Transforming PICTURE to BPMN 2.0 as Part of the Model-driven Development of Electronic Government Systems

Henning Heitkoetter
University of Muenster
heitkoetter@wi.uni-muenster.de

Abstract

The support of public administration processes through information systems is a precondition for electronic government projects. Model-driven development is a promising concept to overcome the challenges associated with developing these systems, for example changing requirements, diverse stakeholders and evolving processes. This paper presents a model-to-model transformation between PICTURE, a domain-specific process modeling language for the public administration sector, and BPMN 2.0. The transformation can be part of the model-driven development of electronic government systems. With the building block-based language PICTURE, administration processes can be modeled in a domain-specific way under involvement of all relevant stakeholders. BPMN, as a generic process modeling language, enables the generation of detailed and executable models from PICTURE models. Besides the individual transformation rules, this paper includes an overview of the implementation with Eclipse and QVT Operational.

1. Introduction

The development of electronic government systems faces a multitude of challenges that are as a whole specific to this domain. Changes in the political and social environment impact electronic government in terms of unstable and gradually evolving requirements [4]. The need to integrate legacy systems and a heterogeneous system landscape further increase the technical complexity [10]. The development should actively involve different stakeholders with diverse backgrounds [16], not only in order to reuse their domain knowledge [15], but also to increase the acceptance and transparency of electronic government [3].

This set of challenges can be addressed by employing model-driven development techniques. Model-driven development (MDD) aims to base the engineering of software systems on models by representing domain concepts in a more straightforward way than implementation-specific artifacts. Automatic transformations derive executable software or software artifacts from models. Advantages include a higher abstraction level and increased flexibility of the development process.

In the context of electronic government, the application of MDD promises to meet several of the challenges outlined above. Models employ a domain-specific language (DSL) that is particularly suitable for expressing the concepts in a concise and unambiguous manner. This simplifies, if not enables, communication between developers and various stakeholders, involving them in the development process. Domain knowledge and requirements are clearly separated from their implementation on a specific platform, allowing greater flexibility with regard to changes in the domain or platform.

Supporting the business processes of public administrations through information systems is essential for the success of electronic government projects [10, 16]. In the context of MDD, process models do not primarily serve analysis purposes, but are used as executable models in workflow management systems or as models for the orchestration of services in a service-oriented architecture.

Business Process Model and Notation (BPMN) is a widely used process modeling language. BPMN models are suited for process analysis or can be executed directly by process engines, provided they are of sufficient detail. While benefitting from a high level of detail and executability, BPMN as a domain-neutral language is not an ideal starting point for a model-driven development workflow and more appropriate as its target, because BPMN models are rather technical and extensive. In order to fully benefit from the advantages of MDD in the context of electronic government, the business processes have to be modeled in a domain-specific way. A domain-specific language for public administration processes is needed for the input models of the MDD process,
so that processes can be modeled intuitively and with an appropriate level of detail. The process modeling language PICTURE [2] was developed to efficiently model the process landscape of a public administration and has been successfully applied in several projects [8]. Predefined types of process building blocks encapsulate complex domain-specific semantics and are the basic construct of this approach. In combination with restricted, but domain-adequate options to express process flow, the PICTURE language hides the complexity from the modeler and from other model users. The information is, however, still available implicitly. A transformation from PICTURE to other process modeling languages can resort to this implicit information and produce more detailed models, enriched with information needed for execution.

This paper explores the feasibility of such a model-to-model transformation and provides a complete and automated PICTURE-to-BPMN transformation. BPMN was chosen as the target language because of its advantages outlined above and its widespread acceptance. Specifically, the new version 2.0 of BPMN was targeted to evaluate its usability in the context of MDD.

The remainder of this paper is structured as follows. In the following section we discuss related work. The subsequent two sections describe the basics of PICTURE and BPMN respectively. Section 5 provides the transformation rules, while section 6 gives an overview of the implementation of the PICTURE-to-BPMN transformation. We conclude with a summary and an outlook.

2. Related Work

A comparison of concepts available in PICTURE and their potential counterparts in BPMN and Event Driven Process Chains (EPC) is presented in [1], where migration paths between these languages are explored. As our focus lies on executable transformation rules and the implementation of the transformation within a MDD workflow, the rules presented here are as detailed as possible and thus differ from the conceptual mappings in [1]. In particular, process building blocks and connections between PICTURE elements are handled differently and with greater specificity. Furthermore, our transformation targets the new version 2.0 of BPMN.

Existing approaches to apply MDD concepts in the domain of electronic government often focus on data aspects and build on data modeling [4]. The application of Model-Driven Integration Engineering in the domain of electronic government [9] is based on process modeling. However, it is restricted to the subdomain registration inside public administrations. The authors employ MDD concepts to integrate existing software systems and generate executable orchestration models in languages like BPEL from domain-specific models. The transformation presented here creates BPMN models and thus can as well be applied to integrate components, for example by means of web service technology. It is, however, not limited to integration scenarios.

Model-to-model transformations involving BPMN mainly use BPMN models as input, mapping them to process execution languages [14, 18]. In [7], a transformation between domain-neutral UML Activity diagrams and BPMN is explored. As far as we know, no transformations targeting version 2.0 of BPMN and its new metamodel have been presented.

3. PICTURE

PICTURE is a domain-specific process modeling language for the sector of public administration [2]. Due to its explicit consideration of interfaces it is particularly suited to model processes that support electronic government. It aims to reduce the complexity associated with process modeling in comparison to general process modeling languages like BPMN or EPC. The different way of modeling with PICTURE intentionally restricts the modeler’s flexibility in order to enhance the comparability and to simplify the modeling process. Nevertheless, PICTURE is at least as expressive for modeling public administration processes as conventional languages, because it is domain-specific.

Process Building Blocks (PBB) are the central model elements in PICTURE. A process building block represents a specified unit of work within a
process. Domain-specific subtypes of building blocks describe typical activities within public administrations, like formal assessment or creation of documents. Each subtype comprises the typical behavior associated with a domain-specific activity. Instances of building blocks in PICTURE models (a-e in Figure 1) can be configured with attributes in order to accurately describe their appearance in the process to model.

A linear sequence of building blocks is the source of a PICTURE subprocess, which represents a delimitable part of the process. However, in order to allow for alternative control flows, PICTURE uses the concept of subprocess variants. Each variant features a complete sequence of building blocks (see variants of subprocess D in Figure 1) and is independent from other variants, except for a common start and end. At runtime, the execution of a subprocess corresponds to the execution of exactly one of its variants.

Building block instances are defined on the subprocess level and shared among variants of their subprocess (see Figure 2). A process building block occurrence denotes that the referenced building block is present in the variant. The predecessor-successor-relationship between building blocks is defined on the occurrence level, as the sequence can differ between variants. For example, building block a of Figure 1 has b as successor in variant 1 and c in variant 2, each defined at their occurrences.

A subprocess can have several predecessors and several successors with parallel semantics: it starts after the completion of all preceding subprocesses. When the subprocess finishes, the process flow splits into one parallel branch for every successor.

Parallel branching and synchronization on the building block level are supported through Anchor Links. A triggering anchor link connects a building block with a subprocess. When the block is executed, the linked subprocess is triggered in parallel. A synchronizing link takes the opposite direction after a preceding trigger. Its target building block waits for the parallel subprocess to complete.

A set of connected subprocesses – by the successor relationship or by links – constitutes a process. Subprocesses without predecessors and incoming links act as the entry points of the process (A-C in Figure 1). All subprocesses reachable from these start subprocesses via the successor relationship form the main flow of the process (A-F). Every triggering link creates a parallel flow, consisting of all subprocesses reachable from the triggered subprocess via the successor relationship (G, H). The process view of PICTURE described above integrates other perspectives, most importantly the organizational view. Subprocesses and building block instances denote an organizational element, e.g. a department or a position, as their performer. Further views handle resources, like information systems, and processed objects (e.g. documents).

4. Business Process Model and Notation

Business Process Model and Notation (BPMN) is a process modeling language standardized by the OMG [11] and widely accepted for the analysis and execution of business processes. For the first time, the new version BPMN 2.0 [12] introduces a complete metamodel of the language. This allows for greater interoperability, as BPMN models can now be exchanged in a defined way. Furthermore, the new specification contains detailed execution semantics, so that BPMN models can include all of the information necessary to be executable by corresponding engines without intermediary steps.

BPMN supports the modeling of different types of business processes. Main element is the process, which describes a sequence of activities carried out within an organization in order to achieve certain objectives. In BPMN, a process is a graph consisting of flow nodes and sequence flow connections as edges. Sequence flow is directed, thereby imposing an ordering on the flow nodes of a process.

The shaded part of the BPMN diagram in Figure 3 depicts a process with different types of flow nodes. The central type of flow node is an activity,
respectively its specializations task and subprocess. A task represents a unit of work, while a subprocess encapsulates another flow graph. Gateways are nodes for controlling the control flow of a process. They are used to split the sequence flow into parallel or exclusive branches as well as to merge such branches. Event flow nodes describe points within a process where either something is triggered by the process (throw event) or the process waits for a trigger to occur (catch event). Dependent on its position in the process, an event can be a start, intermediate or end event. Additionally, events are classified with regard to the type of their trigger, e.g. message, timer or error events.

A business process often consists of several intra-organizational processes combined through the exchange of information, thus forming an inter-organizational business process. In BPMN 2.0, a collaboration diagram like Figure 3 can be used to model these processes. A collaboration features two or more participants, visualized as pools. The process visualized within a participant’s pool describes the activities performed by that participant in the collaboration. A process is contained completely in one pool, as sequence flow must not cross the pool boundaries. The interaction between participants is visualized through message flow. Message flow connects tasks or events in different pools, thereby visualizing the exchange of information between the corresponding participants and processes.

The abstract syntax of BPMN 2.0 as established in the metamodel of the specification (Figure 4) designates the model element Definitions as the root of every BPMN model. Collaboration and Process elements, as subtypes of RootElement, are directly contained in a Definitions object. A process contains flow nodes of different subtypes and connecting Sequence Flow objects combined in a collection of Flow Elements. A collaboration includes Participants and Message Flow elements. Each participant can reference a process from a top level Definitions element. In a diagram representation, such a participant would be depicted as a pool that displays the nodes and edges as described by the participant’s process.

5. Transformation

The development of the transformation between PICTURE and BPMN adhered to the following design principles. The domain-specific behavior embodied in PICTURE models is to be expressed explicitly in BPMN in order to be able to achieve the long term goal of executability and the integration into a MDD workflow. Control flow implicitly present in PICTURE, especially different paths in variants, has to be made explicit in BPMN through the use of sequence and message flow. The transformation further aims to closely follow the BPMN specification to ensure interoperability.

The next subsection outlines the fundamental transformation rules for the central structural elements in PICTURE models (process, subprocess and building block). Sequences of building blocks in consideration of variants are transformed in the second subsection. The third subsection deals with the mapping of connections between subprocesses. The last subsection discusses the mapping of anchor links. Figure 6 shows the BPMN model that results

---

**Figure 3. BPMN model elements.**

**Figure 4. BPMN metamodel (simplified excerpt).**
from transforming the example process from Figure 1 according to the rules of this section.

5.1 Transformation core

Figure 5 provides a schematic overview of the core transformation rules.

A PICTURE process as a combination of subprocesses with different performers corresponds to the concept of a business process in BPMN and is best described as a collaboration. It is transformed to a single BPMN model with a Definitions object as root element and a Collaboration object contained therein. All elements created by the transformation will be stored in this model. Due to this transformation rule, a single BPMN diagram displays all information related to a PICTURE process. A BPMN process is not a suitable mapping for a PICTURE process, because it is intended to model activities within a single organization.

Based on the transformation rule for PICTURE processes, an obvious mapping for the next level in PICTURE models, i.e. subprocesses, appears to be a BPMN process participating in the collaboration. However, because a participant, respectively a pool as its visualization, can only reference and depict one process, every PICTURE subprocess would be represented in its own pool, even if it is performed by the same organizational element as another subprocess. Organizational responsibilities defined in PICTURE would not be visible in BPMN and be lost for subsequent steps in the MDD workflow, e.g. during execution.

Therefore, all subprocesses of the same performer are logically grouped together and as a whole transformed to one BPMN process and one participant referencing that process. The process includes the representation of all subprocesses belonging to the group.

Due to the possible presence of parallel paths in PICTURE, induced by triggering links, the transformation rule needs a minor modification to address the following problem: subprocesses with the same performer can belong to different main or parallel flow. If these subprocesses were mapped to the same BPMN process, this process would contain completely separated and independent paths. For every path, one instance of the process would be created at runtime and this instance would traverse only this path. The process would in fact consist of several separate processes. This characteristic is made explicit by the transformation, as the subprocesses of a PICTURE process are grouped into sets of subprocesses that not only have the same performer, but also belong to the same main or parallel flow. These virtual collections are called Subprocess Group with same Performer (SGWSP).

In the example PICTURE model of Figure 1, three SGWSP exist: subprocesses belonging to the main flow and performed by P1 (A, B, D, E), main flow performed by P2 (C, F) and parallel flow performed by P2 (G, H). Every SGWSP is transformed to a process and a corresponding participant. Thus, the definitions object representing a PICTURE process contains a collection of BPMN processes (one per SGWSP), and the collaboration object contains one participant per SGWSP.

Consequentially, a participant is defined by the combination of the performer and the main or parallel flow, of which its process is a part. Thus, the same organizational element can correspond to more than one participant, if it performs subprocesses of different paths, while the same path of connected subprocesses is spread over several participants, one for each performer present in the path.

The individual subprocesses of a SGWSP are not transformed on their own, but through their building blocks. While it would be possible to transform them to BPMN subprocesses, this would restrict flexibility, as sequence flow may not cross the boundaries of a subprocess in BPMN. The representation of a PICTURE subprocess in BPMN consists of the individual representations of its process building blocks and additional connecting elements (see section 5.3).
Every building block instance in a subprocess is transformed only once, independent of its number of occurrences in variants of the subprocess. All occurrences share the same BPMN representation, because they stand for the same process step and do not differ from each other. The most evident equivalent of a process building block in BPMN is a task, because both represent a unit of work. As the different building block subtypes incorporate complex domain-specific semantics, more detailed transformations are often feasible, possibly using attributes to further tailor the mapping. Therefore, the transformation allows for different building block representations in BPMN by defining few requirements they have to fulfill. Generally speaking, a building block can be mapped to a generic task, a specific type of task (e.g. User Task or Service Task), a subprocess containing a collection of flow elements (graph of flow nodes and sequence flow) or directly to such a collection (not nested in a subprocess). The concrete representations are required to identify flow nodes as connection points for incoming and outgoing connections, so that the transformation is able to integrate them into the process flow. In the following, these points are called in-connector and out-connector, respectively. In case of a single task, that task is both in- and out-connector.

The default transformation rule that maps a process building block to a task can be overridden by concrete process building block types as needed, if the new rule adheres to these requirements. The transformation result in Figure 6 shows several examples for specific mappings.

Instances of the building block type Print Document are mapped to a service task. A service task is a subtype of task representing an activity that uses an external service. In Figure 6, the task labeled “Print” is marked with gears, the symbol for a service task. If a transformation project in a public administration can identify a specific service to be used by all instances of this building block, its details (implementation, address) can be provided to the execution of the transformation as a configuration parameter. The transformation rules integrate this additional information, so that the resulting BPMN model includes these details. A different strategy to refine the BPMN representations produced by the transformation uses the attributes of building blocks.

Figure 6 also shows the special mapping of the building block type Formal Assessment. Instances of this type in PICTURE are mapped to a BPMN subprocess that makes the inherent semantics visible. The transformation uses attributes defined for the type Formal Assessment to adjust the representation of a particular instance. An attribute of Formal Assessment determines which criteria have to be checked. Depending on its value, some paths of the subprocess can be eliminated from the result. If, for example, the responsibility for the incident did not have to be checked in a particular PICTURE process, the corresponding task shown in Figure 6 and its connections would not have been generated.

5.2 Connections between building blocks

The representation of a process building block in BPMN, whether it is a single task or a more complex collection of flow elements, has to be connected to the representations of preceding and succeeding blocks by means of sequence flow. A building block occurrence within a subprocess variant has at most one predecessor and one successor, which are as well of type PBBOccurrence and thus reference a building block on subprocess level. A shared building block potentially has occurrences in different variants,
where it can be succeeded by occurrences referencing different building blocks. Therefore, a building block can have different successors on subprocess level. Under this definition, building block \( a \) of figure 1 has \( b \) and \( c \) as successors. The virtual set of successors of a building block \( B \) is defined as all blocks referenced by an occurrence which succeeds an occurrence of \( B \). If occurrences of \( B \) are followed by an occurrence of the same block in each variant, this successor list has exactly one element, otherwise, it is multi-valued. The same line of reasoning applies to predecessors.

Since the building block has a direct representation in BPMN, but its occurrences have not, these virtual predecessor and successor lists on subprocess level determine the sequence flow to be created in BPMN. If the successor list of a building block \( A \) includes the block \( B \), the BPMN representation of \( A \) has to be connected to the representation of \( B \). Generally speaking, the transformation creates a sequence flow connecting the out-connector of \( A \)'s representation with the in-connector of \( B \)'s representation. Even if \( B \) is the direct successor of \( A \) in more than one variant, only one sequence flow element represents the process flow of all these variants.

Because of the multiplicities outlined above, the representation of a building block may have to be connected to several predecessors and successors and thus may be part of different process paths, as shown in Figure 7. At runtime, only one variant of a subprocess is executed and the process flow follows just the corresponding path. Hence, the paths are exclusive. In BPMN, exclusive branches are modeled via exclusive gateways. If a building block has more than one successor, an exclusive gateway is placed after its representation. The gateway is connected to the out-connector and is the source for sequence flow to successors. The outgoing edges are labeled with the names of all variants they appear in, allowing a clear identification which path is taken in a specific variant.

No special handling is needed in case a building block has more than one predecessor. In BPMN, a gateway is not needed to merge exclusive branches. Instead, sequence flow from predecessors connects directly to the in-connector of the representation of the building block. As only one incoming path is executed at runtime, the BPMN representation is executed only once irrespective of the number of different predecessors.

The transformation rules for predecessor-successor-relationships between building blocks ensure an explicit and correct representation of the process flow inherent in PICTURE models in BPMN. Process parts that are identical across variants are transformed to one and the same collection of BPMN elements. Those parts that differ are clearly identifiable by exclusive gateways and branches. The resulting BPMN model integrates the information that was spread across the variants of a subprocess in a concise manner. The process is modeled as compact as possible and as detailed as necessary.

### 5.3 Connections between subprocesses

The BPMN representation of a PICTURE subprocess, consisting of representations for building blocks and connecting elements, has to be connected to preceding and succeeding subprocesses to express the process flow captured in PICTURE. The transformation distinguishes two types of predecessor and successor relationships: a preceding or succeeding subprocess is either performed by a different performer than the current subprocess (its representation resides in a different pool of the BPMN model), or it is performed by the same performer and resides in the same pool.

A subprocess can be connected in arbitrary combinations of these two categories. It can have several predecessors with the same performer and at the same time several ones with different performers. Subprocesses that have the same performer and are mapped to the same pool are connected with sequence flow, while message flow connects the representation of subprocesses with different performers, because sequence flow may not cross the boundary of pools.

If a subprocess has no direct predecessor with the same performer, it can still have indirect predecessors with the same performer. Consider, for example, the sequence of subprocesses \( C, D, \) and \( F \) in Figure 1, in which performer \( P2 \) executes \( C \) and \( F \), while \( D \) is performed by \( P1 \). \( F \) has no direct predecessor with the same performer \( P2 \); however, \( C \) is an indirect predecessor with the same performer to \( F \). In the BPMN model, the representations of PICTURE subprocesses with the same performer belong as SGWSP to the same BPMN process. Therefore, they should be connected by sequence flow in BPMN, even though they are not directly connected in PICTURE. In summary, in case a subprocess has no
In order to create correct and executable BPMN models, the transformation of subprocess connections needs to create additional BPMN elements besides sequence or message flow. Figure 8 gives an overview over subprocess connections and displays the elements needed in addition to the representation of the internals of a subprocess. Message flow should connect to BPMN elements that are able to process messages, i.e. message events. If necessary, the transformation creates a throw event as the source for message flow to successors in a different pool. The throw event is connected to the representation of the last building block of the subprocess. After the BPMN equivalent of the subprocess has been executed at runtime, the throw event is reached and a message is sent. The message flow targets a message catch event, placed in front of the actual representation of the succeeding subprocess and connected to the representation of the first building block. The event waits for the message to arrive, thereby synchronizing predecessor and successor.

If a subprocess has several successors (predecessors) with a different performer and thus in a different pool, several messages have to be sent (received). A multiple throw event is used in the first case to send messages to all successors concurrently upon activation. A parallel multiple catch event handles the predecessors. All messages are required for the process flow to continue and the event waits until all messages have been received. Figure 8 displays the symbols for these events below the regular message events.

The catch event in front of a subprocess is an intermediate catch event, if the subprocess additionally has predecessors with the same performer. Otherwise, it serves as a start event and initiates a new process instance when triggered. If there are no successors with the same performer, the throw event after the subprocess is an end event.

A single predecessor or successor with the same performer can be connected through a sequence flow. Several predecessors in the same pool require a parallel gateway for synchronization. The gateway is connected to the message catch event, if present, or to the representation of the first building block. The case of several successors does not need special handling with a gateway. Multiple sequence flow diverging from the message throw event or from the representation of the last building block constitute so-called uncontrolled flow. All paths are taken concurrently, leading to the intended parallel branching.

5.4 Transformation of anchor links

Anchor links between process building blocks and subprocesses are transformed into message flow, because they represent the exchange of information between parts of a PICTURE process. From the point of view of a subprocess, an anchor link corresponds to a connection with a subprocess that has a different performer. The message flow representing a triggering link targets the message catch event, while the representation of a synchronizing link originates from the message throw event. Note that these are start and end events respectively, as the starting point of a parallel branch induced by anchor links is not allowed to have a regular predecessor and the end point has no successor. As outlined in section 5.3, these parallel branches lie in different pools than the main flow. The other end of the message flow connects to the representation of the process building block that is referenced by the anchor link, either as source or target.

6. Implementation

The implementation of the PICTURE-to-BPMN transformation is based on technology provided by the Eclipse platform [5]. Eclipse offers a wide range of tools in the context of modeling and MDD and is a prominent platform for MDD. With the meta-metamodel Ecore, the Eclipse Modeling Framework (EMF, [17]) provides a model for describing modeling languages like PICTURE and BPMN. Furthermore, EMF supports the creation and handling of models, which is crucial for the transformation process.
of models in languages described in Ecore by generating an implementation in Java.

The implementation introduced here uses EMF to describe the PICTURE language. The PICTURE metamodel was implemented with Ecore/EMF, so that PICTURE models can be created, manipulated, stored and loaded. An official Eclipse project is implementing the metamodel of BPMN 2.0 with EMF. As the development of the project was still in early stages due to the beta status of the BPMN specification, its implementation had to be extended for the transformation. The focus lay on persisting generated BPMN models in compliance with the XML schema of the specification to ensure interoperability with BPMN tools.

The transformation itself is written in the Operational Mapping language of OMG’s Query/View/Transformation standard [13]. We chose this model-to-model transformation language because it is standardized and its imperative programming style fits in well with the complex transformation rules. The Eclipse project Operational QVT [6] implements this language and uses EMF for input and output models. The transformation picture2bpmn2 reads an EMF-based PICTURE model and transforms this input model into a BPMN 2 model as output. It accesses a library of additional operations defined for PICTURE model elements. These operations provide often needed information, for example the list of successors for a process building block, by deriving it from properties of model elements. The output model is a valid, schema-compliant and thus interoperable BPMN model.

The picture2bpmn2-transformation implements the rules outlined in section 5 and roughly follows the structure presented there. Many mapping operations simply create a new BPMN element corresponding to the particular PICTURE element, set its properties and call the transformation rules for child elements. The transformation of a subprocess is an example for a more complex operation, as it not only initiates the creation of the internal representation but also creates additional elements as described in section 5.3. It returns a collection of flow elements consisting of all flow nodes and connecting elements that form the representation of the subprocess. Similar to this, the operation that transforms a building block takes care of the creation of its representation and generates connections and gateways as per section 5.2. The representation of a building block is specified by three operations. The main operation creates the graph of flow nodes and sequence flow constituting the representation. Two additional operations specify the connectors for incoming and outgoing connections. They each return the single BPMN element from the complete representation that should be the target or source of sequence flow connected to the representation.

In general, a process building block is transformed to a graph merely consisting of a single generic task created by the main operation, that serves as in- and out-connector. Subtypes of PBB are transformed this way by the same operation, unless they explicitly replace the default behavior. A building block subtype that needs a custom representation, e.g. as a subprocess or complex graph of flow elements, overloads the main operation. The overloaded main operation creates the necessary BPMN elements, while the connector operations return those elements of the extended representation that should serve as connectors. The semantics of QVT Operational ensure that the overloaded operations are automatically called instead of the default operations if an instance of the subtype has to be transformed. In case only small modifications to the default behavior are necessary, shortcuts exist, so that duplication of code can be avoided.

7. Conclusion

The PICTURE-to-BPMN transformation presented in this paper serves as a component of an exemplary model-driven development process in the context of electronic government. With the transformation, it is possible to model administration processes in PICTURE and to obtain BPMN 2 models for these processes automatically. This facilitates electronic government by enabling the implementation of supporting processes. Modeling in PICTURE features the advantages of stakeholder involvement, concise and comparable models as well as reduced complexity. On the other hand, BPMN offers an accepted and well-supported process modeling language for analysis and execution purposes. The transformation combines these advantages to some degree, while simplifying the development process and enhancing its flexibility. Therefore, it meets technical and organizational challenges arising in the development of systems that support electronic government.

A key concept of the transformation is the flexible mapping of process building blocks within a fixed framework. It allows transforming the domain-specific behavior encapsulated in PICTURE models into BPMN and at the same time integrates the behavior into a complete process flow. The resulting different representations can additionally be adapted and configured by means of transformation.
parameters and building block attributes. The complex BPMN representation of building blocks reflects the differences between PICTURE and BPMN that make a transformation between these languages worthwhile. Another main component of the transformation is the explicit and complete mapping of control flow that is implicitly present in PICTURE, especially between subprocesses and between building blocks.

The design and implementation of the transformation show that it is possible to represent PICTURE semantics in BPMN. The resulting BPMN models, while lacking the conciseness of domain-specific models, are still understandable and useful for development, analysis and communication purposes. Furthermore, they are generated with the goal of executability in mind. The BPMN models can describe service orchestration and workflow management in arbitrary level of detail dependent on the individual mapping for specific building blocks and on the configuration of the transformation.

The transformation benefitted from the improvements delivered by the new version 2.0 of BPMN, leading to a generally positive evaluation. Its focus on execution semantics was in line with one of the goals of the transformation and had a positive impact as it reduced ambiguities. The metamodel ensures interoperability, so that one can make further use of the generated models. The support by tools and process engines to be expected for BPMN 2.0 should make BPMN suitable for model-driven development projects, as the generated BPMN models can be used for different purposes.

Future work will focus on providing more information in generated BPMN models, especially data that is usable during execution. Two obstacles remain in this context: most attributes of PICTURE elements are aimed at analysis purposes. Therefore, additional attributes that can provide execution information need to be added. Secondly, BPMN has a strong focus on the process view, providing only limited means to model data and hardly any to include organizational information. It has to be explored to what extent these views can be considered during the transformation to BPMN and if otherwise the extension mechanism of BPMN 2 is capable of overcoming the limitations. Further research will explore the possibility of transferring the knowledge gained here to other domains besides electronic government.

References