Model Transformations with Reference Models

Willem-Jan van den Heuvel and Manfred Jeusfeld

Department of Information Systems and Management
PO Box 90500, 5000 LE, Tilburg, The Netherlands
wjheuvel@uvt.nl, jeusfeld@uvt.nl

Abstract
In this paper we introduce and explore an extension to the existing paradigm of model transformation. Specifically, we extend existing model transformation approaches by considering reference models and human input as important sources for and during model transformation. To cater for this new type of model transformation, we develop an approach grounded on a common generic model and a series of transformation operators, which constitute an non-trivial extension to the "classical" model management operators.
Keywords: model transformation, model-driven architecture, reference model, customization

INTRODUCTION

Model transformations have been studied from the 80s, playing in central role in software development in general, and model-driven software engineering (MDSE) more in particular [5]. The Model Driven Architecture (MDA) that is promoted by the OMG, reflects the latest wave of efforts to make the vision of automatic generation of programming code from models a reality. At heart of this paradigm lays the notion of a transformation, which allows translating one or more target models into one or more source models [6].
Unfortunately, before the MDE and its relative, the MDA, become a reality, many open issues need to be tackled. Firstly, today's transformation approaches implicitly assume that a target model may be constructed from scratch, using a new source model as its input. This assumption is unrealistic given the fact that in many cases enterprise models are constructed by customizing pre-existing, standard enterprise models, also referred to reference models. Notably, reference models are used to configure ERP systems so that they fulfill organization specific requirements and constraints. Some recent estimates indicate that approximately 70% of all companies are implementing some form of ERP system [16].
Given this situation, it is of critical importance to introduce new types of transformations, which allow tailoring and/or augmenting ERP systems and

underlying reference architectures so that they adhere to business requirements of specific enterprises.
This paper scrutinizes how pre-existing models, in particular reference models, may be configured in a semi-automatic manner by using model transformations as the key mechanism. In the next section, we will firstly consider the application of transformations for the purpose of customization of reference models in more detail. Next, we present a running example that will be used throughout the paper to motivate and exemplify a methodological framework for customization that is introduced in section 4. This framework combines procedural and declarative transformation rules and allows a semi-automatic transformation process that is not only driven by models as input, but also utilizes information from external (human) resources. Lastly, section 5 concludes this paper summarizing findings and explicating our future research agenda.

TRANSFORMATION FOR CUSTOMIZATION

Model Transformation

A transformation may be defined as the (automatic) generation of a target model from a source model, according to a set of transformation rules ([5], [10]). Abstractly speaking, transformation may be achieved in two orthogonal ways, adopting a declarative or procedural approach. Procedural transformations are typically coded in an imperative language, explicitly specifying every transformation step and their sequencing. Declarative languages on the other hand, implicitly define model transformations, and are typically more flexible and powerful. Graph transformations lay at heart of many contemporary declarative approaches in MDSE practice, defining a transformation rule as a graph-rewriting rule allowing a LHS "source" graph to be substituted by a RHS "target" graph. This is basically realized as follows. Once a matching LHS graph is found, a rule is triggered that swaps the LHS (sub-) graph by a RHS graph. A prominent example of a transformation language that is based on graph grammar entails GReaT [1].
Some languages, PROGRESS [5], the QVT-Partners approach [11] and MT [12], constitute a blend of both approaches, combining the precision of declarative and the comprehensiveness of procedural approaches.

Reference Models

Unless in unusual cases in which software can be developed from scratch, transformations need to be capable to incorporate pre-existing knowledge that is captured in several (given) source models. Notably, reference models encapsulate generic know-how to speed up modeling and to transform generic (ERP) solutions into into enterprise-specific ones [14]. In particular, reference models capture generic characteristics and dynamics of a domain in standard process and/or entity models [8].

Variability and Parameterization

Genericity in reference models may be basically accommodated in three ways [7], (1) offering patterns (as offered by IBM's SanFransico Frameworks), (2) allowing various similar options using OO-like specializations or aggregations, and, (3) pre-defining abstract parameters that need to instantiated according to reflect business conditions. Once tailored to accommodate common domain requirements, reference models have proven to be valuable in the process of customizing ERP packages such as SAP's MySAP and Microsoft's Axia.
Parameterization plays a pivotal role in customization: the parameterization process allows setting parameters of reference models given enterprise specific business processes and policies. For example, the delivery notes can be processed in the SAP Blueprint without any reference to a preceding order (option 1), referencing an individual sales order (option 2), referring to a delivery-due list (option 3), or from a stock transport order (option 4) [2].
In particular, with customization, also referred to as parameterization, we imply that the parameters/variants of features (processes, functions and/or entities) in a reference model are set according to customer-specific requirements. In particular, we may distinguish between the following types of parameters [4]:

- Dimension; feature whose subfeatures all represent a variant
- Dimension with optional features
- Extension point
- Extension point with optional features
- Extension point with OR-features

Rendering Variability

Variability and commonality in software systems has attracted much attention in the domain of Domain Modeling and Product Line Architectures (PLAs) with UML [11]. In short, variability refers to the capacity of systems and models to be changed, tailored or parameterized to enterprise specific requirements.
Surprisingly, only very limited attention to variability is paid in the domain of business modeling (and reference models alike). In [9], extensions to classical Event Process Chains are proposed that allow for configurable functions (that may be included, skipped or conditionally skipped) and connectors (Joins and Splits).

MOTIVATING SCENARIO

The motivating scenario deals with a realistic, yet simplified, case study that transforms two source models (an enterprise model capturing a invoice processing scenario and a reference model of SAP R/3 invoice processes) into a customized reference model that serves as the basis for the actual configuration and deployment of SAP R/3. In this section, we will first introduce the two source models.

Reference Model

Figure 1 depicts an excerpt from the SAP R/3 Reference Model (taken from:[ROS06]) that is modeled using an Event Process Chain (EPC) [SC99]. Events are represented as hexagons, a task or function is depicted as a rectangle, and dotted arrows denote the control flow. Three basic logical operators may be used to express logical relationships between events and functions/tasks: XOR, AND, and, OR. The dotted lines and boxes on the lines are extensions to demarcate four model chunks (see below).

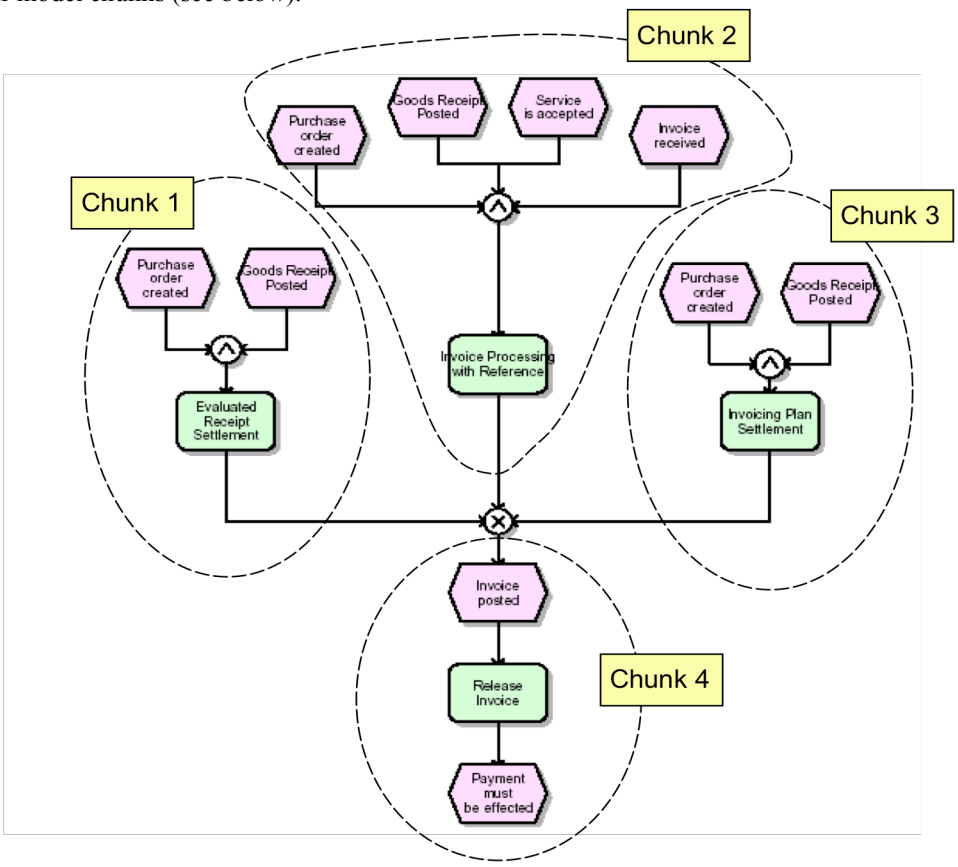

Figure 1 A reference model for invoice processing (adopted from: [9])

This model encompasses three variants of invoice processing that are implemented in SAP R/3 (labelled Chunk 1-3 in Figure 1): standard, evaluated receipt settlement (ERS) and invoice plan settlement (IPS). Standard invoice processing consists of creating a purchase order, receiving the invoice and a check upon actual receipt of goods or services. ERS circumvents these checks and allows to directly processing the invoice after goods have been received and a purchase order has been created. IRS entails an alternate invoicing scheme that enables staged or planned (e.g., periodic) payments. Standard invoice processing is drafted at the core of this

reference model while variant 2 (ERS) and variant 3 (IPS) are rendered at respectively the left hand and the right hand side.

Enterprise Model

This enterprise model depicts the key business tasks for creating a voucher, which serves as a preparatory step for actual payment to creditors of the University of

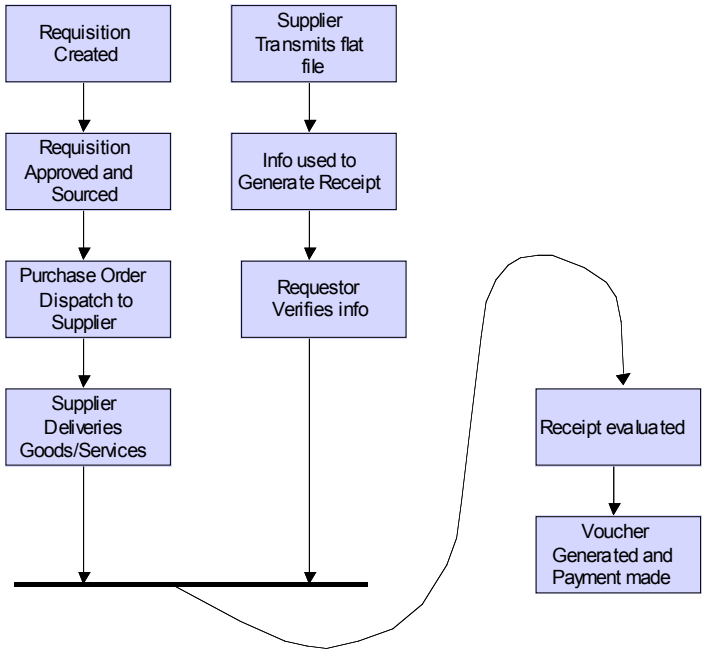

Figure 2 Enterprise Model

Sydney[1]. It is rendered in a UML Activity Diagram alike notation[2]. The invoicing process at the university works basically as follows. Once a purchase requisition is authorized, it is sourced into a purchase order, and subsequently dispatched to the corresponding Supplier. Upon delivery of the goods and services at the University, a Receipt document is created that details the Purchase order, mentioning the quantity that was received. Once the receipt of an Invoice is received from the Supplier and chekced. If the unversity agrees, the receipt is used as the basis for creating a Voucher. Payment is then made to the Supplier in accordance to the payment terms and payment method previously agreed.

---

[1] Source: http://www.finance.usyd.edu.au/docs/supplier_invoice_flatfile.pdf

[2] We found out that typically, business users will resort to a non-formal type of graphical notation, free natural language texts or tables for the purpose of customizing a reference model.

The university assumes an ERS style of payment so Vouchers will only be generated for Receipts that have a matching Goods Receipt Note (GRN), that is approved for payment by the Requestor. In this situation, it is no longer required that paper versions of the Invoice are sent to the Requestor for authorization, smoothening the payment process.

THE METHODOLOGICAL FRAMEWORK

While traditional model transformations establish (fully) automated mappings between a source and target model, transformation for the purpose of customizing reference models entails a more complex, multi-dimensional and inherently interactive process.

Transformation approach

Complexity of reference-guided transformation is believed to be higher than in case of traditional transformations because of four main reasons. Firstly, a reference model does not just simply serve as input to the transformation, but also pre-determines the structure and notation of the target model. This is due to the fact that a reference model contains re-usable building blocks that are plugged into some position of the target model. Secondly, a reference model abstracts from a specific usage, i.e. it can be applied for many application scenarios but needs to be instantiated before it can be used. Thirdly, reference modeling requires a sophisticated matchmaking process to compare a reference model to parts of a source model in order to ascertain its re-usability. The latter is a particular challenge because the reference model is abstract in nature, while a source model is enterprise-specific and rendered in another modeling language. Figure 3 presents the conceptual architecture of reference model guided model transformation.
Transformation lies at heart of this architecture, using multiple models as input, including a reference mode that is to be morphed into a customized counterpart. Although we have depicted the transformation process using one ellipse, we foresee an incremental transformation process comprising a series of

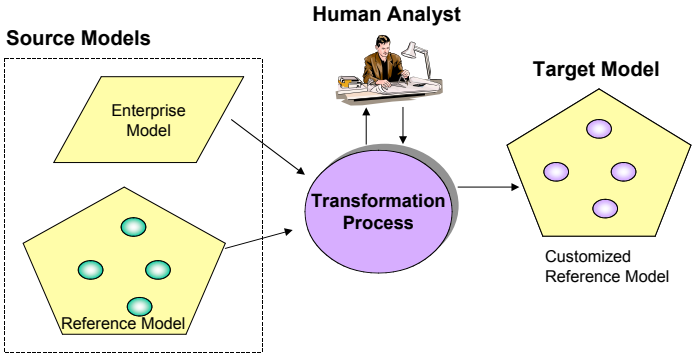

Figure 3: Conceptual Architecture of model transformation of Reference Models

transformation steps, each of which may formulate its own pre- and post-conditions to ensure proper operations. During each transformation step one or more of the input models may be consulted.
The source model constitutes the traditional input of the transformation process, and its usage is obvious: concepts of the source model are used to tailor or extend the functionality in the reference model. We here assume that the source model entails a domain model of an enterprise, representing requirements derived from business processes, roles, entities and the like. Reference models serve as additional input to the transformation process. They are matched against the source model, and are instantiated during the transformation. The selection of a suitable reference model requires formulating a search criterion based on the input concepts of the source model and possible the (partially existing) target model. If information contained in the source and reference models is insufficient for a transformation step, then a human user can provide the missing information.
Based on this architecture, we propose a staged methodology that comprises four

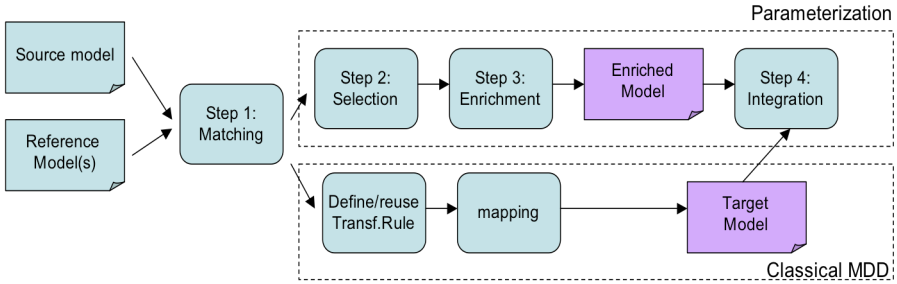

Figure 4 Methodological Framework for Transforming Reference Models

basic steps (see Figure 4): (1) matchmaking, (2) selection (3) enrichment and (4) integration. During the first step, the source model is matched to reference models in order to identify which reference models cater best the requirements from the business domain. In case no suitabe reference model(s) is avaialable, a traditional MDD approach may be followed, encompassing definition or reuse of transformation rules, and transforming (part of) the source into a target model. Next, in step 2 scenarios are selected from the reference model. Step 3 then aims at enriching the reference model to capture more detail and select variants. Lastly, the step 4 aims at integrating the enriched reference model with other target models, e.g., generated from a source model deploying a traditional MDD approach. If the target model is empty, it is just copied. Modeling and schema integration techniques may be used to facilitate this step. In the remainder of this paper we will assume that a reference model has been pre-determined (given the choice of a particular ERP packagee). Instead, we will concentrate on Step 2 and Step 3 of the approach, as they play a pivotal role in the methodological framework.

SELECTION

Intuitively, the running example suggests that the SAP R/3 reference model is a suitable candidate to implement the invoicing scenario. However, only the left (evaluated receipt settlement) branch is applicable and shall be included in the target model.
To support identification of reusable chunks in a reference model, the following transformation primitives should be supported:
`1. Remove(p):` The chunk p of the reference model is removed. The result is a subset of the reference model, that is better aligned to the business scenario.
`2. Aggregate(c1,c2,c3):` Concept c1 and c2 are assembled into a aggregate concept c3. For example, a sequence of activities is aggregated into a single (more abstract) activity.
`3. Rename(n1=n2)`: Concept label n1 in the reference model is replaced by n2.
`4. Split(c1,c2,c3,n)`: Concept c1 is split into c2 and c3; the concepts c2 and c3 are connected by a link labeled n. This is the reverse operation to Aggregate.

Selection using a Generic Model

Matching the enterprise and reference models requires to represent them in a way that makes them comparable. We adopt the meta-model approach used in GoRoMe [17] for data model and apply the idea to process models. The goal is to define a set of primitives for process models that form a super-set of all concepts occuring in process model. Hence, it shall be possible to represent practically any proces model in terms of the generic model and have a uniform representation.

Figure 3 lists the core concepts of the generic model. The central concept is the process step. It can have a state as pre-condition and post-condition. Places in petri nets and events in event-process chains are incarnations of this concept. Furthermore, information objects can serve as input and outputs of process steps. Process steps can be associated to each other by a successor link (one step is successor to another step). Process steps with more than one predecessor are called joins. Steps with more than one successor are calle joins. Synchronized joins are joins that wait for some pre-decessor steps to be completed. A special version is an AND-Op. It waits for all predecessors to be completed. An XOR-Op is a special join step that will wait until the first predecssor has been completed. While still simple, this generic model is expressive enough to capture both the EPC reference model and the more informal enterprise model.

We use a fact quadruple Q(model,source,label,destination) as uniform representation for enterprise and reference models. The first component identified the name of the model, e.g. fig1 for the reference model in figure 1. The remaining three components are virtually like RDF triples. The special label 'type' is used to classify an element into the generic model.

Table 1 shows an excerpt of the fact representation of chunk 1 in figure 1 and the some steps of figure 2. Note that the fact representation uses the same concepts from the generic model. The reference model uses the event-process chain

language to denote process models. This features the concept of events (classified as states in the genric model). The enterprise model of figure 2 uses a simpler notation where process steps follow each other. To make the two modeling languages more comparable, we introduce a logical rule

forall m,p1,p2,s Q(m,p1,post,s1) and Q(m,p2,pre,s) ==> Q(m,p1,successor,p2)

that derives p2 to be sucessor or p1 if there is an state s that is post-condition of p1 and pre-condition of p2.

Table 1: Uniform quadruple representation of enterprise and reference model

Model	**Source**	**Label**	**Destination**
fig1	purchase order created	type	State
fig1	and1	type	AND-Op
fig1	and1	pre	purchase order created
fig1	and1	successor	evaluated receipt settlement
fig1	xor1	type	XOR-Op
fig1	evaluated receipt settlement	type	ProcessStep
fig2	requistion created	type	ProcessStep
fig2	requisition approved and sourced	type	ProcessStep
fig2	requistion created	successor	requisition approved and sourced
fig2	supplier deliveries goods/services	successor	join1
fig2	requestor veriefies info	successor	join1
fig2	join1	type	Join
fig2	join1	successor	receipt evaluated
fig2	voucher generated and payment made	type	ProcessStep

As preparatory step to the actual selection, we split the operation 'voucher generated and payment made' by the operation split into a process step 'generate voucher and a state 'payment made'. This results in the Q facts

fig2	generate voucher	post	payment made
fig2	payment made	type	State

By identifying the state 'payment made' of fig2 with 'payment must be effected' of fig1, we can match fig2 against fig1 after applying `fig1.Remove(chunk3)` and `fig1.Remove(chunk2)`. The XOR operation `xor1` vanishes and the remaining `and1` of fig1 matches against `join1` of fig2.

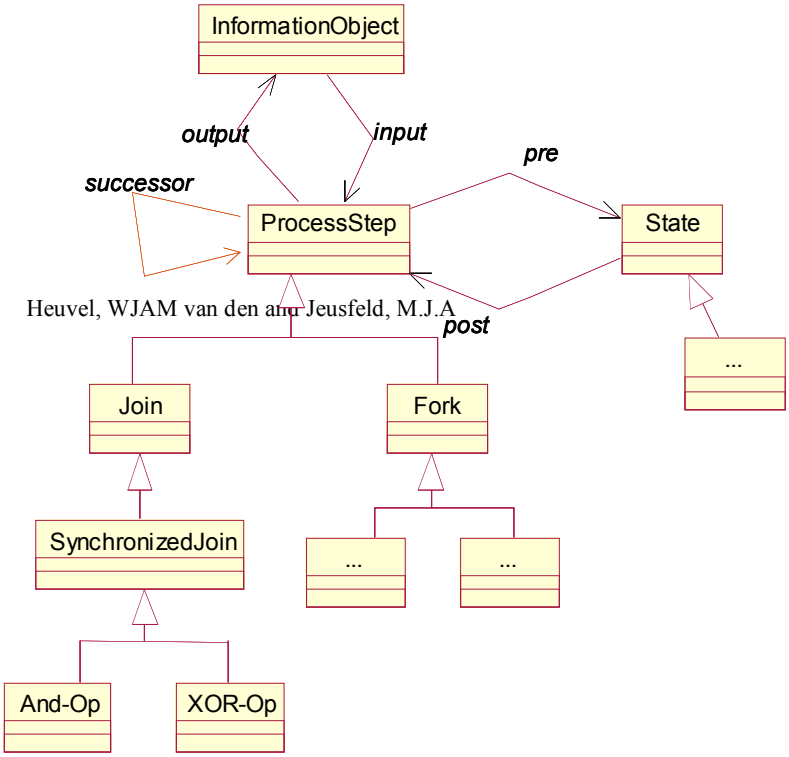

Figure 3 Generic Model for Processes

Now the model primitives, `figure1.Remove(chunk3)AND Remove (chunk2)` are deployed to prune the reference model into a scaled-down version with one invoice processing scenario. Likewise, this procedure may be followed for the other nodes in the models.
In many cases, the compose model operator is used to deal with granularity conflicts, e.g., the activities in the left branch of the enterprise model ("Requisition Created – Supplier Deliveries") were manually combined in an aggregated and renamed concept "Purchase Order Created" to facilitate the selection process.

The result of the selection phase is graphically depicted in Figure 4. The dotted lines in the figure indicate mappings that were constructed based on the results of queries on the enterprise- and reference model and detailed knowledge of the human analyst on the business context. The colored ellipses in the enterprise model denote matched concepts; note that two of these concepts constitute the result of the composition primitive.

Enrichment

After the reference model has been scoped, it may be customized adding enterprise-specific information that was not available in the original reference model. For this purpose, we define the following operators:

1. `Refine(m,n)`: Part m of the reference model is refined by specification n. Refinement implies that m ⊆ m must hold (thus N must be able to replace M without someone noticing it). Refinement may include decomposition of activities in the reference model into sub-activities. This operator may use information that was gained during the selection step, e.g., the "Purchase Order" step in the reference model may be refined into four smaller steps. This decision is left to the human analyst.
2. `Extend():` Add part of model that is not covered in source model. This implies adding extra functionality to the reference model, which was not

foreseen. For example, the extend primitive may be used to extend the reference model in Figure 1 with one additional end state expressing that a voucher will be created.

3. `Choice (x,y)`: a parameter x is chosen from a predefined list of type-level variation points. This operator serves to choose a particular variant at a variation point (e.g., select a discount rate 10% from a given list of discount rates: {10%,20% and 30%}).

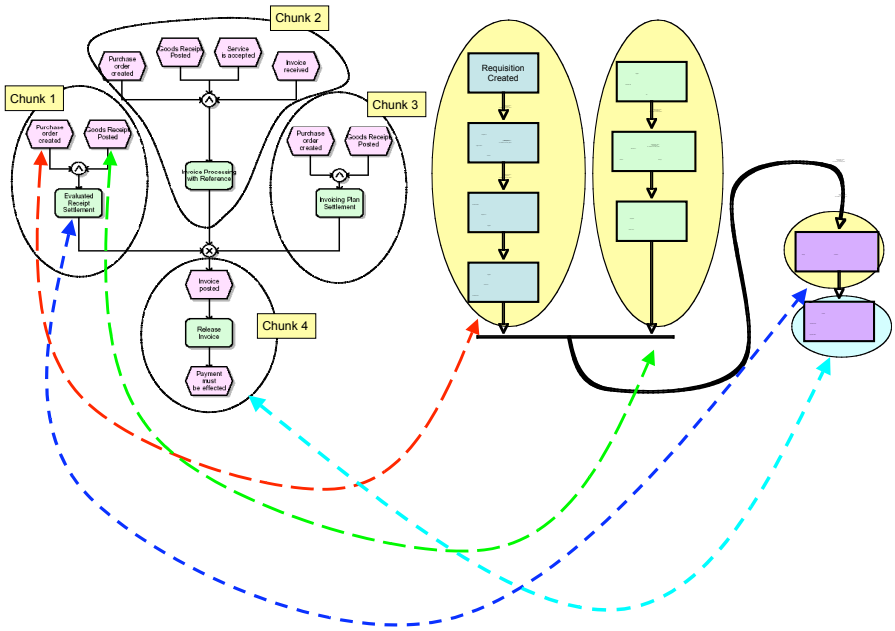

Figure 4 Mappings between Reference Model and Enterprise Model

Hence, these primitives result into a instantiated and extended reference model. In terms of the generic model, this implies that existing facts are specialized or new facts are introduced. For example, the fact Q(generate voucher,successor,release invoice) is added to the knowledge base as a result of execution of the `Extend` primitive. Currently, rudimentary and experimental (ConceptBase) implementations of the above operators have been designed. More research and experimentation is of critical importance before the methodological framework can be used in practice. In the next section, we will review our research results and outline our research roadmap.

DISCUSSION

Existing techniques like graph grammars are compatible to our framework, in particular to express the model transformation operations. The framework that has been introduced in this paper is core research in nature; extensions and refinements

are needed in several directions: (1) Develop a base of reference models and tag them with meta data, in particular specifying the chunks and specifying which chunks are optional, (2) select or develop algorithms for matching source and reference models, (3) generate from the matching algorithm sequences of operations that specialize a given reference model such that the specialization is a close approximation to source model (or source model chunk), (4) Control the order in which source model chunks are matched against reference models. (5) Define rules that control the application of enrichment operations.

REFERENCES

[1] A. Agrawal, G. Karsai, and F. Shi; “A UML-based Graph Transformation Approach for Implementing Domain-Specific Model Transformations”. International Journal on Software and Systems Modeling, 2003.

[2] Thomas A. Curran and Andrew Ladd, “SAP R/3 Business Blueprint”, Prentice Hall, 2000

[3] Blostein D., Schürr A., ”Computing with Graphs and Graph Rewriting”, Technical Report AIB 97-8, Fachgruppe Informatik, RWTH Aachen, Germany.

[4] K. Czarnecki and U. Eisenecker, “Generative Programming: Methods, Tools and Applications”, Addison-Wesley, 2000

[5] S. Sendall and W. Kozaczynski, "Model Transformation - the Heart and Soul of Model-driven Software Development", IEEE Software, Vol. 20, No.5, pp. 42-45, Sept/Oct 2003

[6] "MDA Guide Version 1.0.1", Object Management Group, Document Nr: omg/2003-06-01, June 2003

[7] Tewfik Ziadi and Jean-Marc Jézéquel. -- Families Research Book, chapter Product Line Engineering with the UML: Products Derivation. -- To be published in LNCS. Springer Verlag, 2006

[8] A.W. Scheer, “ARIS- Business Process Frameworks”, Springer, Third Edition, 1999

[9] M. Rosemann and W.M.P. van der Aalst. A Configurable Reference Modelling Language. Information Systems, 2006 (to appear).

[10] A. Kleppe et al., “MDA Explained: Practice and Promise”, Addison-Wesley, 2003

[11] QVT-Partners, First revised submission to QVT RFP, August 2003. OMG document: ad/03-08-08

[12] L. Trat, “The MT model transformation language”, Proceedings of SAC’06, ACM, 2006

[13] J. Mylopoulos et al (1990), "Telos: representing knowledge about information systems”, ACM Transactions on Information Systems (TOIS) , 8 (4): 325-362, 1990

[14] P. Fettke et al., “Business Process Reference Models: Survey and Classification”, In Proceedings of BPM Workshops, 469-483, DBLP/conf/bpm, 2005

[15] Bernstein, P.A., A.Y. Levy, R.A. Pottinger, "A Vision for Management of Complex Models," Microsoft Research Technical Report MSR-TR-2000-53, June 2000, PDF, 179KB (short version in SIGMOD Record 29, 4 (Dec. '00)).

[16] Beatty, R.C. and Williams, G.D., “ERP-II”, Communications of the ACM, 46(3): 105-110, March 2006

[17] D. Kensche, D., Quix, C., Chatti, M.A., Jarke, M., GeRoMe: A Generic Role Based Metamodel for Model Management, Proc. 4th Intl. Conf. on Ontologies, DataBases, and Applications of Semantics (ODBASE), Agia Napa, Cyprus, 2005