal of Systems and Software* 171 (2021), p. 110815. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2020.110815. URL: https://www.sciencedirect.com/science/article/pii/S0164121220302144.

[15] Max Emanuel Kramer. *Specification Languages for Preserving Consistency between Models of Different Languages*. de. 2017. DOI: 10.5445/IR/1000069284. URL: https://publikationen.bibliothek.kit.edu/1000069284 (visited on 09/14/2024).

[16] *OMG Document – ormsc/14-06-01 (MDA Guide revision 2.0)*. URL: http://www.omg.org/cgi-bin/doc?ormsc/14-06-01 (visited on 09/23/2024).

[17] Andy Schürr. "Specification of graph translators with triple graph grammars". en. In: *Graph-Theoretic Concepts in Computer Science*. Ed. by Ernst W. Mayr, Gunther Schmidt, and Gottfried Tinhofer. Berlin, Heidelberg: Springer, 1995, pp. 151–163. ISBN: 978-3-540-49183-5. DOI: 10.1007/3-540-59071-4_45.

[18] Herbert Stachowiak. *Allgemeine Modelltheorie*. de. Springer, 1973. ISBN: 978-3-211-81106-1.

[19] Dave Steinberg et al. *EMF: eclipse modeling framework*. Pearson Education, 2008. URL: https://books.google.de/books?hl=en&lr=&id=sA0zOZuDXhgC&oi=fnd&pg=PT20&dq=EMF:+Eclipse+Modeling+Framework+Steinberg&ots=2LJKXVXkIp&sig=uUG9wKW8bSbFFP47IpWYJ1mUwmQ (visited on 09/27/2024).

[20] *Vitruv-Change/bundles/tools.vitruv.change.correspondence/metamodel/correspondence.ecore at main · vitruv-tools/Vitruv-Change*. en. URL: https://github.com/vitruv-tools/Vitruv-Change/blob/main/bundles/tools.vitruv.change.correspondence/metamodel/correspondence.ecore (visited on 10/09/2024).

[21] Nils Weidmann et al. "Incremental Bidirectional Model Transformation with eMoflon::IBeX". en. In: ().