

Article

Extending Drag-and-Drop Actions-Based Model-to-Model Transformations with Natural Language Processing

Paulius Danenas ¹, Tomas Skersys ^{1,2,*} and Rimantas Butleris ^{1,2}

¹ Center of Information Systems Design Technologies, Kaunas University of Technology, 51368 Kaunas, Lithuania; paulius.danenas@ktu.lt (P.D.); rimantas.butleris@ktu.lt (R.B.)

² Department of Information Systems, Kaunas University of Technology, Studentu str. 50-309, 51368 Kaunas, Lithuania

* Correspondence: tomas.skersys@ktu.lt

Received: 10 August 2020; Accepted: 25 September 2020; Published: 29 September 2020



Featured Application: The main application of the developed solution lies in the area of systems design, where model-to-model transformations are being utilized to their full extent. The presented extension enhances the previously developed solution with certain natural language processing capabilities delivering more features and flexibility. The current version of the complete solution can be applied to model transformations based on Unified Modeling Language (UML) or any other modeling language implemented as a UML profile.

Abstract: Model-to-model (M2M) transformations are among the key components of model-driven development, enabling a certain level of automation in the process of developing models. The developed solution of using drag-and-drop actions-based M2M transformations contributes to this purpose by providing a flexible, reusable, customizable, and relatively easy-to-use transformation method and tool support. The solution uses model-based transformation specifications triggered by user-initiated drag-and-drop actions within the model deployed in a computer-aided software engineering (CASE) tool environment. The transformations are called partial M2M transformations, meaning that a specific user-defined fragment of the source model is being transformed into a specific fragment of the target model and not running the whole model-level transformation. In this paper, in particular, we present the main aspects of the developed extension to that M2M transformation method, delivering a set of natural language processing (NLP) techniques on both the conceptual and implementation level. The paper addresses relevant developments and topics in the field of natural language processing and presents a set of operators that can be used to satisfy the needs of advanced textual preprocessing in the scope of M2M transformations. Also in this paper, we describe the extensions to the previous M2M transformation metamodel necessary for enabling the solution's NLP-related capabilities. The usability and actual benefits of the proposed extension are introduced by presenting a set of specific partial M2M transformation use cases where natural language processing provides actual solutions to previously unsolvable situations when using the previous M2M transformation development.

Keywords: model-to-model transformation; M2M transformation; model-driven development of M2M transformation; natural language processing; NLP; UML profile; CASE tool

1. Introduction

With the introduction of model driven architecture (MDA) by the object management group (OMG) in 2003, model-to-model (M2M) transformations have become an essential part of the modern

model-driven development paradigm. An additional impulse was given by the introduction of unified modeling language (UML) 2.0, featuring a powerful model extension mechanism. With the adoption of UML 2.0, M2M transformations have become a common means of generating platform-specific models from platform-independent models, which in turn made M2M transformations a highly desirable feature in any advanced computer-aided software engineering (CASE) tool supporting MDA principles. Also, M2M transformations have become a part of common practice in other relevant fields of system engineering, such as model-based systems engineering (MBSE).

Currently, M2M transformations are being extensively researched and applied in numerous system engineering contexts. One can find various approaches utilizing visual modeling environments to describe elements of source and target models and their metamodel-level relationships and mappings [1,2], or explicitly specifying transformation rules using formal language, such as QVT [3] or ATL [4]. At the same time, the natural language processing (NLP) field has been booming recently with state-of-the-art developments that were mostly influenced by breakthroughs in the field of deep learning [5,6]. They enable advanced context-driven detection of grammatical and semantic inconsistencies, extraction of relations or entities, tagging of particular subjects of interest in the text, identification of synonyms, and detection of duplicity. All of this requires fundamental knowledge of multiple techniques coming from the fields of information extraction, computational linguistics, ontology engineering, and machine learning. We argue that the application of such techniques in the context of M2M transformation has not been researched and represented enough. This could be explained by the complexity of practical implementation, as well as the lack of actual integration of advanced NLP techniques with textual and visual modeling languages. Such an observation can be made after reviewing several recent M2M transformation research papers focusing on issues related to formal specification, verification, validation, or representation rather than extension of or integration with other research domains, such as NLP [7–10]. The complexity of transformation development, the steep learning curve, and the struggle to learn from examples were identified as some of the key difficulties in practical M2M transformation development and application [11]. The increasing complexity of existing transformation approaches and their implementation can be seen as yet another significant obstacle limiting their extension with additional capabilities from other domains.

While model-driven development has successfully adopted so-called controlled natural language for synchronizing graphical models and their textual representations in the form of object process language [12] or tested the semantics of business vocabulary and rules (SBVR) as an alternative [13], it still does not exploit any actual NLP capabilities for M2M transformations, which require more advanced text processing of source element names to properly represent the semantics of target element names. This problem is also different from specific text-to-model transformations, which generally target predefined models and are based on text specifications, while more universal M2M transformations may be applied to different kinds of source and target models. As a consequence, the development of an M2M transformation approach that would feature high levels of extensibility, flexibility, conceptual simplicity, and usability by M2M transformation designers as well as ordinary modelers becomes a sophisticated and resource-demanding conceptual and technological issue.

To tackle the aforementioned issues as well as other issues inherent in M2M transformations, we introduced a method for the model-driven development and execution of so-called partial M2M transformation specifications [14]. The idea for this development came from our personal professional experience in system design as well as researching state-of-the-art of M2M transformations and other relevant topics. Our common observation is that an expert modeler prefers to have some “freedom of choice” when it comes to selecting the process of the work, the ability to create a few visual models at a time while reusing others’ concepts, modifying these models on demand, or even developing and executing one’s own M2M transformations dedicated to specific modeling uses. These are just a few common situations where the advantage of partial model-based M2M transformations over full M2M transformations becomes substantial.

When a partial M2M transformation specification is developed, it can be interpreted and executed by the transformation engine within the same environment where the actual visual modeling takes place. Our solution enables us to do that across different meta object facility (MOF)-based modeling languages, such as unified modeling language (UML), semantics of business vocabulary and rules (SBVR) [15,16], business process model and notation (BPMN) [17], or service-oriented architecture modeling language (SoaML) [18]. We expect them to apply to other modeling languages (e.g., systems modeling language (SysML) [19] and object-process methodology [12]) and for as long as they are implemented using UML profiles in the modeling environment.

Not being bound to the specific modeling language, our solution can be used within a single level or across different levels of MDA. For example, the developed set of “BPMN process model-to-SBVR business vocabulary” (and vice versa) transformations would be used within the MDA’s business model (or computation-independent model (CIM)), while the transformation set of “SBVR business vocabulary-to-UML conceptual data model” would be applied when transiting between MDA’s CIM and platform-independent model (PIM). The partial M2M transformation approach enables one to perform specific transformations whenever and wherever needed, provided one has a set of developed transformations required to do the partial M2M transformation tasks. In [14], we conducted an extensive experiment and surveyed system design experts to evaluate the usability, expressiveness, understandability, conciseness, and other relevant characteristics of such an approach in the course of performing actual modeling activities with UML, BPMN, and SBVR languages. The acquired results showed high evaluation ratings across the majority of the evaluated characteristics, reassuring us about the relevance and practicality of our development.

Nevertheless, our personal experience showed that the original solution [14] lacked certain crucial capabilities required for the development of more complex transformation models relying on NLP. Particularly, this concerned the identification and processing of text expressions representing the names of model concepts that are involved in those transformations. To introduce such a capability to our method, it had to be enhanced with several NLP features, including the extraction of noun and verb phrases, as well as text and semantic normalization capabilities. Moreover, we also found out that users of custom domain-specific languages (DSLs) might also benefit from being able to detect and extract specific entities or their classes, such as geographic locations, people’s names, or organizations, which is also solved using modern NLP tools. Our newest extension introduces all of these advanced features directly into the visual modeling environment.

Overall, the main benefits of the original M2M transformation solution [14] can be generalized into the following capabilities:

- It supports one-to-one, one-to-many, many-to-one, and many-to-many concept mappings. A set of defined concept types of a source model (or their properties) can be mapped to a set of defined concept types of a target model (or their properties).
- It supports consistency of concept mappings within a single model or across multiple models based on UML or UML profiles. In the M2M transformation specifications, one can specify concept mappings within a single model or among different models expressed using the same or different modeling languages, including UML, BPMN, SBVR, SoaML, and others implemented as UML profiles.
- It supports reuse of M2M transformations. Out of the set of defined transformation specifications, a particular transformation can be enacted by the transformation engine depending on the conditions of the actual situation related to the source model elements and/or drag-and-drop actions performed. This increases the reuse of libraries of transformation specifications across multiple domains.
- It assures traceability between the source model elements and the transformed target model elements [20].

The NLP extension introduced in this paper augments this list with the following characteristics:

- It enables conditional transformations, which allow us to specify and perform more advanced transformation scenarios based on different conditions or constraints.
- It enhances tolerance to various concept-naming conventions used in the source model when generating valid results: Advanced natural language-based parsing of expressions provides an additional means to acquire valid results (i.e., properly named target concepts) compared to our previously developed M2M transformation solution without NLP support. If required, one may specify and perform conditional processing depending on different, sometimes poor, concept naming conventions, e.g., one can properly process use case concepts named by both a verb + noun (e.g., “Issue invoice”) and a single verb or noun (e.g., “Issue”, “Invoice”).
- It reduces the level of redundancy in the target model: If required, one may simplify the target model by combining concepts that have identical or synonymous meanings. This is achieved by identifying synonymous forms and abbreviations.

The overall methodological background of the partial M2M transformation approach together with the aforementioned experimentation on the usability, user acceptance, and other relevant criteria was already extensively discussed in our previous research paper [14]. With that being done, our main objective in this paper was to present the main conceptual and engineering aspects of the developed NLP-enhanced extension of the existing partial M2M transformation solution and introduce a set of detailed application use cases on the transformation specification level. This enables us to explicitly represent the aforementioned beneficial capabilities of the NLP-enhanced M2M transformation solution in the scope of model-driven systems engineering.

The paper is structured as follows: Section 2 provides an overview of the related work in intersecting fields. In Section 3, an illustrative example is introduced, explaining the basic scenario of applying partial M2M transformation in the context of system design. In Section 4, we present an extension to the previously developed metamodel of M2M transformation together with the actual implementation details. Section 5 mainly focuses on a set of NLP operations that we brought into our M2M transformation method. Section 6 extensively describes a set of main M2M transformation use cases utilizing new capabilities introduced by the developed NLP extension, while Section 7 presents and discusses the experimental results. Finally, Section 8 offers conclusions and future work insights.

2. Related Work

Despite numerous contributions and discussions in the field of M2M transformations, visual model-based M2M transformations remain very relevant to date. For an extensive evaluation of the existing M2M transformation approaches and their engineering solutions, we refer the reader to a recent survey by Kahani et al. [21], which examines over 60 available tools for M2M transformation development, including many well-known CASE tools, and evaluates their technical, modeling, and other relevant capabilities, including aspects related to the user experience. They also provide a classification of the analyzed tools, grouping them into declarative, imperative, graph-based, and hybrid approaches. Further, the main focus is on the most relevant visual modeling-based tools.

A significant number of existing solutions are based on triple graph grammar formalism [22], including eMoflon [1], EMorF [2], and Henshin [23]. The original paper on eMoflon presents its implementation as an extension for the Enterprise Architect CASE tool and discusses the application of UML profiles. In this respect, it is similar to the basic principles of our approach, in particular regarding its modularity, extensibility, and reusability. EMorF, based on the eclipse modeling framework (EMF), is used for the development of model-based M2M transformations and supports visual modeling using visual editor, palette, and different views. It also provides support for the interactive application of transformation rules on selected concepts or all possible matches; this bears certain similarities to our approach, which also offers an agile approach to selecting individual transformations involving subsets of model concepts. Henshin’s [23] approach is also similar to our approach in its way of visually presenting M2M transformations as transformation rules with left-hand side (LHS) and right-hand side (RHS) elements; also, UML metamodel concepts are used for specification of transformation

models. The aforementioned EMF [24] also followed a similar principle by utilizing ECore metaclasses to specify transformation rules with certain structural constraints. When compared to our approach, the main difference lies in the general use and modularity of the developed transformations. These tools are applied for full-scale M2M transformations; meanwhile, our approach supports agile, selective application of partial M2M transformations throughout multiple projects and application domains. Shippers et al. [25] described an imperative transformation modeling technique based on story-driven modeling (SDM), which leverages UML profiling technology. This technique was later applied to develop the MoTMoT transformation approach, which combines SDM imperative modeling with declarative modeling of TGG rules; the MagicDraw tool was used as a platform to implement this approach [26]. It was proven to be applicable for both M2M modeling and model refactoring tasks [27]. This research showed that the modern CASE tool with UML profiling support can be successfully transformed into a powerful transformation modeling environment, which is also one of the key features of our approach. Moreover, features of conciseness, readability, maintainability, efficiency, and scalability were also addressed [27], which reconfirmed our experimental findings in [14].

Next to model-driven development of M2M transformations, constraint specification is another significant feature in the area of M2M transformations. As shown in this paper, conditional processing is mandatory to enable advanced transformation scenarios. Some available solutions rely on OMG's Object Constraint Language (OCL), which is a dedicated language for the specification of constraints in UML diagrams. It was used in some solutions that relied on UML class diagrams [28–30]. Several more recent works also consider OCL as a base language to formalize or embed constraints in their transformation tools [2,31]. MOLA [32] and VIATRA [33] use custom graphical modeling languages to imperatively define full-scale M2M transformations. Yet, we were not able to identify any extension points required to implement NLP functionality for these languages.

Design patterns in M2M transformations is also a relevant subject of discussion in the modeling community. Starting with the previous version [14], our solution relies heavily on transformation pattern construction. Design pattern-driven development of M2M transformations is supported in [34]. The authors of [35] introduced a taxonomy of design patterns for model transformation formalizing typical cases that arise during transformation specifications; they also identified the use of these patterns in other related research [10]. SPLIT and CONCAT patterns, further introduced in Section 5.1, might be classified under such taxonomy as entity splitting and entity merging patterns, respectively. Section 6 also introduces a set of transformation patterns that are not expected to meet any categorization yet, although similar taxonomies can be created in the future, after NLP integration with M2M transformations becomes more mature.

NLP-extended model transformations are directly influenced by the quality of automated processing and interpretation of textual labels in visual models. Most of the research in this field was carried out specifically in business process modeling [36,37]. Many aspects of this research, however, can be transferred to other models as well, provided those models apply similar naming and modeling practices. While in our research we presented an exemplary transformation pattern to transform UML activities defined using different naming conventions, this task could also be undertaken by using techniques for activity labeling style identification [37] or by extensively utilizing relevant text corpora, lexical resources like WordNet, and linguistic knowledge [38,39]. Pittke et al. [40] proposed a set of activity naming anti-patterns, which were addressed in [41] by introducing the notion of canonicity to define the so-called atomic business process consisting of exactly one action, one business object, and no more than one addition. While our approach tries to apply NLP technology for processing labels, a different route was taken in [41] that, instead of trying to detect naming patterns and preprocess labels automatically, allows us to define conditional transformations for different types of process names. An extensive study of preprocessing textual labels is presented by Pittke in [39]. This includes algorithms for detecting homonyms and synonyms in the context of process models by combining semantic vectors with information obtained from external resources. Arguably, such an approach could be more suitable for model processing than deep learning approaches; however, this assumption

requires additional verification. Matching elements from multiple models can also be solved using several model matching techniques, particularly text-based or similarity-based matching, which is one of the four categories proposed in a recent survey of model matching techniques [42]. Finally, solving semantic interoperability issues should also be considered to ensure proper transfer of concepts from the source model to the target model while taking into account their contextual discrepancies. Again, there are multiple ways to solve these issues, as suggested in the relevant literature [43].

Finally, one must acknowledge the many recent developments in the area of NLP itself. Some of the core and most recent works are discussed in Section 5.2. However, we reviewed only a small part of the most relevant state-of-the-art works in this area. Besides various NLP-related issues listed in Section 5.2, we identify several other fields that are relevant to our future research and may provide additional capabilities for model processing in the context of M2M transformation:

- Error correction is performed to identify and correct typographical, grammatical, and semantic errors (e.g., accidental use of homophones). Multiple approaches are used to solve these issues, including string distances, similarity techniques, n-grams, and rule-based, statistical, or probabilistic techniques [44]. Recent approaches apply statistical machine translation [45] or neural machine translation principles, mostly transformer architecture [46,47]; however, application of the latter techniques might be limited due to the lack of context required for detection.
- Parse tree generation is focused on performing grammatical analysis for the given text using identified parse trees. Dependency parsing tries to extract intra-sentence relationships between words, while constituency tree generation is focused on grammatical processing using context-free grammars [48] or recurrent neural networks [49,50].
- Dependency parsing is used to identify dependencies between entities within the given text. As with most natural language-related tasks, currently, the best performance is obtained by applying deep learning-based techniques, e.g., Stanford's deep bi-affine parser based on bidirectional long-short term memory (Bi-LSTM) network to produce vector representations for part-of-speech (POS) tagging and dependency relations tasks [51].
- Relation extraction is focused on detecting semantic relationship mentions within the given text. Such relationships can be defined between two or more entities and represent particular connections between them. A wide range of techniques are applied for solving this task, including similarity-based [52], supervised [53,54], and multiple deep learning techniques; for extensive surveys of these approaches, one may refer to [55,56].
- Role labeling seeks to determine the roles of entities by performing classification. Its goal essentially is to determine who did what to whom, when, and where [57]. Current advances to solve this task are related to deep learning techniques, e.g., bidirectional LSTM [57].
- Relation classification is focused on selecting proper relations from a predefined set of classes for two given entities. Consider a tuple (Manager, Invoice, write (Manager, Invoice)) consisting of two subjects and the associated verb "write." The presence of the context word "write" justifies that this is related to a person and enforces assigning the class PERSON-ACTION. This is particularly important in the context of our research, as identifying particular types of entities will lead to the generation of different target elements. The state of the art in this research is focused on identifying such relation classes in unstructured text, with bidirectional LSTM [58] or convolutional neural networks (CNNs) [59].

For more information on this subject, we refer the reader to the most recent survey papers [5,6,60].

3. Introductory Example

In this section, we start with a simple yet practical example illustrating the main concept of the developed solution. The main workflow presented in this example is quite similar to the one introduced in our previous paper [14]. In this case, however, we present a situation in which it is necessary to additionally apply NLP to produce valid results (i.e., a fragment of a target model), thus

showing the actual benefit of the developed NLP enhancement. Note that a more comprehensive list of potential use cases for the application of this NLP enhancement is presented in Section 6.

Let us assume that a system analyst has created (or somehow obtained) a UML use case model, which is a part of some system specification (Figure 1, panel 1); this model is represented in the UML use case diagram (Figure 1, panel 3). Having this model, he would then like to use it as a source of knowledge while developing a UML class model expressing a conceptual data model for that business domain. While this conceptual model could be developed manually from scratch, its development could also be automated to some degree by utilizing the model transformation functionality provided by the modeling environment. Our solution enables the user to selectively and intuitively use drag-and-drop actions on certain use case model elements triggering predefined transformation actions to generate a set of one or more related class model elements and represent those elements in the opened UML class diagram (Figure 1, panel 2).

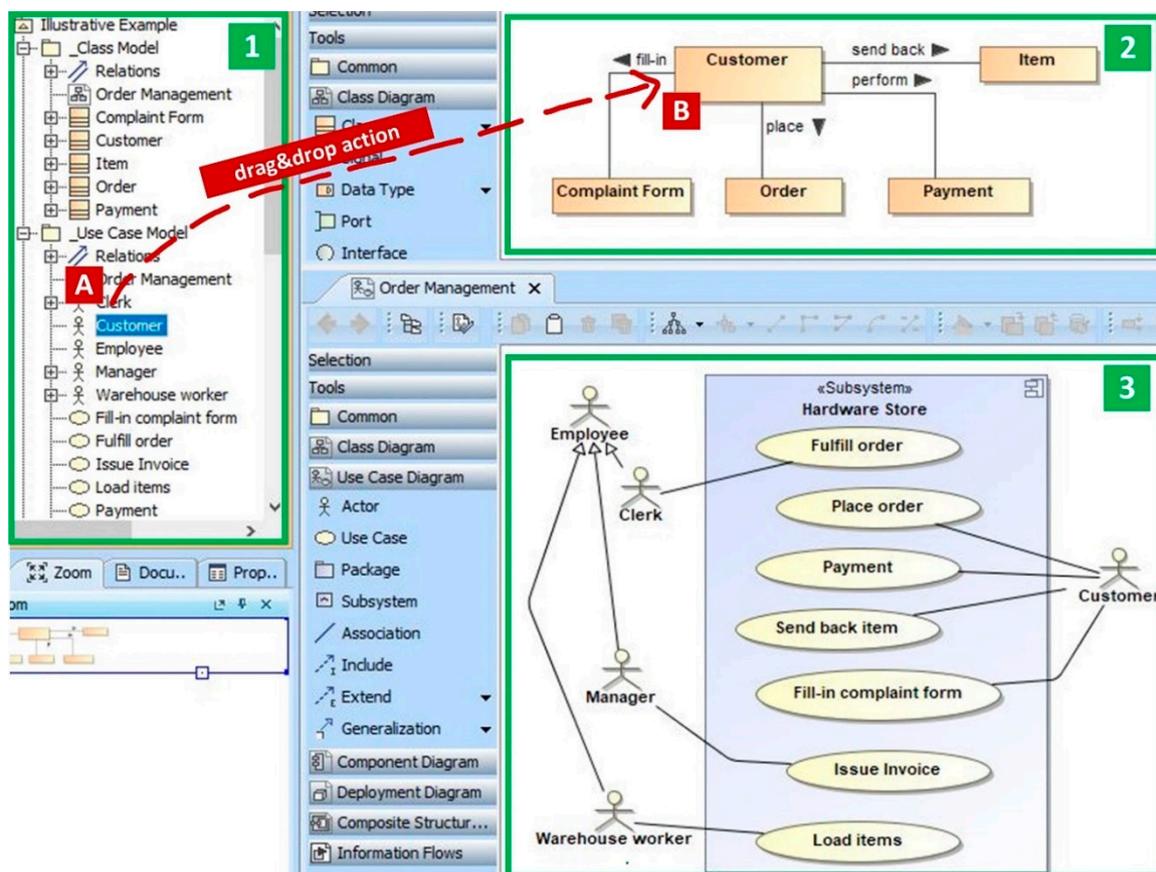


Figure 1. Illustrative example of partial model-to-model (M2M) transformation using drag-and-drop action.

Now, let us be more specific and analyze this particular example of the partial M2M transformation enhanced with NLP (note: while our solution introduces many useful features (e.g., detection and merging of duplicate model elements, assuring traceability between source model elements and transformed model elements), those were described in [14] and other referenced sources and will be considered out of the scope of this paper; instead, we will focus solely on the aspects of the developed NLP extension):

1. We will assume that the user dragged the Actor element “Customer” from the use case model onto the opened class diagram “Order Management” (Figure 1, tag A).

2. This action subsequently triggers a transformation action, which in turn triggers the transformation engine to execute the specific transformation specification visually designed for this action (i.e., dragging an Actor element from the use case model onto the UML class diagram).
3. The transformation specification instructs the transformation engine to select “Customer” together with use cases associated with this actor and transform it into UML classes and a set of associations connecting those classes. In the exemplary use case model, “Customer” is associated with four use cases: “Place order”, “Payment”, “Send back item”, and “Fill-in complaint form”.
4. This chain of performed actions will result in the generation of a fragment of the UML class diagram, as presented in Figure 1, tag B.

At first glance, this would seem like a relatively simple transformation, which was already discussed in our previous paper [14]. However, this particular example illustrates a situation where certain NLP involvement is already required if one wishes to acquire a valid result. The reason behind this is that the conditions defining the extraction of multi-word verb and/or noun phrases are nontrivial. Here are several specific things to consider for this particular transformation:

1. In the resulting UML class diagram, the association between the two classes “Customer” and “Item” is named as the two-word verb phrase “send back”, which was extracted from the source element name, namely the use case “Send back item”. Such extraction requires NLP-enhanced extraction of noun and verb phrases. In our original solution [14], a simple text-processing technique was used. This technique always extracts the first and only the first word from the use case name to be considered as a candidate for the association name in the class model, while the rest of the use case name is transformed into the name of the corresponding class. Such an approach would obviously provide an invalid result in our example presented here, i.e., the association would be named as “send” and the class to which this association is connected to as “Back Item”.
2. In the exemplary use case diagram, we have the use case “Payment”. We intentionally used a bad naming practice here, i.e., the use case is named using only a noun, without any preceding verb or verb phrase. This is arguably one of the most common bad naming practices in use case modeling, which will cause erroneous transformation results if no NLP is involved. In our NLP-enhanced solution, we solve the issue of the single-worded use case name by (1) identifying that there is only a noun or a noun phrase composing the name, which is then transformed into the class with its name equal to that noun or noun phrase; and (2) creating an association and naming it as “perform”, which is a predefined name intended for handling cases where the use of this specific bad naming practice is detected. This particular case of NLP-enhanced processing is presented in more detail in Section 6.3.

Again, these as well as other specific use cases of natural language processing within the scope of partial M2M transformation are presented and discussed in more detail in Section 6.

4. Extending M2M Transformations with NLP

In this section, we describe the main aspects of the NLP extension to the previously developed model-driven M2M transformation based on drag-and-drop actions, which was extensively presented in [14]. First, in Sections 4.1 and 4.2, we present conceptual and implementation aspects of the metamodel extensions, aimed at combining conditional processing-based logic with advanced natural language processing functionality; then, the implementation architecture in the actual CASE tool is presented and briefly discussed in Section 4.3.

Note that to make them more noticeable, the newly introduced conceptual and implementation elements of the NLP extension are distinguished in gray tones in Figures 2–4.

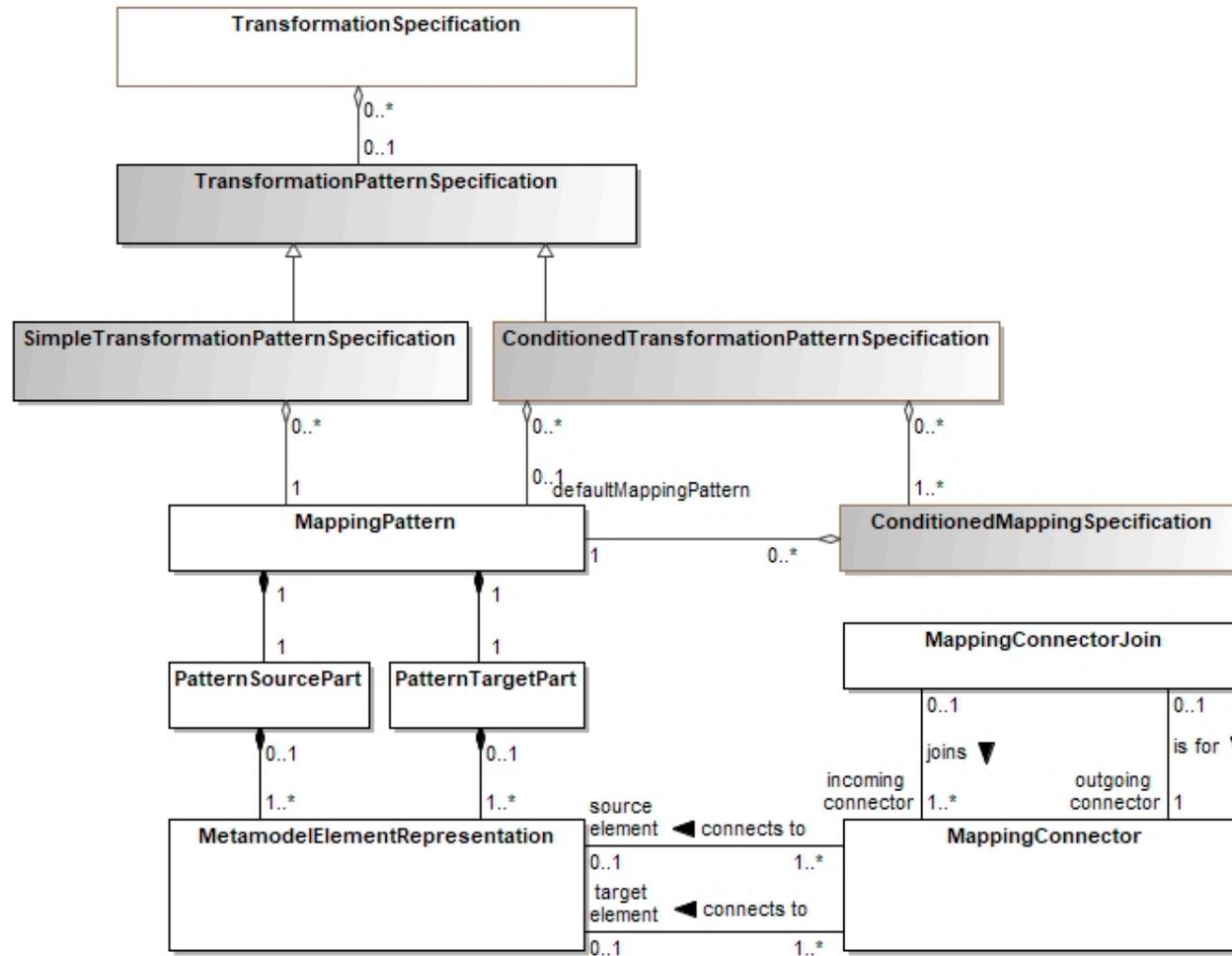


Figure 2. Extended metamodel of model-based M2M transformation.

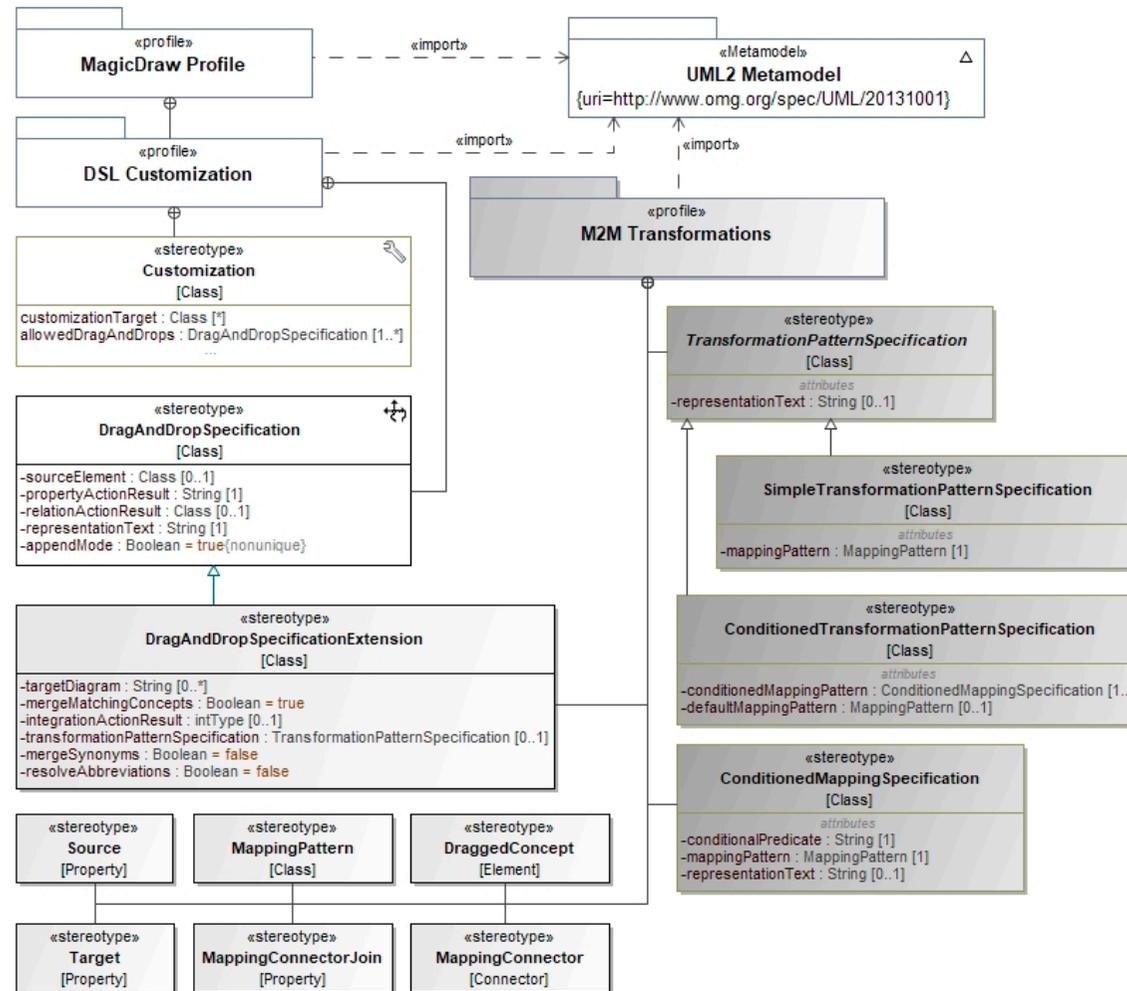


Figure 3. Overall architecture of unified modeling language (UML) profiles supporting the developed solution.

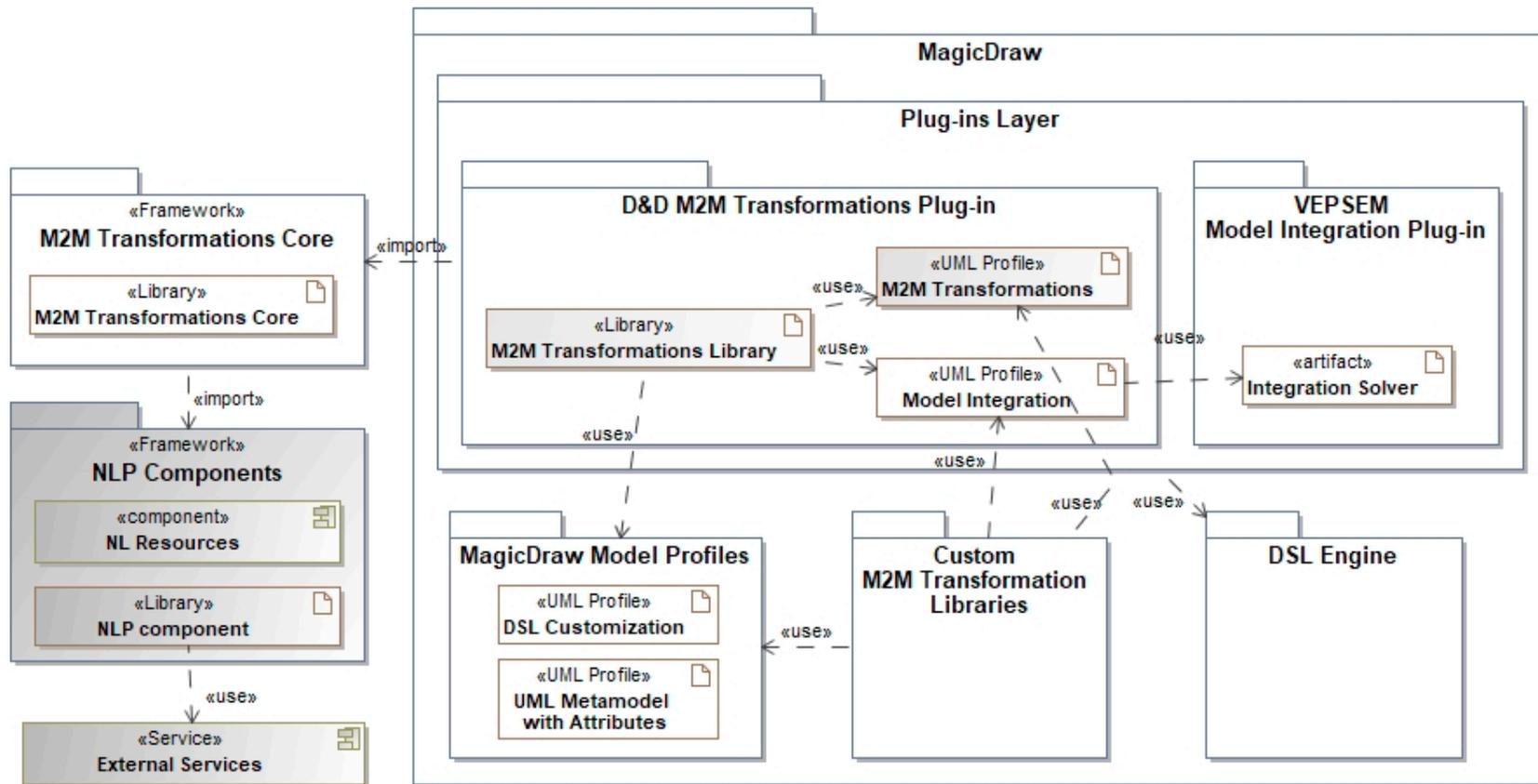


Figure 4. Implementation architecture of extended solution.

4.1. Extended M2M Transformation Metamodel

In this section, we briefly present the core concepts first introduced in [14]; for more information on this subject, refer to that paper. To present definitions and concepts presented in this paper, we will closely follow the notation defined in the previous paper:

- Source model concept type (SMCT); an instance (actual source element) is referred to as I_{SMCT} .
- Target model concept type (TMCT); an instance (actual target element) is referred to as I_{TMCT} .

The implementation-independent metamodel of model-based partial M2M transformations that rely on the specific type of user interaction, i.e., drag-and-drop (D&D) actions, is presented in Figure 2. *TransformationSpecification* defines a set of properties required for atomic “one-to-one” transformations between one SMCT and one TMCT. *TransformationPatternSpecification* is a supplementary component of M2M transformation that enables complex transformations comprising multiple concepts from the source and target models.

The core element of *TransformationPatternSpecification* is a *MappingPattern* composed of two mandatory parts, the *PatternSourcePart* and the *PatternTargetPart*; a complete and valid pattern must contain both parts. The source part contains one or more SMCTs, and the target part one or more TMCTs. The included concept types are mapped using instances of *MappingConnector*. The mapping connector between the particular SMCT and TMCT indicates that upon execution of that M2M transformation specification, the specific instance of TMCT will be created in the target model based on the particular instance of SMCT. In both, the *PatternSourcePart* and *PatternTargetPart* concept types are also interconnected with internal *MappingConnectors* to embed the structure of the underlying source and target metamodels, respectively. *MappingConnectorJoin* is a specialization of the *MappingConnector*, which enables complex mappings of one-to-many and many-to-one types between source and target concept types. Its inverse form is a split, which can be modeled using multiple *MappingConnector* instances with certain conditions, and therefore it is not represented as a separate element in the metamodel. Both join and split will be further discussed in Section 5.1.

As noted in previous sections, NLP extension in our transformation method provides a means to design more complex logic in transformation patterns or their conditional execution based on branching conditions, which often requires specification of certain constraints or predicates to enable such functionality. To introduce such extension, the previous metamodel was extended by adding the following metaclasses (marked as gray in Figure 2):

- *SimpleTransformationPatternSpecification* is a transformation pattern specification (*TransformationPatternSpecification*) containing a single mapping pattern (*MappingPattern*).
- *ConditionedTransformationPatternSpecification* is a transformation pattern specification containing a set of conditioned mapping specifications (*ConditionedMappingSpecification*) together with a default mapping pattern.
- *ConditionedMappingSpecification* is a mapping specification coupling mapping pattern (*MappingPattern*) with conditional expression restricting the execution of the defined mapping pattern based on a specific condition.

Additionally, *TransformationPatternSpecification* was also extended with the following binary properties enabling NLP-based deduplication analysis:

- *mergeSynonyms*: If this property is set to *true*, elements whose names are recognized as synonymous with names of elements already existing in the target model will be automatically merged with those elements by the transformation engine (i.e., new elements will not be created in the target model); otherwise, new elements will or will not be created depending on the *mergeMatchingConcepts* setting, which sets the merging of elements with identical names on and off (straightforward string matching is used here).

- *resolveAbbreviations*: If this property is set to *true*, elements whose names are recognized as abbreviations of names of elements already present in the target model will be automatically merged with those elements by the transformation engine (i.e., new elements will not be created in the target model); otherwise, new elements will or will not be created depending on the *mergeMatchingConcepts* setting.

The actual uses of the aforementioned properties, along with other elements that were extensively described in [14], are presented through actual examples in Section 6.

4.2. Implementing Extended M2M Transformation Metamodel in a CASE Tool

The extended metamodel was implemented in the MagicDraw CASE tool (Figure 3) by extending the previously introduced implementation of the M2M Transformations profile in [14].

At its core, the original solution consists of the native MagicDraw's DSL Customization profile and our developed M2M Transformations profile. In the DSL Customization profile, native «*DragAndDropSpecification*» stereotype is extended with «*DragAndDropSpecificationExtension*» specialization. A model element mapping pattern is realized by a custom stereotype «*MappingPattern*» and implemented as a structured class, holding two parts: A source part (represented using «*Source*» property) and a target part (represented using «*Target*» property). These parts contain one or more SMCTs and TMCTs, respectively, which are interconnected using mapping connectors (*MappingConnector*). These mappings are implemented as UML connectors defined by the «*MappingConnector*» stereotype; connectors may be defined between SMCT elements, TMCT elements, or both elements. In the first two cases, the connectors represent the structure of the underlying source and target metamodels, respectively, while the latter case defines mappings between source and target model elements. For more details on the model-driven specification of M2M transformation patterns, refer to the original paper [14].

Generally, a complete M2M transformation model is composed of a customization class, a D&D action specification, and an optional transformation pattern specification. A customization class relates to one or more D&D action specifications. A D&D action specification is to be executed by the DSL engine upon dragging a certain *sourceElement* specified in that specification and dropping it onto the target element *customizationTarget* specified in the customization class. The customization class will not define any target element if the target is a diagram of the target model itself, that is, the source element is dragged and dropped directly onto the diagram instead of dropping it onto some specific element in the target diagram. The target diagram is specified using the property *targetDiagram*.

Further, Table 1 describes extensions required to enable NLP-based conditional processing. This is enabled by introducing the stereotype «*TransformationPatternSpecification*», which enables selective processing of one or more patterns, along with two of its specializations, and the stereotype «*ConditionedMappingSpecification*».

Table 1. Specification of elements extending the original metamodel implementation.

Stereotype	Description
«TransformationPatternSpecification»	Abstract stereotype containing a single optional <i>representationText</i> property. If specified, this property overrides <i>representationText</i> property in the global drag-and-drop specification. The class has two sub-types defined below.
«SimpleTransformationPatternSpecification»	Specialization of «TransformationPatternSpecification» directly wrapping «MappingPattern» using its single property <i>mappingPattern</i> . This stereotype is generally applied if no conditional branching is required to run a particular transformation between source and target models.
«ConditionedTransformationPatternSpecification»	<p>Specialization of «TransformationPatternSpecification» enabling conditional processing of «MappingPattern» concerning constraints defined for the execution of each pattern. The properties are as follows:</p> <ul style="list-style-type: none"> – <i>conditionedMappingPattern</i>: one or more instances of «ConditionedMappingSpecification» – <i>defaultMappingPattern</i>: specifies «MappingPattern», which will be executed by default if no entry in <i>conditionedMappingPattern</i> meets its conditions for execution. Conceptually this is similar to the default condition in CASE-like statements in programming languages.
«ConditionedMappingSpecification»	<p>Defines the processing of «MappingPattern» per conditional branch and contains the following properties:</p> <ul style="list-style-type: none"> – <i>mappingPattern</i>: «MappingPattern» to be processed and executed – <i>conditionalPredicate</i>: stores a predicate defining conditions under which <i>mappingPattern</i> is to be processed and executed – <i>representationText</i>: description of this mapping specification, which will be shown to a user during the interaction. If specified, <i>representationText</i> property overrides similar <i>representationText</i> property in the global transformation specification or «TransformationPatternSpecification» instance containing this mapping.

4.3. Implementation Architecture

The core of our implementation, along with the requirements for other CASE tools to support the approach, is described in detail in [14]. As stated in [14], the main requirements are support of the UML extension mechanism via UML profiling and extensibility of both the CASE tool functionality and the DSL engine via proper API. *MagicDraw* by No Magic Inc. [61] fully meets these requirements, which was the base factor to select this tool as a platform for the implementation of our M2M transformation method.

The solution architecture, along with the newly introduced extension enabling NLP-based processing, is presented in Figure 4. To enable the NLP functionality, we extended our *M2M Transformations Core* API with additional components for natural language processing (*NLP Components*). As in the initial version, this API fully abstracts M2M transformation processing and can be used to develop similar M2M transformation solutions for other compliant CASE tools. NLP components are also abstracted from specific implementations and include the NLP processing component, which performs defined NLP tasks such as named entity recognition, part-of-speech tagging, or relation classification, together with required natural language resources (such as WordNet thesaurus [62]). Currently, only the English language is supported by NLP operators in the engine. At this stage of development, we also simplified the requirements for resolving synonyms and abbreviations by limiting it mainly to the processing of nouns and noun phrases. For practical implementation, Porter stemmer or lemmatization functionality from a package like Apache OpenNLP [63] can be considered due to their compatibility with our implementation platform. Other frameworks, like Stanford Stanza and Spacy, are being experimented with by utilizing their part-of-speech (POS) tagging and named entity recognition (NER) functionality; as discussed further in Section 7.2, the Stanford Stanza tool provided the best results with NLP-related tasks relevant to our research. Interfaces with supporting external semantic services and linguistic resources, like Linked Data, FrameNet [64], or BabelNet [65], are also considered, but research on their actual applicability is still in progress.

5. NLP Operators for Enhancing Partial M2M Transformations

In this section, we present a set of NLP operators that can be used to develop M2M transformations using our solution. Those are grouped by their purpose and functionality in the following subsections. We also introduce very simple, yet relevant split and concatenation operators that enable atomic many-to-one and one-to-many types of mappings at the element level and ensure that multiple source elements can be mapped or multiple target elements can be generated from any element.

Formally, NLP operators are represented as $OP(SourceElement, TargetElement, parameters)$. Note, however, that a transformation may include multiple source and/or target elements. As those operators are defined for association elements with clear directed relation, we can simplify the previous notation to $OP(parameters)$ by actually omitting the *SourceElement* and *TargetElement* parameters; if parameters are absent, it will simply be referred to as *OP*. If the amount of arguments (*varargs*) is available, we denote them as *params . . .*, similar to Java language notation. Each operator performs specific processing of a source element name or source property name; for simplicity, we will further refer to such operators as source processing operators. Also, we will skip the definitions of generic string operators, such as *REPLACE()*, *SUBSTRING()*, *TRIM()*, *LOWER()*, which provide functionality identical to that defined in modern programming languages that support string processing.

5.1. Split and Concatenation Operators

The essence of concatenation and splitting operators is well-understood and needs no further discussion. Nevertheless, their actual application in our solution holds certain specificity, which goes beyond the common definition. In our case, these operators deal with multiple source/target elements and are visually represented as ternary associations, which consequently requires additional graphical notation extension. This extension is realized as a special type of mapping connection.

CONCAT(strings ...) concatenates multiple source strings into a single target string. A mapping using CONCAT() operator (further denoted as C^{CONCAT}) is represented as an n-ary association with more than one incoming association from source elements. While it deals with multiple strings, it also has to take into account multiple source elements, which can be previously processed using additional string operations. Therefore, its application requires additional variables denoting particular source elements and expressions applied to them; these variables are defined on the corresponding associations in specification between source and target elements.

Visually, C^{CONCAT} is modeled using the *MappingConnectorJoin* element as follows (Figure 5): Let the source element SS_1 have a mapping connector C_1 and the source element SS_2 have a mapping connector C_2 , which are incoming associations in a CONCAT() mapping connector C_1^{CONCAT} . Further, let these mapping connectors have the following variables defined: C_1 : @A; C_2 : @B. These variables may define or use any string processing operators or their combinations. Further, let us define the concatenation of these expressions to a single target string with a space symbol (i.e., ' ') in between those concatenated expressions (note that any other text expression can be used in this place as well). Thus, C_1^{CONCAT} will have the following expression: $C = \text{CONCAT}(@A, ' ', @B)$. The output of C_1^{CONCAT} operation would be an element with the name obtained from processed names of source element SS_1 and SS_2 instances, joined with a space symbol ' '. Intuitively, this operator can be easily generalized for more than two source elements.

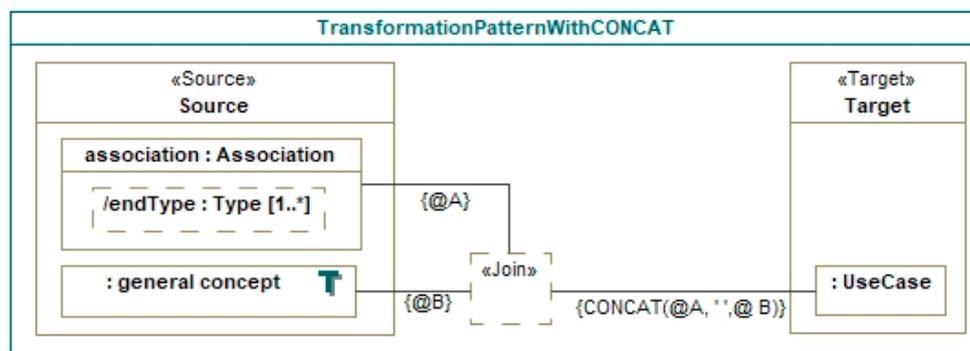


Figure 5. Sample transformation pattern with CONCAT().

Figure 5 presents a sample transformation pattern with CONCAT() operator that defines a transformation between a tuple of SBVR elements *GeneralConcept* and *VerbConcept* (or *Association*) and UML *UseCase* element. The generated use case will be assigned with the name acquired by concatenating two source element names. For example, given an SBVR general concept “order” and a verb concept’s verb expression “place”, the resulting output will be a use case “place order”.

Based on common sense, a mapping using the SPLIT() operator is an inverse to a mapping using the CONCAT() operator, as it defines the generation of more than one target element from a single source element but uses different expressions. It is denoted as C^{SPLIT} and is represented as a ternary association with more than one association outgoing from a single source element and connecting to different target elements. This operator does not require specifying any conditions on the input side, only the expressions on each of the connector parts incoming to the target elements.

The sample transformation pattern incorporating C^{SPLIT} is presented in Figure 6. In that transformation pattern, C^{SPLIT} defines the transformation of the UML *UseCase* element to a tuple of SBVR *GeneralConcept* and *Association* elements. While the whole transformation specification can be recognized as an inverse to the previously defined CONCAT(), it is slightly more complex and involves more elements, i.e., the target *Association* element requires two end elements to form a valid SBVR element (note: in both examples, SBVR *Association* element is specified as UML *Association* element with its internal UML properties and *association* name, which represents the name of a particular stereotype of the SBVR element. This is an implementation aspect; MagicDraw is based on UML profiling, which

enables mapping and representation of elements from other languages (e.g., SBVR) to UML elements. Therefore, in some cases, the design of transformation patterns introduces specifications of properties of UML elements upon which that particular UML profile was built). The *UseCase* element name is used to generate the names of the resulting SBVR *Association* and *GeneralConcept* elements (note that the additional general concept is generated from an *Actor* element associated with the *UseCase* element).

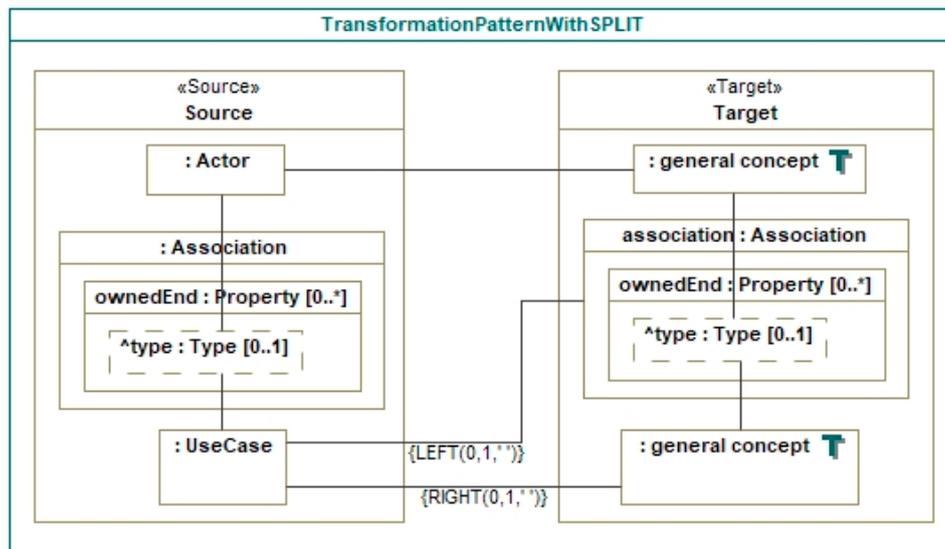


Figure 6. Sample transformation pattern with SPLIT.

The split is modeled as two or more mapping connectors with conditions that define the text fragments to be extracted from the source element name. In our solution, the most basic text processing operators for extracting these text fragments are LEFT() and RIGHT(). These operators are defined as follows:

- LEFT(*from*, *quantity*, *separator*): Extract the *quantity* of tokens starting from the left-hand position *from* (word positions are counted from 0); words are separated using the *separator* string. For example, LEFT(1, 2, ' ') will extract two words starting from the second word in the source element name; words are separated by a white space. If *from* is 0 and *quantity* is undefined, then the whole string will be extracted. By default, it is assumed that white space tokenization will be used unless the separator character is specified as the third parameter.
- RIGHT(*from*, *quantity*, *separator*): Extract the *quantity* of tokens starting from the right-hand position *from* (word positions are counted from 0); words are separated using the *separator* string. For example, RIGHT(, 2, ' ') will extract the last two words from the element name (i.e., the first two words counting from the right). If *from* is 0 and *quantity* is undefined, then the whole string will be extracted.

While the example in Figure 6 demonstrates a simple use of source element name splitting using LEFT() and RIGHT() operators, it is evident that such functions would not be sufficient for many situations, particularly where the length of noun/verb phrases varies (as illustrated in Section 3). Identifying boundaries in such phrases requires more advanced text processing beyond the use of generic substring operators. Therefore, a set of NLP-based operators is introduced further in Section 5.3.

5.2. NLP Techniques for Implementing Advanced Text Processing Operators

To implement more advanced text processing operators (see Section 5.3), we utilized a set of well-established NLP techniques:

- Lexical normalization deals with obtaining the initial word form. Two forms of normalization, namely, stemming and lemmatization, are widely known and applied in text processing. While stemming simply aims to reduce words to their base or root form, lemmatization depends on the part of speech and context and seeks to obtain the base form that is used in a dictionary. Besides dictionary-based lookup, statistical classifiers are used to get lemmas [66,67].
- Tokenization splits the text into separate chunks (tokens). Simple tokenizers generally use predefined separator symbols, like white spaces or commas, to identify limits of such tokens. Real-world texts, however, may contain such characters inside the tokens themselves (abbreviations are one example). This is also different for Asian languages, which may require deeper morphological analysis. Modern toolkits include advanced tokenizers that can handle such problems or deal with specific situations, such as REPP tokenizer [68].
- Part-of-speech (POS) tagging identifies part-of-speech tags for each token in the sentence. This is one of the most researched topics in the natural language processing domain. Most POS tagging solutions usually rely on multiple statistical and machine learning approaches, such as convolutional networks (CNNs) [69], conditional random fields (CRFs) [70,71], or transformation-based learning [72]. This is one of the key techniques used in noun phrase and verb phrase extraction, which is required to process element designations for proper generation of target elements.
- Named entity recognition (NER) focuses on finding entity instances in unstructured text and classifying them into a predefined set of categories, such as a person, organization, location, time, or address. Multiple approaches are available to solve this, including recent advances in deep learning [73,74]. Still, hybrid conditional random field-based approaches seem to dominate in this field [75–77].
- Semantic analysis focuses on capturing synonyms, homonyms, antonyms, and other semantic relations. Lexical databases such as WordNet [62] can be directly applied to solve this task. Finding synonymous forms is one of the most popular tasks in the context of knowledge extraction, and is greatly beneficial for query processing or semantic entity-based searching. Recent developments in deep learning provide advanced techniques to find synonymous or contextually related entries by learning and comparing contextual representations in the form of redistributable embeddings [75,78,79]. These representations can be transferred and reused for different NLP tasks, including POS tagging, document classification, sentiment analysis, and others.
- Hypernym/hyponym discovery enables the extraction of hierarchical relationships to form taxonomies or augment existing ontologies or vocabularies. Rule-based [80–82], vector space [83], neural [84], and hybrid [85] approaches are among the most prominent ones in this category. In the context of our research, semantic analysis can be used to identify and merge synonymous forms and generate generalization relationships or categorizations.

5.3. Advanced Text Processing Operators

To categorize natural language-based text processing operators defined in our approach, we distinguish between text normalization, text recognition, and detection of semantic relationships. While text normalization and recognition both fall under the domain of computational linguistics, text normalization is more focused on obtaining base forms of nouns, verbs, or phrases, whereas recognition operators perform linguistic processing and categorization tasks to obtain particular classes or types of words.

The following operators are defined for text normalization:

- **NORMALIZE(*phrase*)**: Normalizes the given text by converting a plural noun form to a singular form or converting a verb form to the present tense.
- **INFINITIVE(*verb*)**: Gets an infinitive form for the given verb.
- **STEM(*word*)**: Extracts a stem for the given word.

- LEMMA(*word*, *pos*): Gets a lemma for the given word when a particular part of speech (POS) is defined.

The following operators are defined for linguistic text processing:

- POS(*word*): Gets a part of speech for the given word. If more than one part of speech option is identified, only the first one is returned.
- EXTRACTNE(*phrase*, *type*): Extracts named entities (individual concepts) from the given phrase. If *type* is not set, it will try to extract all existing named entities; otherwise, it will try to find entities of the defined type. Currently, *Location*, *Person*, *Organization*, and *Time* entity types are supported in the implementation.
- EXTRACTVERB(*phrase*): Extracts a verb or a verb phrase from the given phrase.
- EXTRACTNOUN(*phrase*, 'all'): Extracts a noun or a noun phrase from the given phrase. Setting the second parameter to 'all' allows one to extract all possible nouns/noun phrases; otherwise, the most general noun phrase is extracted.
- CONTAINS(*phrase₁*, *phrase₂*): Returns *true* if *phrase₁* contains *phrase₂* and *false* otherwise.
- CONTAINSNE(*phrase*, *entity*): Checks whether the given *phrase* contains a particular named *entity*; returns *true* if such entity was found and *false* otherwise.
- TYPENE(*phrase*): Extracts named entities from the given phrase and determines their types. The determined types are returned together with corresponding entities. Currently, 'Location', 'Person', 'Organization', and 'Time' entity types are supported.
- ISNOUNPHRASE(*phrase*): Determines whether the given phrase is a noun phrase; returns *true* if this condition is satisfied and *false* otherwise.
- ISVERBPHRASE(*phrase*): Determines whether the given phrase is a verb phrase; returns *true* if this condition is satisfied and *false* otherwise.

The third group of operators is for semantic relation detection. These operators are defined for the processing of noun phrases only:

- ISSYNONYM(*phrase₁*, *phrase₂*): Returns *true* if *phrase₁* is a synonym of *phrase₂* and *false* otherwise.
- ISHYPONYM(*phrase₁*, *phrase₂*): Returns *true* if *phrase₁* is a hyponym of *phrase₂* and *false* otherwise.
- ISHYPERNYM(*phrase₁*, *phrase₂*): Returns *true* if *phrase₁* is a hypernym of *phrase₂* and *false* otherwise.
- ISMERONYM(*phrase₁*, *phrase₂*): Returns *true* if *phrase₁* is a meronym of *phrase₂* and *false* otherwise.
- ISHOLONYM(*phrase₁*, *phrase₂*): Returns *true* if *phrase₁* is a holonym of *phrase₂* and *false* otherwise.

6. Basic use Cases of M2M Transformation Utilizing NLP Extension Capabilities

To illustrate the application of the presented NLP extension, we present a set of basic M2M transformation use cases requiring NLP assistance and M2M transformation specifications used to resolve those use cases. The use cases were introduced after careful experimentation and observations of our original transformation approach based on drag-and-drop actions, which we applied in real-world working conditions. Note, however, that this is not a finite set of possible use cases (and, correspondingly, transformation specifications); the set may be further augmented by introducing new use cases and corresponding transformation specifications as new modeling situations requiring NLP assistance are discovered.

6.1. Extracting Phrases from the Source Element to Generate the Target Elements

6.1.1. Description and Applicability

The requirement is to transform I_{SMCT} with a heterogeneous multi-word or phrase-based name (or property name) to the I_{TMCT} element (or property) with its name containing only partial expression extracted from the I_{SMCT} name by addressing contextual semantics. The transformation pattern

(Figure 7) used to resolve this use case is an atomic fragment of many other, more complex, transformation patterns as it comprises only one source element and one target element. Also, this pattern is used when only words or phrases of the specific predefined type (e.g., 'Location') need to be extracted from a source element, which is then used to create a target element.

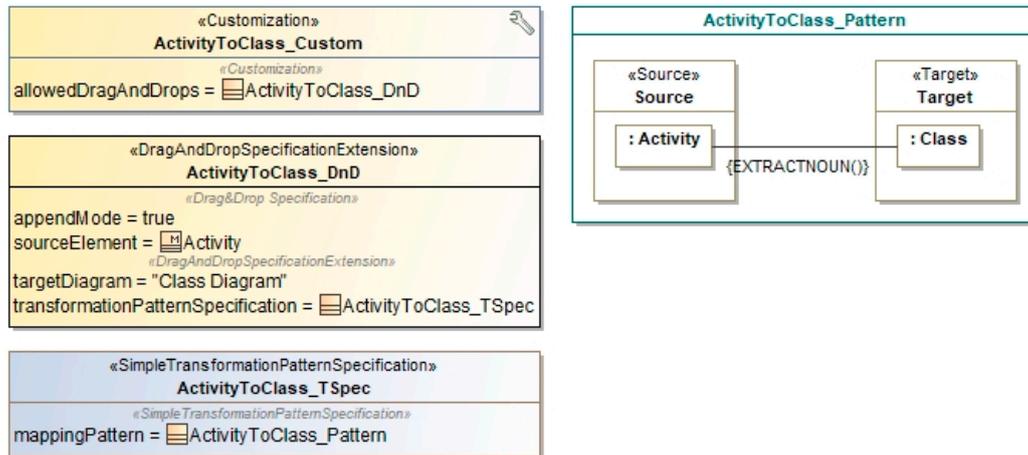


Figure 7. M2M transformation specification for UML Activity to UML Class transformation.

6.1.2. Instance of the use Case Scenario

Let us assume that given a UML Activity element, a user wants to generate a UML Class element representing the extracted noun/noun phrase from that activity name as its name. Note that while this example is about the UML Activity element, this transformation pattern can be easily populated for any specialization of UML Activity, e.g., BPMN Task. NLP extension solves the problem when the names of UML activities contain verb phrases and/or noun phrases comprising different numbers of words. In such cases, the aforementioned string operators LEFT() and RIGHT() would provide inaccurate results.

6.1.3. Transformation Specification for the Presented use Case Instance

The M2M transformation specification is straightforward and requires a single transformation pattern (Figure 7). The name of I_{TMCT} is formed from the output of the EXTRACTNOUN() entity extraction operator, which is specified as a condition of the mapping connector between the respective SMCT and TMCT. Extraction from activities containing more than one named entity will produce a corresponding target element for each recognized entity.

6.1.4. Execution Instance of the Specific use Case Example

Let us consider I_{SMCT}^1 to be a UML Activity element that is being dragged onto the UML class diagram. Then:

- if I_{SMCT}^1 is named as "Issue Invoice" or "Invoice", then the UML class "Invoice" is generated;
- if I_{SMCT}^1 is named as "Issue Invoice to Customer", then the UML classes "Invoice" and "Customer" are generated; and
- if I_{SMCT}^1 does not have any noun/noun phrase in its name, no target elements will be generated.

6.2. Merging Target Elements with Synonymous Meanings

6.2.1. Description and Applicability

The requirement is to avoid adding new duplicate or redundant elements to the target model. A somewhat similar functionality was already enabled in our previous development by specifying the

mergeMatchingConcepts property alone in the D&D specification. However, the detection of synonymous elements was realized by performing a direct comparison of strings, which had its limitations. A more advanced comparison is achieved by applying advanced linguistic analysis, namely, synonymy detection and abbreviation resolution. Thesauri such as WordNet [62] and advanced NLP techniques embracing contextual features, such as representation learning, are used to enable this capability.

6.2.2. Instance of the use Case Scenario

Let us assume that a user needs to transform UML *Class* elements from the conceptual data model to SBVR general concepts by dragging and dropping them onto the diagram representing the SBVR business vocabulary. Names of entity classes are nouns or noun phrases, which makes them ideal for synonymy and abbreviation resolution tasks.

6.2.3. Transformation Specification for the Presented use Case Instance

In this case, the M2M transformation specification does not require any transformation patterns, as it is specified directly in the D&D specification by using the properties *mergeSynonyms* and *resolveAbbreviations* (Figure 8).

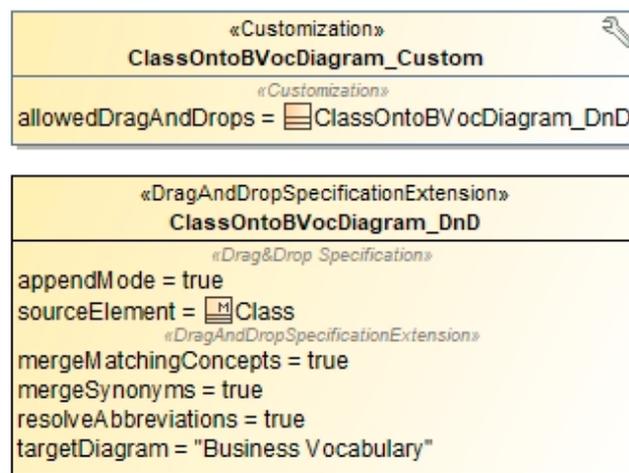


Figure 8. M2M transformation specification for UML *Class* to semantics of business vocabulary and rules (SBVR) *GeneralConcept* transformation.

6.2.4. Execution Instance of the Specific use Case Example

Let us consider a user setting the property *mergeSynonyms* to *true* and I_{SMCT}^1 to be a UML *Class* element that is being dragged onto the SBVR business vocabulary diagram; $I_{TMCT} = \{I_{TMCT}^i\}$, $i = 1 \dots N$ is a set of elements that are already present in this diagram, and I_{TMCT}^i is one of the elements in this set. Then, the following output is expected (note: here and further on in this paper, we assume that the underlying technology is viable to match element names precisely; however, it may differ based on the quality of tools or pretrained models that implement matching analysis or other relevant advanced NLP processing):

- If I_{SMCT}^1 is named as “Employee” and I_{TMCT}^i is named as “Worker”, the user is notified about the already existing matching element and no new element is generated; and
- If I_{SMCT}^1 is named as “Employee” and I_{TMCT}^i is named as “Manager”, no match will be identified, therefore the general concept “Manager” is generated.
- Under the same conditions, let us consider a user setting the property *resolveAbbreviations* to *true*, then the following is possible:

- If I_{SMCT}^1 is named as “Mngr.” and I_{TMCT}^i is named as “Manager”, the user is notified about the already existing matching element and the new element is not generated; the same will happen if the element names are reversed, i.e., I_{SMCT}^1 is named as “Manager” and I_{TMCT}^i is named as “Mngr.”; and
- If I_{SMCT}^1 is named as “Mngr.” and I_{TMCT}^i is named as “Clerk”, the general concept “Clerk” is generated.

6.3. Conditional Processing Addressing Different Element Naming Practices

6.3.1. Description and Applicability

The existence of different practices for naming model elements is one of the main sources of errors when considering extraction from or transformations between models. One can observe many such cases in various modeling-related sources (e.g., academic papers, books, actual projects). Natural language processing can at least partially resolve this issue by enabling the identification of different forms of element names, which then enables automatic selection for the execution an appropriate transformation pattern tailored for that specific form.

6.3.2. Instance of the use Case Scenario

Let us assume that a user transitions from UML to the BPMN model, and in so doing is transforming the selected UML *Activity* elements to the corresponding elements in the BPMN model. Although it is a widely accepted modeling practice to use a verb + noun format for the names of activities and use cases (e.g., “Issue invoice”), one can find numerous cases where nouns and noun phrases alone are used (e.g., “Invoice”, “Customer invoice”), which makes it impossible to develop a transformation that produces valid results by using only common string processing operators. However, by utilizing advanced text processing and conditioning, the right transformation pattern can be executed automatically, selected from the defined set of patterns meeting certain identified conditions (e.g., whether a verb + noun or a noun naming format is used for the activity in question). In this use case scenario, we define the *BPMN Task* element as being generated in all transformation cases; however, the name of that task is formed differently depending on the format of the source element name.

6.3.3. Transformation Specification for the Presented use Case Instance

Transformation specification is implemented using two transformation patterns. In general, atomic one-to-one element transformation does not require any transformation patterns at all, as it can be specified using just *TransformationSpecification* properties. However, to enable conditional processing, it is required to define a transformation pattern per each specific output. In our case, a full transformation model requires two transformation patterns (Figure 9):

- *ActivityToTask_Pattern*, where SMCT is an *Activity* type and TMCT is a *Task* type; and
- *ActivityToTask2_Pattern*, where SMCT is an *Activity* type and TMCT is a *Task* type; additionally, names of the instances of TMCT will be concatenated with a predefined verb “perform” at the beginning of those names.

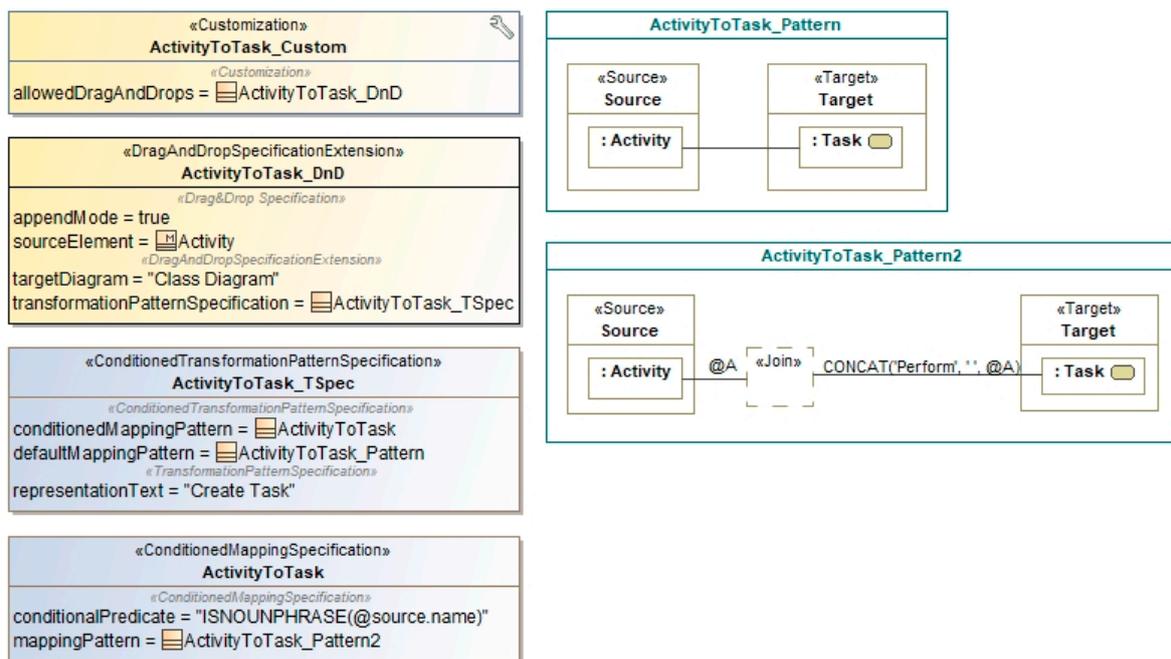


Figure 9. M2M transformation specification for UML *Activity* to business process model and notation (BPMN) *Task* transformation.

Following conditioned processing (*conditionalPredicate* = "ISNOUNPHRASE(@source.name)"), *ActivityToTask_Pattern2* will be selected if the dragged source element name holds only a noun phrase; otherwise, default *ActivityToTask_Pattern* will be selected.

6.3.4. Execution Instance of the Specific use Case Example

Let us consider that I_{SMCT}^1 is a UML activity element that a user drags onto the BPMN process diagram:

- If I_{SMCT}^1 is named as "Issue invoice", then the BPMN task "Issue invoice" is generated; and
- If I_{SMCT}^1 is named as "Payment", then the BPMN task "Perform payment" is generated.

6.4. Conditional Processing Addressing Different Types of Represented Entities

6.4.1. Description and Applicability

Conditional selection of a transformation pattern based on the entity extracted from the source element is arguably one of the most frequent use cases in partial M2M transformation. It can be used in multiple situations with transformation patterns of different complexity.

If SMCT represents a specific noun phrase type, it requires executing a particular transformation pattern from the set of defined transformation patterns based on the recognized type. This is a very common case in conceptual modeling that can be implemented using our solution straightforwardly. This case illustrates the use of named entity recognition and hyponymy determination techniques. This kind of transformation is particularly useful in modeling using custom domain-specific languages (DSLs), which consider specific types of entities as different elements.

6.4.2. Instance of the use Case Scenario

Let us assume that, by using partial M2M transformation functionality, a user wants to create a BPMN *Lane* instance based on the UML *Class* instance, which is dragged and dropped onto the process diagram, provided that class represents an organization or a person, and create a BPMN *DataObject* element if otherwise.

6.4.3. Transformation Specification for the Presented use Case Instance

Even though the whole specification of this partial M2M transformation involves several classes, the overall complexity of this specification remains relatively low (Figure 10).

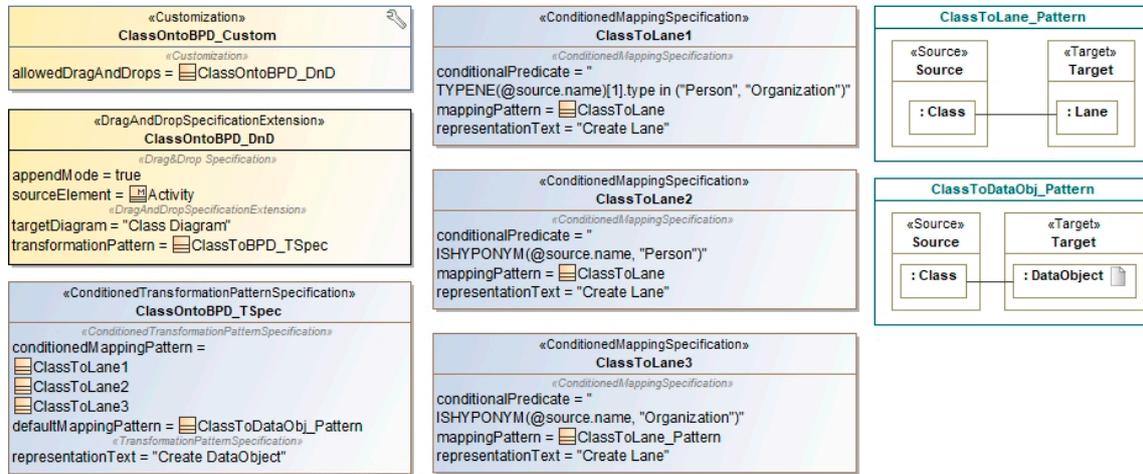


Figure 10. M2M transformation specification for UML *Class* to BPMN *Process model* transformation.

By using NLP techniques for named entity recognition or hyponymy relation extraction, we define two conditions for this transformation:

- Named entities extracted from the name of I_{SMCT}^1 are classified as *Person* or *Organization*;
- Named entities extracted from the name of I_{SMCT}^1 represent a more specific meaning of the *Person* or *Organization*.

Each of these conditions is specified in a separate «*ConditionedMappingSpecification*», and all are coupled with the same transformation pattern “*ClassToLane_Pattern*”. The transformation to UML *DataObject* is specified using the *defaultTransformationPattern* property, indicating that if the aforementioned conditions are not satisfied, the transformation pattern identified in this property will be executed; the latter execution case would result in the generation of a data object based on the dragged and dropped class.

6.4.4. Execution Instance of the Specific use Case Example

Let us consider I_{SMCT}^1 as a UML class that a user drags onto the BPMN process diagram:

- If I_{SMCT}^1 is a UML class named as “*Manager*” and $ISHYPONYM(“Manager”, “Person”)$ is true (i.e., “*Manager*” is identified as a subclass of “*Person*”), then the output of the transformation is a BPMN lane named “*Manager*”.
- If I_{SMCT}^1 is a UML class named as “*Microsoft*” and $ISHYPONYM(“Microsoft”, “Organization”)$ is true (i.e., “*Microsoft*” is identified as a subclass of “*Organization*”), then the output of the transformation is a BPMN lane named “*Microsoft*”.
- If I_{SMCT}^1 is a UML class named “*Invoice*”, which is not a hyponym for either “*Person*” or “*Organization*”, then the output of the transformation is a BPMN data object named “*Invoice*”.

6.5. Resolution of Semantical Relationships: Hyponym/Hypernym, Holonym/Meronym

6.5.1. Description and Applicability

Given two model elements represented in the same diagram, and one is being dragged and dropped onto another one, we can determine the semantic relationship between those elements

and create a generalization or aggregation relationship between them. Hyponym/hypernym and holonym/meronym relationships among identified model elements are detected in this use case.

Note that similar transformations could be developed and used in other models where defined kinds of hierarchical relationships among model elements are present.

6.5.2. Instances of the use Case Scenario

Instances of this use case can be the detection and generation of hierarchical semantic relationships between UML *Class*, *Actor*, *SBVR GeneralConcept* elements, etc. Let us assume that a user drags one class and drops it onto another class, and both classes are deployed in the same UML class diagram. Reacting to this action, the system tries to detect whether a hyponym/hypernym semantic relationship exists between the two classes. If such a relationship is detected, the system automatically creates a generalization relationship between the classes, otherwise a bi-directional association is created.

6.5.3. Transformation Specification for the Presented use Case Instance

Our initial solution considered similar situations by enacting the *relationActionResult* property in the transformation specification; however, the user had to decide on the type of relationship to be generated from the defined set of relationships. The NLP extension enables one to perform automatic conditional processing and consequently decide on the transformation pattern to be executed.

Figure 11 presents the transformation specification incorporating conditional processing with hypernym/hyponym detection. TMCT, representing a “hanging” UML *Class* (note: the “hanging” target element is a target element that has no mapping connection with any source element), represents the element onto which the dragging action is performed. The bi-directional nature of such a semantic relationship requires using two transformation patterns to fully realize the transformation, as it has to be determined whether the source element is a hyponym for the target element or a hypernym. The hypernym/hyponym relationship involves both I_{SMCT} and I_{TMCT} elements; in this context, I_{SMCT} represents the element that was dragged and I_{TMCT} is the element that I_{SMCT} was dropped onto. Hyponym matching must satisfy the predicate $ISHYPONYM(@source.name, @target.name)$; hypernym matching is defined analogously using the $ISHYPERNYM$ operator.

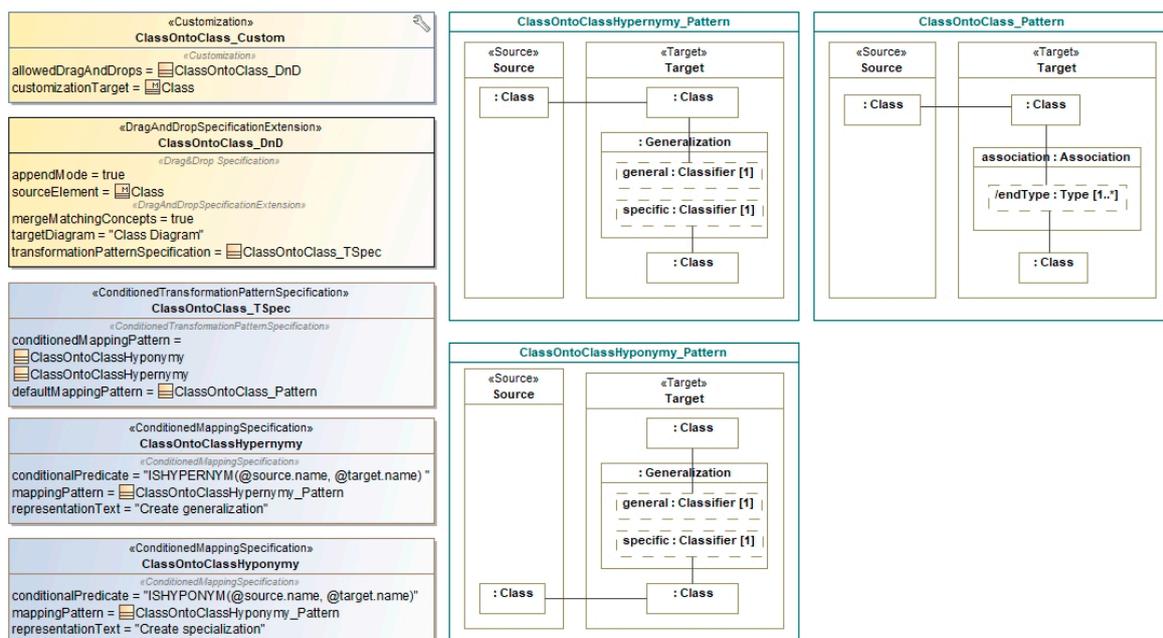


Figure 11. M2M transformation specification example incorporating semantic relationship processing.

6.5.4. Execution Instance of the Specific use Case Example

Consider that I_{SMCT}^1 is a UML *Class* element that a user drags and drops onto another UML *Class* element I_{TMCT}^1 :

- If I_{SMCT}^1 is a class named “Manager” and I_{TMCT}^1 is a class named “Employee”, then the UML *Generalization* relationship is created between the two, where I_{SMCT}^1 is identified as a more specific class (*ClassOntoClassHyponym_Pattern* pattern is executed).
- If I_{SMCT}^1 is a class named “Employee” and I_{TMCT}^1 is a class named “Manager”, then the UML *Generalization* relationship is created between the two, where I_{SMCT}^1 is identified as a more general class (*ClassOntoClassHypernymy_Pattern* pattern is executed).

If I_{SMCT}^1 is a class named “Invoice” and I_{TMCT}^1 is a class named “Manager”, then the UML *Association* relationship is created between the two because there is no hypernym/hyponym relationship detected between the two classes (default *ClassOntoClass_Pattern* pattern is executed).

7. Evaluation

This section presents an experimental evaluation of the proposed approach performed on the real-world dataset of the BPMN process and UML use case models. In Section 7.1, we describe the preliminaries and setup of the experiment, Section 7.2 presents the results obtained during the experiment, and in Section 7.3, we elaborate on the main findings and discuss possible improvements to the presented development.

7.1. Experiment Setup

As a thorough evaluation of the proposed NLP extension requires research and development of multiple tools, our practical experimentation is limited to certain aspects of this complex solution. The main goal of this evaluation is to verify whether the performance of NLP-enhanced partial M2M transformations is superior to partial transformations with no NLP capabilities (i.e., the original solution [14]). To test our hypothesis, transformations were executed in two modes:

1. Applying a simple LEFT(0, 1, ‘ ’) operator (described in Section 5.1), which implements the “first word is a verb” pattern (i.e., “<VERB><NOUN>|<NOUN PHRASE>”); this was done to process all activity-like concept names in the original solution.
2. Applying our proposed NLP extension, which uses NLP operators to recognize and extract the noun and verb phrases composed of any number of words.

In other words, in this experiment, we limited our evaluation to the extraction of noun and verb phrases; these capabilities are utilized in the use cases presented in Section 6.1, Section 6.3, and Section 6.4. Other NLP-related features of the developed NLP extension are out of the scope of this particular experiment simply because they have no analogs in the original solution to be compared with. Other than that, evaluating the other features on their own was also left out of consideration for the following reasons: (1) While abbreviation resolution is discussed in several papers, it is very context-dependent and, in the context of our work, can often be simplified to dictionary-based expansion; and (2) detection of semantic relationships (synonyms, hypernyms, homonyms, and meronyms) can be simplified to performing queries in lexical databases (e.g., WordNet [62]); even though alternative approaches exist (as stated in Section 5.2), so far we have not been successful at finding implementations suitable for practical testing.

For our experimental input dataset, we selected a predefined number of BPMN process models and UML use case models, which were acquired from various available outside sources. The final set of input models consisted of the following:

- 20 UML use case models collected freely from the Internet;

- 20 BPMN process models selected from the large set of Signavio BPMN models provided by the BPM Academic Initiative [86].

The UML class model was selected as the output model of our experiment. Note that it could be safely replaced with the SBVR fact model or virtually any other model being implemented as a UML profile (e.g., our research group has developed such implementation for SBVR [15]). Yet, we chose the UML class model for simplicity, because it is more established and well known in the modeling community.

Next, the acquired set of input models was processed by extracting the names of *Task* elements (for BPMN process models) and *UseCase* elements (for UML use case models) for the actual experimentation. It was considered that the *Task* and *UseCase* elements would contain at least one verb phrase and one noun phrase. The extracted elements were then cleaned of grammatical errors and other inconsistencies. We also excluded entries that would lead to ternary associations in the output results, because this would require more advanced processing to match verb phrases with the corresponding noun phrases, which was not yet considered at this stage of the research. However, entries that had a single verb phrase with multiple noun phrases in conjunctive or disjunctive form were considered for processing, as such structures would be transformed to multiple valid tuples in the output results.

Finally, we applied manual extraction of output results was applied to acquire the expected outputs, which would be considered as the “gold standard” for benchmarking and evaluation. For a more fair comparison of the original solution with the NLP-enhanced development, we also disabled the implemented verb tense normalization feature, which was not implemented in the original approach.

7.2. Selection of NLP Tools for the Experiment

Given the number of possible options for executing our task of extracting noun and verb phrases, current state-of-the-art implementations were thoroughly evaluated to determine their performance in processing model element names. From the initial list of NLP tools, we selected the following for our evaluation:

- Stanford Stanza [87], which uses bi-directional long-short term neural networks (Bi-LSTMs) to implement components and pipelines for solving multiple NLP tasks;
- Spacy 2.0 [88] toolkit by Explosion, which applies convolutional neural networks (CNNs);
- Stanford CoreNLP toolkit [89], which relies on conditional random field (CRF);
- Flair [90] toolkit by Zalando Research, which applies pooled contextualized embeddings together with deep recurrent neural networks for multiple tasks;

The following aspects were evaluated:

- Whether the extractor successfully determined that the model element name had entities that had to be extracted, i.e., whether it contained a verb phrase, a noun phrase, or a named entity.
- Whether the extractor actually extracted the required entities successfully. Note that it was required to evaluate whether both verb phrases and noun phrases were successfully extracted. In cases where multiple phrases had to be extracted, it was considered that all of them had to be present in the output for the result to be considered correct.

The SimpleNLG tool [91] was used to normalize tenses for verb phrases, while the NLTK toolkit [92] was used to implement text chunking with part-of-speech tags obtained as outputs from the NLP tools. The research code required for the experimentation (including NLP operator implementation) is provided in the GitHub repository (<https://github.com/paudan/m2m-nlp-experiment>).

The testing of NLP tools showed that Stanford Stanza was the most promising option, as it outperformed the other tools on both specified tasks in terms of precision and F-score. It was selected for extraction of noun/verb phrases and named entities in the implementation of our NLP-enhanced solution.

7.3. Evaluation Methodology

To get more representative experimentation results, it was decided to perform the evaluation on the micro level, i.e., to run transformations and calculate the performance of each model separately. To simplify the evaluation, we applied decomposition of such transformations into so-called atomic transformations, which are formally represented by the tuple $(I_{SMCT}^1, I_{SMCT}^2, rel(I_{SMCT}^1, I_{SMCT}^2))$, where I_{SMCT}^1 and I_{SMCT}^2 are single instances of *SMCT* and *rel* is a relation or containment type relating I_{SMCT}^1 and I_{SMCT}^2 . For the inputs used in the experiment, these tuples were instances of $(Actor, Use Case)$, $Association(Actor, Use Case)$ and $(Lane, Task, Containment(Lane, Task))$ for the UML use case models and BPMN process models, respectively.

Three measures were selected for the evaluation of experimental results:

- Accuracy, which is defined as the proportion of correctly executed transformations to the total number of performed transformations:

$$accuracy = \frac{\#_{correct}}{\#_{trans}} \quad (1)$$

- Mean deviation between the number of extracted outputs and benchmark output results, which is used to measure the extraction error per model:

$$MeanDiff = \frac{1}{\#_{trans}} \sum_i^{\#_{trans}} \frac{|\#_{actual}^i - \#_{extracted}^i|}{\#_{actual}^i} \quad (2)$$

- Jaccard distance between extracted outputs and actual outputs, which is used to evaluate the proportion of elements generated successfully compared to the set of elements that was generated actually; again, the mean is considered to aggregate the performance per model:

$$Jaccard = \frac{1}{\#_{trans}} \sum_i^{\#_{trans}} \frac{|O_{actual}^i \cap O_{extracted}^i|}{|O_{actual}^i \cup O_{extracted}^i|} \quad (3)$$

Here, $\#_{trans}$ is the total number of atomic transformations performed; $\#_{correct}$ is the number of transformations that were processed correctly; O_{actual}^i is the benchmark set of elements that had to be generated in the i th transformation; $O_{extracted}^i$ is the set of elements that were actually generated in the i th transformation; and $\#_{actual}^i$ and $\#_{extracted}^i$ depict the number of elements in O_{actual}^i and $O_{extracted}^i$ respectively. Note that in this context, a transformation is considered to be correctly executed if all the output elements and their types match the defined benchmark output results for both elements and their types. Therefore, accuracy is considered as the stricter measure, while the mean Jaccard distance value is more appropriate for evaluating partially correct results.

7.4. Experimental Results

The results of the experiment, together with the main descriptive statistics of the performed transformations, are presented in Table 2. In this table, the “Number of executed transformations column” shows the number of transformations performed during the experiment, while the “Number of executed atomic transformations” column presents the actual number of operations performed; the “Number of expected output elements” column shows the total number of elements (I_{TMCT}) per model acquired after manual extraction, and this is considered as the benchmark result; the “Number of output elements” columns for the original and NLP-enhanced solution sections indicate the total number of elements that were acquired after transformation processing of each model. The “MeanDiff”, “Accuracy”, and “Jaccard” columns show the evaluation results obtained with the measures presented in Section 7.3.

Table 2. Experiment results for partial M2M transformation processing.

Model No.	Number of Executed Transformations	Number of Executed Atomic Transformations	Number of Expected Output Elements	Original Solution [14]			NLP-Enhanced Solution				
				Number of Output Elements	MeanDiff	Accuracy	Jaccard	Number of Output Elements	MeanDiff	Accuracy	Jaccard
BPM 1	2	13	41	39	0.462	0.154	0.282	37	0.308	0.846	0.846
BPM 2	2	9	31	27	0.444	0.667	0.667	26	0.556	0.667	0.667
BPM 3	3	9	29	27	0.222	0.556	0.657	24	0.556	0.333	0.444
BPM 4	4	16	48	48	0	0.875	0.911	48	0	0.875	0.938
BPM 5	1	19	57	57	0	0.316	0.535	56	0.053	0.842	0.921
BPM 6	1	13	37	39	0.154	0.154	0.378	37	0	0.923	1
BPM 7	2	13	37	39	0.154	0.462	0.513	37	0	1	1
BPM 8	3	9	32	27	0.778	0.222	0.222	27	0.778	0.444	0.5
BPM 9	1	13	35	38	0.231	0.154	0.231	35	0	0.923	1
BPM 10	3	32	87	93	0.188	0.531	0.538	85	0.063	0.938	0.938
BPM 11	1	10	25	29	0.4	0.4	0.433	23	0.2	0.8	0.8
BPM 12	1	12	33	34	0.083	0.5	0.5	32	0.083	0.917	0.917
BPM 13	3	16	43	48	0.313	0.188	0.271	44	0.063	0.875	0.906
BPM 14	4	9	29	27	0.222	0.556	0.630	24	0.556	0.556	0.556
BPM 15	4	7	21	21	0	0.714	0.75	21	0	1	1
BPM 16	2	12	36	36	0	0.833	0.833	32	0.333	0.667	0.667
BPM 17	1	7	21	21	0	0.286	0.464	21	0	1	1
BPM 18	1	4	10	10	0	0.5	0.5	9	0.25	0.75	0.750
BPM 19	3	10	36	30	0.6	0	0.217	29	0.7	0.6	0.667
BPM 20	4	10	30	30	0	0.8	0.825	27	0.3	0.7	0.7
UCM 1	1	5	14	15	0.2	0.8	0.8	15	0.2	0.2	0.5
UCM 2	1	6	18	18	0	1	1	18	0	1	1
UCM 3	1	5	14	15	0.2	0.8	0.8	13	0.6	0.4	0.4
UCM 4	1	3	9	9	0	1	1	8	0.333	0.333	0.5

Table 2. Cont.

Model No.	Number of Executed Transformations	Number of Executed Atomic Transformations	Number of Expected Output Elements	Original Solution [14]			NLP-Enhanced Solution				
				Number of Output Elements	MeanDiff	Accuracy	Jaccard	Number of Output Elements	MeanDiff	Accuracy	Jaccard
UCM 5	1	3	9	9	0	1	1	9	0	1	1
UCM 6	1	3	9	9	0	1	1	7	0.667	0.333	0.333
UCM 7	1	4	12	12	0	1	1	11	0.25	0.75	0.75
UCM 8	1	4	12	12	0	1	1	11	0.25	0.75	0.75
UCM 9	1	4	12	12	0	1	1	12	0	1	1
UCM 10	1	3	9	9	0	0.667	0.667	8	0.333	0.667	0.667
UCM 11	1	2	6	6	0	0.5	0.667	6	0	0.5	0.75
UCM 12	1	7	21	21	0	1	1	21	0	1	1
UCM 13	2	3	9	9	0	1	1	8	0.333	0.667	0.667
UCM 14	2	2	6	6	0	1	1	6	0	1	1
UCM 15	1	5	15	15	0	0.6	0.6	15	0	1	1
UCM 16	2	3	9	9	0	1	1	9	0	0.333	0.333
UCM 17	1	2	6	6	0	0.5	0.5	6	0	1	1
UCM 18	1	1	3	3	0	1	1	3	0	1	1
UCM 19	1	5	14	15	0.2	0.8	0.8	14	0	0.6	0.7
UCM 20	1	1	3	3	0	1	1	3	0	1	1
Mean values:					0.121	0.663	0.705		0.194	0.755	0.789

The results indicate multiple cases where the NLP-enhanced solution clearly outperformed the original solution. As expected, more in-depth analysis of those cases showed that certain input models contained elements whose names included multiple words in the formulation of noun and verb phrases; additionally, some of those phrases contained articles and other linguistic units that could not be properly handled by the original solution. The NLP enhancements also proved beneficial in processing more complex cases containing model element names with conjunctive/disjunctive clauses. The mean values of transformation accuracy and Jaccard-based measures also confirm the superiority of our new development over the original one. The mean Jaccard distance-based measure between manual expert-based results and results acquired from the NLP-enhanced solution was 0.789, indicating that, on average, almost 80% of the expected output elements were generated successfully per model. The *MeanDiff* measure, representing the mean proportion of failures during the transformation, is also lower for the NLP-enhanced solution, confirming that the application of NLP in the field of M2M transformations has great potential.

It came as no surprise that the original solution in some cases obtained results comparable to or even better than the solution proposed in this paper. The original solution proved to be very effective in those cases, where the most common “<VERB><NOUN>|<NOUN PHRASE>” pattern was used throughout the whole element naming set within a model. In such cases, the introduced LEFT() and RIGHT() operators performed without fail; meanwhile, NLP-based processing may still result in errors due to the false positives in part-of-speech recognition.

7.5. Discussion

The experiment helped us to verify and validate the main conceptual ideas provided in this paper, particularly to prove that language processing technology can be successfully integrated into the M2M transformation solution. The results of the NLP-based processor were reasonable, indicating that almost 80% of target elements could be generated successfully, based on the results obtained using an experimental dataset composed of models from real-world business domains. However, the experiment itself also identified several issues that could be addressed in the future:

- A limited set of bad naming practices, restricted to the most common ones (e.g., naming a use case using a single verb or noun), was considered. During the initial dataset screening, we observed many such cases. Identifying more cases of bad practices and introducing automated resolution of such cases into the developed solution could provide even better transformation results.
- The use of non-alphanumeric symbols (e.g., dashes, commas, apostrophes) inside words also matters. It is advised to remove them from the model element names. While more advanced tokenizers should be able to handle many such cases, the risk of mishandling such cases remains.
- Detection and resolution of abbreviations is also an actual issue. As stated in Section 7.1, they can be very context-dependent and may or may not be recognized as expected. Another similar issue is case sensitivity (e.g., “US” vs. “us”), as named entity recognizers can easily be confused.
- Part-of-speech tagging can be sensitive to letter cases. While some modelers do prefer starting each word with a capital letter when naming activities, tasks, or use cases, some NLP tools may fail to tag them correctly (e.g., “issue invoice” could be tagged as <VERB><NOUN>, but “Issue Invoice” might become <NOUN><NOUN>, which would be an incorrect tagging result). During initial experimentation, we observed that some NLP tools, like Spacy, were quite sensitive to letter cases, which is also significant for practical application, as modeling practitioners tend to use proper or even mixed-case names. While such cases could be normalized to lowercase, doing so increases the risk that some features required for proper processing could be lost (e.g., recognition of named entities by the first uppercase letters).
- Generally, using conjunctive/disjunctive clauses in element names is also considered as a bad modeling practice in modeling BPMN processes, UML use cases, or any other type of activity-like element, as they should be refactored to two or more elements. In our experiment, we considered

that cases containing a single verb phrase and multiple noun phrases could be processed to generate multiple I_{TMCT} output elements (e.g., “assign manager and assistant” can be processed as “assign manager” and “assign assistant”). Multiple verb phrases for the same subject could be processed similarly, provided the verb phrases are identified correctly (e.g., “create and process invoice” can be processed as “create invoice” and “process invoice”). However, having multiple candidate verb phrases and noun phrases leads to much more complex processing, which may require more advanced NLP techniques, such as dependency parsing. The same applies to cases where verb phrases, noun phrases, or both are separated by commas or other separator symbols (e.g., “create, process, and manage invoice”). This could be addressed in our further research.

- There can be general ambiguity in detecting named entities and abbreviations. As stated in the previous section, there are numerous situations where entities cannot be processed correctly due to a lack of contextual information. Additional tools that take into account multiple features to improve precise meaning detection (e.g., context-based classifiers) could be applied to mitigate this problem and to correct extractor prediction. Such developments would require additional sources of input data and could also be considered as one of the goals of our future research.

8. Conclusions

In this paper, we present an NLP extension to user-interacted partial M2M transformations based on drag-and-drop actions. The previously introduced key features, emphasizing the customization, usability, and reusability of model-driven M2M transformations [14], were extended with NLP capability to enable advanced text processing of model element names and properties. This ensures that the outputs of M2M transformation are more representative of and aligned with the actual intent of transformation designers. This work presents an additional extension to the underlying visual language for modeling mapping patterns in M2M transformation specifications, mainly natural language-based operators, which can be used to develop M2M transformations with greater flexibility. Additionally, we refined our previously developed M2M transformation metamodel by introducing an extension that allows one to model conditional transformations for cases where more advanced transformation scenarios are required.

As shown in our previous work as well as in this paper, our solution can be applied to any graphical modeling language implemented as a UML profile or the UML itself. The solution has already undergone experimentation with four modeling languages, UML, BPMN, SoaML, and SBVR, by using their UML profiles [15,17,18] implemented in the MagicDraw CASE tool. Again, we expect our approach to be easily portable to other visual modeling platforms that meet certain requirements due to the underlying abstract framework we developed. This framework also encompasses all of the extensions presented in the paper, therefore it could be used to build CASE tool-specific components for other MOF-based platforms as well.

In this paper, we present several actual transformation use cases illustrating the usability of the proposed NLP-extended solution for designing and executing partial M2M transformations across different modeling languages. Obviously, this is not a finite number of all possible use cases; specialized libraries of such transformation specifications could be developed and customized by other practitioners working in this field. Evaluating recent NLP technologies based on state-of-the-art machine learning and deep learning techniques proved the potential of the proposed approach. The Stanford Stanza-based extractor was recognized as the leading tool and was used to implement the functionality offered in this paper. Nevertheless, we also conclude that data quality should be taken into account, as bad modeling practices or invalid names will not be processed correctly by any tool, irrespective of its performance in related fields.

While this approach presents a significant amount of novelty, we already established several extension points for our future research. Extending condition specification language to support expressions of even greater complexity, including composite predicates and disjunctive and conjunctive clauses, is considered as one of the primary tasks; this is one of the basic requirements leading to the

development of more advanced and comprehensive model-based transformation language. Condition resolution and elimination (or notification) of contradictions/overlaps in branching conditions would be another advanced feature contributing to future development. Also, more sophisticated transformations could be enabled by extending the existing transformation engine so that it could process deeper hierarchical levels of the designed transformation patterns. We are quite firm in our assumption that such extensions could lead to supporting full-scale M2M transformations and serve as a competitive alternative to existing model-wide M2M transformation solutions.

Author Contributions: P.D.: Conceptualization, Investigation, Software, Writing—original draft, Writing—review & editing. T.S.: Conceptualization, Investigation, Methodology, Writing—original draft, Writing—review & editing. R.B.: Investigation, Supervision. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Anjorin, A.; Lauder, M.; Patzina, S.; Schürr, A. eMoflon: Leveraging EMF and professional CASE tools. In Proceedings of the Tagungsband Der Informatik 2011, Lecture Notes in Informatics, Berlin, Germany, 4–7 October 2011; Volume 192.
2. Klassen, L.; Wagner, R. EMorF—A tool for model transformations. *Electron. Commun. EASST* **2012**, *54*, 1–6.
3. Object Management Group (OMG). Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. OMG spec v.1.2. 2015. Available online: <https://www.omg.org/spec/QVT/1.2> (accessed on 1 July 2020).
4. Jouault, F.; Allilaire, F.; Bézivin, J.; Kurtev, I.; Valduriez, P. ATL: A QVT-like transformation language. In Proceedings of the Companion to the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA, Portland, OR, USA, 22–26 October 2006.
5. Young, T.; Hazarika, D.; Poria, S.; Cambria, E. Recent trends in deep learning based natural language processing. *IEEE Comput. Intel. Mag.* **2018**, *13*, 55–75. [[CrossRef](#)]
6. Otter, D.W.; Medina, J.R.; Kalita, J.K.A. Survey of the usages of deep learning in natural language processing. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, 1–21. [[CrossRef](#)]
7. Mens, T.; Van Gorp, P.A. Taxonomy of model transformation. *Electron. Notes Theor. Comput. Sci.* **2006**, *152*, 125–142. [[CrossRef](#)]
8. Rose, L.M.; Herrmannsdoerfer, M.; Mazanek, S.; Van Gorp, P.; Buchwald, S.; Horn, T.; Kalnina, E.; Lano, K.; Schätz, B.; Wimmer, M. Graph and model transformation tools for model migration. *Softw. Syst. Model.* **2012**, *13*, 323–359. [[CrossRef](#)]
9. Hildebrandt, S.; Lambers, L.; Giese, H.; Rieke, J.; Greenyer, J.; Schäfer, W.; Lauder, M.; Anjorin, A.; Schürr, A. A survey of triple graph grammar tools. *Electron. Commun. EASST* **2013**, *57*. [[CrossRef](#)]
10. Lano, K.; Kolahdouz-Rahimi, S.; Yassipour-Tehrani, S.; Sharbaf, M. A survey of model transformation design patterns in practice. *J. Syst. Softw.* **2018**, *140*, 48–73. [[CrossRef](#)]
11. Silva, G.C.; Rose, L.; Calinescu, R.A. Qualitative study of model transformation development approaches: Supporting novice developers. In Proceedings of the 1st International Workshop in Model-Driven Development Processes and Practices (MD2P2), Valencia, Spain, 28 September–3 October 2014; pp. 18–27.
12. Dori, D. *Model-Based Systems Engineering with OPM and SysML*; Springer: New York, NY, USA, 2016.
13. Skersys, T.; Danenas, P.; Butleris, R. Extracting SBVR business vocabularies and business rules from UML use case diagrams. *J. Syst. Softw.* **2018**, *141*, 111–130. [[CrossRef](#)]
14. Skersys, T.; Danenas, P.; Butleris, R. Model-based M2M transformations based on drag-and-drop actions: Approach and implementation. *J. Syst. Softw.* **2016**, *122*, 327–341. [[CrossRef](#)]
15. Skersys, T.; Pavalkis, S.; Nemuraite, L. Implementing Semantically Rich Business Vocabularies in CASE Tools. In Proceedings of the AIP International Conference on Numerical Analysis and Applied Mathematics (ICNAAM-2014), Rhodes, Greece, 22–28 September 2014; Theodore, E.S., Charalambos, T., Eds.; AIP Publishing: Melville, NY, USA, 2015; Volume 1648, pp. 1–4.
16. Object Management Group (OMG). Semantics of Business Vocabulary and Business Rules (SBVR) v.1.5, OMG Doc. No.: Formal/2019–10–02. 2019. Available online: <https://www.omg.org/spec/SBVR/About-SBVR/> (accessed on 1 July 2020).

17. Object Management Group (OMG). UML Profile for BPMN Processes. OMG spec. v.1.0. 2014. Available online: <http://www.omg.org/spec/BPMNProfile/1.0/> (accessed on 1 July 2020).
18. Object Management Group (OMG). Service Oriented Architecture Modeling Language (SoaML). OMG Spec. v. 1.0.1. May 2012. Available online: www.omg.org/spec/SoaML/ (accessed on 1 July 2020).
19. Object Management Group (OMG). OMG System Modeling Language. OMG spec. v.1.6. 2019. Available online: <https://www.omg.org/spec/SysML/1.6/> (accessed on 1 July 2020).
20. Vileiniškis, T.; Skersys, T.; Pavalkis, S.; Butleris, R.; Butkienė, R. Lightweight Approach to Model Traceability in a CASE Tool. In Proceedings of the AIP Conference Proceedings: International Conference of Numerical Analysis and Applied Mathematics (ICNAAM 2016), Rhodes, Greece, 19–25 September 2016; AIP Publishing: Melville, NY, USA, 2017; Volume 1863 A, pp. 1–4. [[CrossRef](#)]
21. Kahani, N.; Bagherzadeh, M.; Cordy, J.R.; Dingel, J.; Varro, D. Survey and classification of model transformation tools. *Softw. Syst. Model.* **2019**, *18*, 2361–2397. [[CrossRef](#)]
22. Schürr, A. Specification of graph translators with triple graph grammars. In Proceedings of the WG'94 Workshop on Graph-Theoretic Concepts in Computer Science, Herrsching, Germany, 16–18 June 1994; pp. 151–163.
23. Arendt, T.; Biermann, E.; Jurack, S.; Krause, C.; Taentzer, G. Henshin: Advanced concepts and tools for in-place EMF model transformations. In Proceedings of the Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Oslo, Norway, 3–8 October 2010; Volume 6394, pp. 121–135.
24. Biermann, E.; Ehrig, K.; Köhler, C.; Kuhns, G.; Taentzer, G.; Weiss, E. Graphical definition of in-place transformations in the Eclipse modeling framework. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4199, pp. 425–439.
25. Schippers, H.; Van Gorp, P.; Janssens, D. Leveraging UML profiles to generate plugins from visual model transformations. *Electron. Notes Theor. Computer Sci.* **2005**, *127*, 5–16. [[CrossRef](#)]
26. Van Gorp, P. Model-Driven Development of Model Transformations. Ph.D.Thesis, University of Antwerpen, Antwerpen, Belgium, 2008.
27. Muliawan, O.; Janssens, D. Model refactoring using MoTMoT. *Int. J. Softw. Tools Technol. Trans.* **2010**, *12*, 201–209. [[CrossRef](#)]
28. Sendall, S.; Perrouin, G.; Guelfi, N.; Biberstein, O. *Supporting Model-to-Model Transformat: The VMT Approach*; CTIT Technical Report TR-CTIT-03–27; University of Twente: Enschede, The Netherlands, 2003.
29. Willink, E.D. UMLX: A graphical transformation language for MDA. In Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications, Nuremberg, Germany, 7–10 November 2003; pp. 13–24.
30. Agrawal, A.; Karsai, G.; Neema, S.; Shi, F.; Vizhanyo, A. The design of a language for model transformations. *Softw. Syst. Model.* **2006**, *5*, 261–288. [[CrossRef](#)]
31. Salemi, S.; Selamat, A.; Penhaker, M. A model transformation framework to increase OCL usability. *J. King Saud Univ. Comput. Inf. Sci.* **2016**, *28*, 13–26. [[CrossRef](#)]
32. Kalnins, A.; Barzdins, J.; Celms, E. The model transformation language MOLA. *Lect. Notes Comput. Sci.* **2005**, *3599*, 62–76.
33. The Eclipse Foundation: Viatra Project. 2016. Available online: <http://www.eclipse.org/viatra/> (accessed on 1 July 2020).
34. Ergin, H.; Syriani, E.; Gray, J. Design pattern-oriented development of model transformations. *Comput. Lang. Syst. Struct.* **2016**, *46*, 106–139. [[CrossRef](#)]
35. Lano, K.; Kolahdouz-Rahimi, S. Model-transformation design patterns. *IEEE Trans. Softw. Eng.* **2014**, *40*, 1224–1259. [[CrossRef](#)]
36. Leopold, H.; Smirnov, S.; Mendling, J. On the refactoring of activity labels in business process models. *Inf. Syst.* **2012**, *37*, 443–459. [[CrossRef](#)]
37. Leopold, H.; Rami-Habib, E.-S.; Mendling, J.; Guerreiro Azevdo, L.; Baião, F.A. Detection of naming convention violations in process models for different languages. *Decis. Support Syst.* **2013**, *56*, 310–325. [[CrossRef](#)]
38. Leopold, H. *Natural Language in Business Process Models: Theoretical Foundations, Techniques, and Applications*; Springer International Publishing: Cham, Switzerland, 2013.

39. Pittke, F. Linguistic Refactoring of Business Process Models. Ph.D. Thesis, WU Vienna University of Economics and Business, Vienna, Austria, 2015.
40. Pittke, F.; Leopold, H.; Mendling, J. When language meets language: Anti patterns resulting from mixing natural and modeling language. In Proceedings of the BPM: Business Process Management Workshops, Eindhoven, The Netherlands, 7–11 September 2014; pp. 118–129.
41. Leopold, H.; Pittke, F.; Mendling, J. Ensuring the canonicity of process models. *Data Knowl. Eng.* **2017**, *11*, 22–38. [[CrossRef](#)]
42. Somogyi, F.A.; Asztalos, M. Systematic review of matching techniques used in model-driven methodologies. *Softw. Syst. Model.* **2020**, *19*, 693–720. [[CrossRef](#)]
43. Pileggi, S.F.; Fernandez-Llatas, C. *Semantic Interoperability Issues, Solutions, Challenges*; River Publisher: Wharton, TX, USA, 2012.
44. Kukich, K. Techniques for automatically correcting words in text. *ACM Comput. Surv.* **1992**, *24*, 377–439. [[CrossRef](#)]
45. Rozovskaya, A.; Roth, R. Grammatical error correction: Machine translation and classifiers. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Berlin, Germany, 7–12 August 2016; Volume 1, pp. 2205–2215.
46. Junczys-Dowmunt, M.; Grundkiewicz, R.; Guha, S.; Heafield, K. Approaching neural grammatical error correction as a low-resource machine translation task, NAACL 2018, Association for Computational Linguistics. *arXiv* **2018**, arXiv:1804.05940.
47. Kiyono, S.; Suzuki, J.; Mita, M.; Mizumoto, T.; Inui, K. An empirical study of incorporating pseudo data into grammatical error correction. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Hong Kong, China, 3–7 November 2019; pp. 1236–1242.
48. Petrov, S.; Barrett, L.; Thibaux, R.; Klein, D. Learning accurate, compact, and interpretable tree annotation. In Proceedings of the ACL-44 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics, Sydney, Australia, 22 July 2006; pp. 433–440.
49. Socher, R.; Lin, C.C.-Y.; Ng, A.Y.; Manning, C.D. Parsing natural scenes and natural language with recursive neural networks. In Proceedings of the 28th International Conference on International Conference on Machine Learning (ICML'11), Madison, WI, USA, 24–27 July 2011; Getoor, L., Scheffer, T., Eds.; Omnipress: Madison, WI, USA, 2011; pp. 129–136.
50. Vinyals, O.; Kaiser, L.; Koo, T.; Petrov, S.; Sutskever, I.; Hinton, G. Grammar as a foreign language. In Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS'15), Cambridge, MA, USA, 7–12 December 2015; MIT Press: Cambridge, MA, USA, 2015; Volume 2, pp. 2773–2781.
51. Dozat, T.; Manning, C.D. Simpler but more accurate semantic dependency parsing. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, Melbourne, Australia, 15 July 2018; Volume 2, pp. 484–490.
52. Agichtein, E.; Gravano, L. Snowball: Extracting relations from large plain-text collections. In Proceedings of the Fifth ACM Conference on Digital Libraries, ACM, San Antonio, TX, USA, 2–6 June 2000.
53. Lai, S.; Leung, K.S.; Leung, Y. SUNNYNLP at SemEval-2018 Task 10: A support-vector-machine-based method for detecting semantic difference using taxonomy and word embedding features. In Proceedings of the 12th International Workshop on Semantic Evaluation (Semeval), Association for Computational Linguistics, New Orleans, LA, USA, 5–6 June 2018; pp. 741–746.
54. Santus, E.; Biemann, C.; Chersoni, E. BomJi at SemEval-2018 Task 10: Combining vector-, pattern- and graph-based information to identify discriminative attributes. In Proceedings of the 12th International Workshop on Semantic Evaluation (SEMEVAL), Association for Computational Linguistics, New Orleans, LA, USA, 5–6 June 2018; pp. 741–746.
55. Kumar, S. A Survey of deep learning methods for relation extraction. CoRR abs/1705.03645. *arXiv* **2017**, arXiv:1705.03645.
56. Smirnova, A.; Cudré-Mauroux, P. Relation extraction using distant supervision: A survey. *ACM Comput. Surv.* **2018**, *51*, 106. [[CrossRef](#)]
57. He, L.; Lee, K.; Lewis, M.; Zettlemoyer, L. Deep semantic role labeling: What works and what's next. In Proceedings of the 55th Annual Meeting of the Association For Computational Linguistics, Vancouver, WA, Canada, 30 July–4 August 2017; Volume 1, pp. 473–483.

58. Lee, J.; Seo, S.; Choi, Y.S. Semantic relation classification via bidirectional LSTM networks with entity-aware attention using latent entity typing. *Symmetry* **2019**, *11*, 785. [CrossRef]
59. Ren, F.; Zhou, D.; Liu, Z.; Li, Y.; Zhao, R.; Liu, Y.; Liang, X. Neural relation classification with text descriptions. In Proceedings of the 27th International Conference on Computational Linguistics, Santa Fe, NM, USA, 20–26 August 2018; pp. 1167–1177.
60. Belinkov, Y.; Glass, J. Analysis methods in neural language processing: A survey. *Trans. Assoc. Comput. Linguist.* **2019**, *7*, 49–72. [CrossRef]
61. No Magic, Inc.: UML Profiling and DSL. User Guide, v 19.0 LTR. 2019. Available online: <https://docs.nomagic.com/display/MD185/UML+Profiling+and+DSL+Guide> (accessed on 1 July 2020).
62. Miller, G. WordNet: A lexical database for English. *Comm. ACM* **1995**, *38*, 39–41. [CrossRef]
63. Apache Software Foundation. Apache OpenNLP Natural Language Processing Library. 2014. Available online: <http://opennlp.apache.org/> (accessed on 1 July 2020).
64. Baker, C.F.; Fillmore, C.J.; Lowe, J.B. The Berkeley FrameNet project. In Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1 (ACL '98/COLING '98), Association for Computational Linguistics, Stroudsburg, PA, USA, 10–14 August 1998; Volume 1, pp. 86–90.
65. Navigli, R.; Ponzetto, S. BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artif. Intell.* **2012**, *193*, 217–250. [CrossRef]
66. Chrupala, G.A.; Dinu, G.; van Genabith, J. Learning morphology with Morfette. In Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08), Marrakech, Morocco, 28–30 May 2008.
67. Müller, T.; Cotterell, R.; Fraser, A.; Schütze, H. Joint lemmatization and morphological tagging with lemming. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Lisbon, Portugal, 17–21 September 2015; pp. 2268–2274.
68. Dridan, R.; Oepen, S. Tokenization: Returning to a long solved problem—a survey, contrastive experiment, recommendations, and toolkit. In Proceedings of the ACL 12 Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2, Jeju Island, Korea, 8–14 July 2012; pp. 378–382.
69. Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; Kuksa, P. Natural language processing (almost) from scratch. *J. Machine Learn. Res.* **2011**, *12*, 2493–2537.
70. Toutanova, K.; Klein, D.; Manning, C.D.; Yoram Singer, Y. Feature-rich part-of-speech tagging with a cyclic dependency network. In Proceedings of the HLT-NAACL, Edmonton, AL, Canada, 27 May–1 June 2003; Volume 1, pp. 173–180.
71. Huang, Z.; Xu, W.; Yu, K. Bidirectional LSTM-CRF models for sequence tagging. *arXiv* **2015**, arXiv:1508.01991.
72. Brill, E. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Comput. Ling.* **1995**, *21*, 543–565.
73. Chiu, J.P.C.; Nichols, E. Named entity recognition with bidirectional LSTM-CNNs. *Trans. Assoc. Comput. Ling.* **2016**, *4*, 357–370. [CrossRef]
74. Lample, G.; Ballesteros, M.; Subramanian, S.; Kawakami, K.; Dyer, C. Neural architectures for named entity recognition. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL), Association for Computational Linguistics, San Diego, CA, USA, 12–17 June 2016; pp. 260–270.
75. Peters, M.E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; Zettlemoyer, L. Deep contextualized word representations. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), New Orleans, LA, USA, 1–6 June 2018; Volume 1, pp. 2227–2237.
76. Ghaddar, A.; Langlais, P. Robust lexical features for improved neural network named-entity recognition. In Proceedings of the 27th International Conference on Computational Linguistics (COLING), Association for Computational Linguistics, Santa Fe, NM, USA, 20–26 August 2018; pp. 1896–1907.
77. Liu, L.; Shang, J.; Ren, X.; Xu, F.; Gui, H.; Peng, J.; Han, J. Empower sequence labeling with task-aware neural language model. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; pp. 5253–5260.

78. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.; Dean, J. Distributed representations of words and phrases and their compositionality. In Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 2 (NIPS'13), Lake Tahoe, NV, USA, 5–8 December 2013; pp. 3111–3119.
79. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, MN, USA, 2–7 June 2019; pp. 4171–4186.
80. Hearst, M.A. Automatic acquisition of hyponyms from large text corpora. In Proceedings of the Conference on Computational Linguistics, Volume 2, Association for Computational Linguistics, Nantes, France, 23–28 August 2013.
81. Snow, R.; Jurafsky, D.; Ng, A.Y. Learning Syntactic Patterns for Automatic Hypernym Discovery. In Proceedings of the 17th International Conference on Neural Information Processing Systems (NIPS'04), Singapore, 5–8 December 2006; Saul, L.K., Weiss, Y., Bottou, L., Eds.; MIT Press: Cambridge, MA, USA, 2004; pp. 1297–1304.
82. Onofrei, M.; Hulub, I.; Trandabăt, D.; Gifu, D. Apollo at SemEval-2018 Task 9: Detecting hypernymy relations using syntactic dependencies. In Proceedings of the 12th International Workshop on Semantic Evaluation (SEMEVAL), Association for Computational Linguistics, New Orleans, LA, USA, 5–6 June 2018; pp. 898–902.
83. Kawaumra, T.; Sekine, M.; Matsumura, K. Hyponym/hypernym detection in science and technology thesauri from bibliographic datasets. In Proceedings of the 2017 IEEE 11th International Conference on Semantic Computing (ICSC), San Diego, CA, USA, 17–19 September 2017; pp. 180–187.
84. Zhang, Z.; Li, J.; Zhao, H.; Tang, B. SJTU-NLP at SemEval-2018 Task 9: Neural hypernym discovery with term embeddings. In Proceedings of the 12th International Workshop on Semantic Evaluation (SEMEVAL), Association for Computational Linguistics, New Orleans, LA, USA, 5–6 June 2018; pp. 903–908.
85. Hassan, A.Z.; Vallabhajosyula, M.S.; Pedersen, T. UMDuluth-CS8761 at SemEval-2018 Task9: Hypernym discovery using hearst patterns, co-occurrence frequencies and word embeddings. In Proceedings of the 12th International Workshop on Semantic Evaluation (SEMEVAL), Association for Computational Linguistics, New Orleans, LA, USA, 5–6 June 2018; pp. 914–918.
86. Weske, M.; Decker, G.; Dumas, M.; La Rosa, M.; Mendling, J.; Reijers, H.A. Model collection of the business process management academic initiative. *Zenodo* 2020. [[CrossRef](#)]
87. Qi, P.; Zhang, Y.; Zhang, Y.; Bolton, J.; Manning, C.D. Stanza: A Python natural language processing toolkit for many human languages. *arXiv* 2020, arXiv:2003.07082.
88. Honnibal, M.; Montani, I. spaCy 2 GitHub Directory (2020). Available online: <https://github.com/explosion/spaCy> (accessed on 1 July 2020).
89. Manning, C.D.; Mihai, S.; Bauer, J.; Finkel, J.; Bethard, S.J.; McClosky, D. The Stanford CoreNLP natural language processing toolkit. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, Baltimore, MD, USA, 23–24 June 2014; pp. 55–60.
90. Akbik, A.; Bergmann, T.; Blythe, D.; Rasul, K.; Schweter, S.; Vollgraf, R. FLAIR: An easy-to-use framework for state-of-the-art NLP. In Proceedings of the 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations), NAACL 2019, Minneapolis, MN, USA, 2–7 June 2019; pp. 54–59.
91. Gatt, A.; Reiter, E. SimpleNLG: A realisation engine for practical applications. In Proceedings of the 12th European Workshop on Natural Language Generation (ENLG 2009), Athens, Greece, 30–31 March 2009; pp. 90–93.
92. Bird, S.; Loper, E.; Klein, E. *Natural Language In Processing with Python*; O'Reilly Media Inc.: Newton, MA, USA, 2009.

