# Experiences in enhancing existing BPM Tools with BPEL Import and Export

Jan Mendling[1], Kristian Bisgaard Lassen[2], Uwe Zdun[1]

[1] Institute of Information Systems and New Media
Vienna University of Economics and Business Administration
Augasse 2-6, A-1090 Wien, Austria.
{jan.mendling|uwe.zdun}@wu-wien.ac.at
[2] Department of Computer Science, University of Aarhus
IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark.
k.b.lassen@daimi.au.dk

**Abstract.** *The Business Process Execution Language for Web Services (BPEL) has become a de-facto standard for executable process specifications. It is an expressive but also highly complex language. The language is rather targeted towards describing processes at the implementation level and is too low-level for describing processes as analysis models or design models. Also, since BPEL has no formal semantics it is difficult to automate verification of properties, such as deadlock and liveness. Our concept is to transform BPEL to and from other graph-based languages, which are more suitable to support these goals. We discuss transformation strategies as a concept for the transformations, as well as a case study in which we have applied these strategies in an industry project.*

## 1 Introduction

The Business Process Execution Language for Web Services [1] (BPEL4WS or BPEL for short) has become a de-facto standard for executable process specifications. Although the BPEL 2.0 standard is not yet published by OASIS, there are already several systems that support BPEL, including Oracle BPEL Process Manager, IBM Websphere, or the open source system ActiveBPEL. For an overview of currently available BPEL implementations see [8]. This broad industry acceptance forces other workflow and BPM system vendors to consider BPEL support as well.

Basically, tool vendors have two options to approach this challenge: to provide a native BPEL implementation with a corresponding new modeling tool; or, to enhance the existing modeling tool with BPEL import and export. The import/export option might be preferable to vendors for several reasons. First, it is much quicker, easier, and cheaper to be implemented than a native BPEL component. Furthermore, the evolution of the vendor's tool is decoupled from potential modifications of the BPEL standard. Finally, the experiences that went into the tool are a valuable asset for the vendor. Therefore, in the context of an

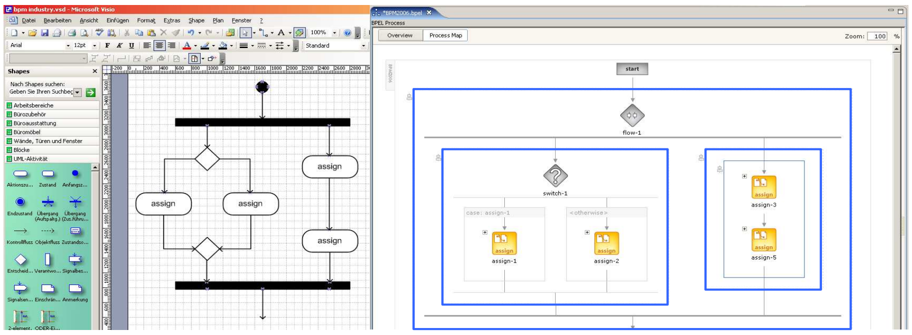

**Fig. 1.** Graph-based modeling with UML Activity Diagrams in MS Visio versus block-based modeling with BPEL in Oracle BPEL Designer

existing industrial tool, it is often not an option to start yet another BPEL standard implementation from scratch. Rather it is desirable to enhance the existing tool with BPEL import and export.

The trade-off of enhancing an existing tool with BPEL import and export is that conceptual mappings have to be identified between the modeling language of the BPM tool and BPEL. In particular, the mapping of control flow is a non-trivial task, especially if the BPM tool uses a graph-based language. Such graph-based languages like EPCs, UML Activity Diagrams, BPMN, or Workflow nets are used by many BPM modeling tools because they are handy in the analysis and design phase of a project. In the following, we will use the term *graph-based BPM tool* to refer to them. On the other hand, processes can be modeled in a block-oriented fashion, similar to process calculi. BPEL is in first place a block-based language, as the control flow can be defined by nesting structures, e.g. sequence, while, or flow structured activities. Yet, BPEL also includes graph-based links that can be used within a flow block. Figure 1 illustrates graph-based versus block-based modeling of processes.

The representational differences of control flow between graph-based BPM tools and rather block-based BPEL is a major problem for the implementation of BPEL import and export interfaces. In the following, we consider control flow transformation strategies as defined in [9] to solve this problem. Section 2 presents available transformation strategies for importing and exporting BPEL from graph-based BPM tools. In Section 3 we present our experiences of a case study where we utilized the transformation strategies for the implementation of an export interface for a commercial graph-based BPM tool. This tool utilizes UML Activity Diagrams in its modeling component. After discussing some related research in Section 4, we give a conclusion of the case study in Section 5.

## 2 Transformation Strategies for Graph-based BPM Tools

In this section we describe transformation strategies for importing from and exporting to BPEL, respectively to and from a graph-based language such as EPCs, BPMN, Workflow nets, and UML Activity Diagrams. Most of these languages support the definition of sub-processes, and we will take advantage of that fact in some transformation strategies. The idea of the strategies is to explicate mapping options between BPEL and graph-based languages and to provide formal algorithms that can be adapted to the specifics of any graph-based language. Formal definitions and algorithms for each strategy are available in [9].

One specific problem of the mapping between BPEL and graph-based languages is that a transformation is not always possible. Some strategies require structural properties of the input format to be satisfied. Table 2 distinguishes structured graphs and acyclic graphs. Essentially, a structured graph uses only control flow patterns that can be mapped to BPEL structured activities. This also includes a simple loop that can be mapped to a BPEL while. An acyclic graph can include any kind of split and join conditions as long as there is no cycle. This implies that an acyclic graph does not need to be structured. Such graphs can always be mapped to a BPEL flow that permits only acyclic links. Furthermore, a BPEL process is structured if it does not include any link elements. Table 2 summarizes for which input the different strategies are applicable. In the following, we briefly describe transformation strategies for the import of BPEL to a graph-based BPM tool (Section 2.2) and for the export of BPEL from (Section 2.1).

| Transformation Strategy from Graph to BPEL | Structured Graph | Acyclic Graph | All Graphs | Transformation Strategy from BPEL to Graph | Structured BPEL | All BPEL |
|---|---|---|---|---|---|---|
| Element-Preservation | - | + | - | Flattening | + | + |
| Element-Minimization | - | + | - | Hierarchy-Preservation | + | - |
| Structure-Identification | + | - | - | Hierarchy-Maximization | + | + |
| Structure-Maximization | + | + | - | | | |

**Table 1.** Transformation strategies and applicable models

### 2.1 Exporting BPEL from a Graph-based BPM Tool

Transformation strategies in this section can be divided into two categories: Either they preserve the graph-based modeling paradigm by mapping to a BPEL flow (Element-Preservation, Element-Minimization) or they map to structured activities whenever possible (Structure-Identification, Structure-Maximization). The general idea of each strategy is illustrated in Figure 2.

*Element-Preservation* This strategy maps all process graph elements to a flow construct and arcs to links. It is a prerequisite of this strategy that the process graph is acyclic. This is because a BPEL flow is not allowed to have cycles

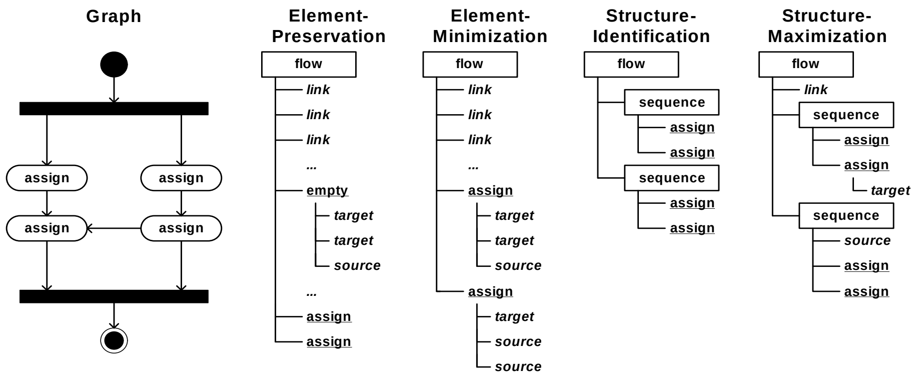

**Fig. 2.** Illustration of Transformation Strategies for Export

defined with links [1]. Routing elements of the graph-based language such as decision nodes and synchronization points are mapped to BPEL empty activities with respective join conditions and links carrying the appropriate split conditions (see Figure 2). The advantage of the Element-Preservation strategy is that it is simple to implement and the resulting BPEL will be very similar to the original process graph since there is a one-to-one correspondence between nodes and activities. As a drawback, the resulting BPEL control flow includes more elements than actually needed: joins and splits are translated to separate empty activities in BPEL although split and join conditions could also be annotated to other activities. Furthermore, the resulting BPEL might be more difficult to read than structured activities, such as sequences. If the BPEL code is used in a scenario where readability is important, then it should be applied only for small process graphs since all elements of the process graph are mapped to one flow construct.

*Element-Minimization* This strategy simplifies the generated BPEL code of the Element-Preservation strategy. The general idea is to remove the empty activities that have been generated from connectors and instead represent splitting behavior by transition conditions of links and joining behavior by join conditions of subsequent activities. As a prerequisite the process graph needs to be acyclic in order to make dead path elimination of BPEL work. The advantage of the resulting BPEL process is that it follows the semantics of the flow construct more closely than the Element-Preservation strategy, since it removes empty activities generated from joins and splits (see Figure 2). As a drawback, it is less intuitive to identify correspondences between the process graph and the generated BPEL specification. This strategy should be used in scenarios where the resulting BPEL code needs to have as few nodes as possible. This might be the case when runtime performance of the BPEL process matters. In contrast to the Element-Preservation strategy, the amount of nodes is decreased since all empty activities translated from connector nodes are skipped.

*Structure-Identification* The general idea of this transformation strategy is to identify structured activities in the process graph and apply structural reduction rules as defined in [9]. As a prerequisite the process graph needs to be structured according to a definition also described in [9]. The advantage of this strategy is that all control flow is translated into structured activities (see Figure 2). With regard to the readability of the resulting code, this is the most suitable strategy since it reveals the structured components of the process graph. As a drawback the relation to the original process graph might not be intuitive to identify. This transformation strategy is appropriate in a scenario when the BPEL should be edited by a BPEL modeling tool such as Oracle BPEL designer that displays the process as a nesting of structured activities.

*Structure-Maximization* The general idea of this strategy is to apply the reduction rules of the Structure-Identification strategy as often as possible to identify a maximum of structure (see Figure 2). The remaining annotated process graph is then translated following the element-preservation or Element-Minimization strategy. The advantage of this strategy is that it can be applied for arbitrary unstructured process graphs as long as its loops can be reduced via the reduction rules defined in [9]. Still this strategy is also not able to translate arbitrary cycles, i.e. cycles with multiple entrance and/or multiple exit points. A drawback of this strategy is that both the Structure-Identification strategy and at least the Element-Preservation strategy need to be implemented. This strategy could be used in scenarios where models have to be edited by a BPEL modeling tool such as Oracle BPEL designer that uses structured activities as the primary modeling paradigm.

### 2.2 Importing BPEL into a Graph-based BPM Tool

Transformation strategies for importing BPEL can be divided into two categories: Either the BPEL structure is transformed into a graph with no hierarchy (Flattening Strategy), or a graph where the BPEL structure is preserved as much as possible (Hierarchy-Preservation, Hierarchy-Maximization). The general idea of each strategy is illustrated in Figure 3.

*Flattening* The general idea of this strategy is to map BPEL structured activities to respective process graph fragments. The nested BPEL control flow then becomes a flat process graph without hierarchy (see Figure 3). For this strategy, there are no prerequisites, both structured and unstructured BPEL control flow can be transformed according to this strategy. The advantage of flattening is that the behavior of the whole BPEL process is mapped to one process graph. Yet, as a drawback the descriptive semantics of structured activities get lost. Such a transformation strategy is useful in a scenario where a BPEL process has to be visually communicated to business analysts.

*Hierarchy-Preservation* This strategy maps each BPEL structured activity to a sub-process in a hierarchy of nested graph-based processes (see Figure 3). The

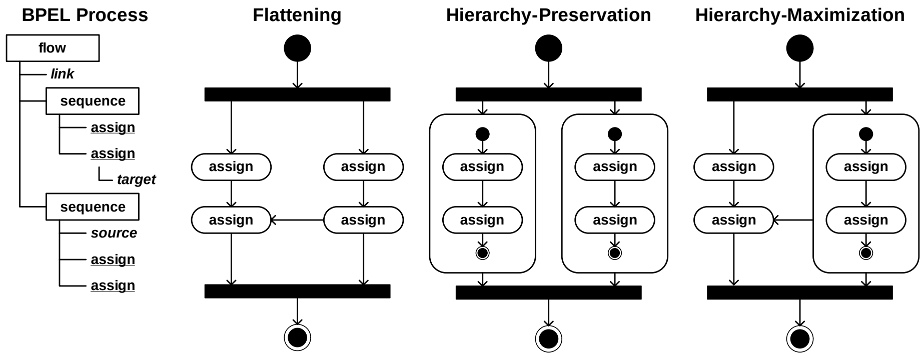

**Fig. 3.** Illustration of Transformation Strategies for Import

nesting of structured activities is preserved as nodes with sub-process relations. The algorithm can be defined in a top-down way similar to the Flattening strategy. Changes have to be defined for the transformation of structured activities as each is mapped to a new process graph. A prerequisite of this strategy is that the BPEL code is structured: links across the border of structured activities cannot the expressed by the subprocess relation. The advantage of this strategy is that the descriptive semantics of structured activities is preserved. Furthermore, such a transformation can correctly map the BPEL semantics of Terminate activities that are nested in Scopes. As a drawback, the model hierarchy has to be navigated in order to understand the whole process. This strategy might be useful in a scenario where process graphs are formally verified and then mapped back to BPEL structured activities.

*Hierarchy-Maximization* One disadvantage of the Hierarchy-Preservation strategy is that it is bound to structured BPEL. The Hierarchy-Maximization strategy aims at preserving as much hierarchy as possible, and it is applicable to any (structured or unstructured) BPEL control flow. This strategy maps BPEL structured activities to sub-processes if there are no links nested that cross the border of the activity (see Figure 3). Accordingly, this strategy does not have any prerequisites regarding the BPEL code structure. The advantage of Hierarchy-Maximization is that as much structure as possible is preserved. Yet, the transformation logic of both previous strategies, Flattening and Hierarchy-Preservation, needs need to be implemented to realize Hierarchy-Maximization.

## 3 Case Study

In an industry project, we designed a BPEL export filter for a workflow designer that uses a graph-based notation based on UML activity diagrams including product-specific extensions. In essence, we followed the element-preservation strategy and deviated in order to capture specifics of the UML Activity Diagram variant of the workflow designer. These deviations related to start and end

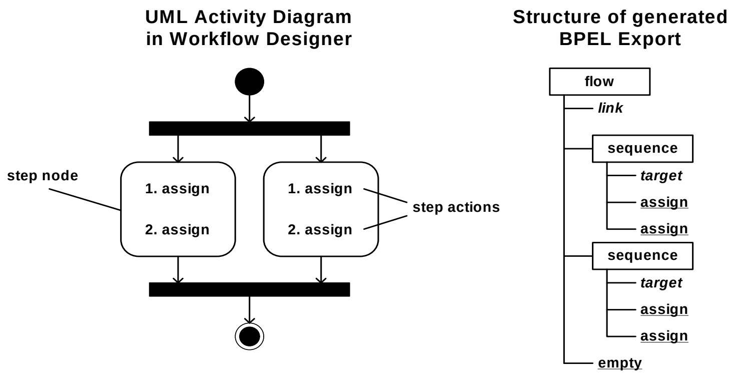
UML Activity Diagram in Workflow Designer
Structure of generated BPEL Export
step node
1. assign
2. assign
step actions
flow
link
sequence
target
assign
empty

**Fig. 4.** Illustration of Mixed Strategy used for the BPM Tool

events, split elements, and a two-level modeling concept. Models built by the workflow designer have exactly one start node and end nodes with implicit termination semantics. As they do not need to be represented in the flow element, we decided not to transform them to BPEL. Accordingly, also arcs connected with start and end nodes are not mapped to BPEL links. The workflow designer offers two split elements that have semantics comparable to an XOR split; these are switch nodes (two alternatives) and decision nodes (multiple alternatives). We decided to map both of these elements to a BPEL switch that includes empty elements for each alternative that serves as a source for a link to the subsequent activity. This design has been chosen instead of a mapping to empty activities in order to easier distinguish different types of splits when the exported BPEL is re-imported. Furthermore, the workflow designer offers a two-level modeling approach: step nodes similar to process graph functions have to be specified by a sequence of one or multiple step actions. Step nodes are part of the UML model, step actions have no visual representation. As a consequence, we map step nodes to BPEL sequences that nest further BPEL activities corresponding to the semantics of the step actions (see Figure 4).

The mapping of many proprietary concepts of the workflow designer turned out to be a problem. These proprietary concepts include sub-workflow elements, step actions, and properties of the indiviual visiual elements:

- Regarding the sub-workflow concept, we decided to map each sub-workflow to a BPEL scope and a nested invoke. This allows us to define the input parameters of the sub-workflow as local variables in the scope and represent the invocation of the sub-process via a BPEL invoke. For a more appropriate mapping, the upcoming BPEL-SPE extension will be very helpful [7]. *U: say why and how it is helpful*
- Step actions are defined in an abstract class, which is customized by a number of different possible step actions, such as defining a (local) variable, inline Java code, or mail sending. To map these steps, we first defined a generic mapping operation to BPEL in the abstract step action class which is used

when no special class overrides the operation. In this case, a BPEL invoke is written to the output, containing the name of the step as a partner link. We also defined mappings for a number of concrete step actions. For instance, in the step action for invoking a form-based input, the partner link is set to the application receiving the form-based input. The inline Java code step action is transformed to a BPELJ snippet [3]. The variable setting step action is mapped to a BPEL assign activity.

– All visual elements of the workflow designer can have additional properties. Some of those, such as time-out conditions and escalations, might even have an influence on the control flow. We defined a special XML namespace for these properties and included them as attributes in the respective BPEL activity. Finally, we had to map step actions contained in the step nodes to BPEL basic activities.

In conclusion, the transformation strategies have helped us to find a systematic, initial approach and process for the transformation of the workflow designer's notation to BPEL. They are also useful for explaining the overall design decisions.

The case study also shows that the transformation strategies can be mixed. The strategies define ideal, prototypical mappings, but in a complex product like the workflow designer in our case study it is necessary to identify the most suitable transformation strategy for the different parts of the mapping.

In addition, in a real-world industry product, there are proprietary extensions, such as step actions or properties in our case, and model elements with further semantics, such as sub-workflows, which are not addressed by the transformation strategies. These require further deviations from the general strategies.

## 4 Related Work

There have been several works on transformations between BPEL and graph-based process modeling languages. We highlight only some of them and refer to [9] for a more comprehensive overview.

The export of BPEL from a graph-based BPM tool can be related to work dedicated to *model-driven development* of executable BPEL process definitions. In [4] a BPM-specific profile of UML is used to generate BPEL code. From the paper the transformation strategy is not *clear was missing ... is it correct?*clear, but the figures suggest that the author uses an Element-Preservation strategy and maps sequences to BPEL sequences.

A conceptual mapping from EPCs to BPEL is presented in [13]. The authors choose a transformation based on the Element-Preservation strategy for the reason that it is easy to implement.

The BPMN specification [12] comes along with a proposal for a mapping to BPEL. As BPMN is a graph-based BPM language, the strategies of Section 2.1 can be applied. The subsection 6.17 of BPMN spec presents a mapping that is close to the Structure-Identification strategy. Yet, the mapping is given rather in prose, a precise algorithm and a definition of required structural properties is missing.

Ouyang et al. [10] show a translation from Standard Process Models (SPMs) [5] to BPEL. They generate what they call Event-Condition-Action (EAC) rules for each activity in the SPM that describes what event must occur under what condition for an activity to become active. Each EAC is translated into BPEL as an event handler resulting in the entire BPEL process being a sequence of event handlers that invoke each other. To improve their result they only make EACs for what they call Clusterable Activity Blocks (CABs), parts of an SPM that among other things do not contain AND-splits and AND-joins. This improves the readability of the resulting BPEL since nodes that are local in CABs are local in the BPEL.

In [11] a Workflow-net-based modeling approach for BPEL including a respective transformation is presented. Similar to the Structure-Identification strategy, Workflow nets are reduced by matching components that are equivalent to BPEL structured activities such as switch and pick. The Structure-Identification strategy has been chosen in order to generate readable BPEL template code and not executable BPEL processes.

## 5 Conclusion

In this paper, we discussed import and export interfaces as a simple option for BPM tool vendors to provide BPEL support. We identified transformation strategies between graph-based BPM tools and BPEL as helpful predefined solutions to the problem of mapping control flow in this context. In a case study we applied transformation strategies in the implementation of an export interface of a commercial BPM tool that utilized UML Activity Diagrams for process modeling. The transformation strategies have helped us to find a systematic, initial approach for the export. Yet, several specifics of the tool required deviations and extensions to the strategies. Some of them are already considered as extensions to the new BPEL Version 2.0 [2]. While we could already utilize the BPEL-J specification for inline Java code, the envisioned BPEL-SPE extension would have been very helpful to map sub-processes. Maybe some of the activity properties like escalation would be considered in the future BPEL4People extension [6]. These extensions have the potential to facilitate a more straight-forward mapping and a simpler interchange of process definitions via BPEL.

## References

1. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Specification, BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems, 2003.
2. Assaf Arkin, Sid Askary, Ben Bloch, Francisco Curbera, Yaron Goland, Neelakantan Kartha, Canyang Kevin Liu, Satish Thatte, Prasad Yendluri, and Alex Yiu.

Web services business process execution language version 2.0. wsbpel-specification-draft-01, OASIS, September 2005.

3. Michael Blow, Yaron Goland, Matthias Kloppmann, Frank Leymann, Gerhard Pfau, Dieter Roller, and Michael Rowley. BPELJ: BPEL for Java. Whitepaper, BEA and IBM, 2004.
4. Tracy Gardner. UML Modelling of Automated Business Processes with a Mapping to BPEL4WS. In *Proceedings of the First European Workshop on Object Orientation and Web Services at ECOOP 2003*, 2003.
5. Bartek Kiepuszewski, Arthur H. M. ter Hofstede, and Wil M. P. van der Aalst. Fundamentals of control flow in workflows. *Acta Inf.*, 39(3):143–209, 2003.
6. Matthias Kloppmann, Dieter König, Frank Leymann, Gerhard Pfau, Alan Rickayzen, Claus von Riegen, Patrick Schmidt, and Ivana Trickovic. WS-BPEL Extension for People BPEL4People. Joint white paper, IBM and SAP, July 2005.
7. Matthias Kloppmann, Dieter König, Frank Leymann, Gerhard Pfau, Alan Rickayzen, Claus von Riegen, Patrick Schmidt, and Ivana Trickovic. WS-BPEL Extension for Sub-processes BPEL-SPE. Joint white paper, IBM and SAP, 2005.
8. Dieter König. WS-BPEL Standards Roadmap. Invited Talk at the 3rd GI-Workshop XML4BPM 2006, http://wi.wu-wien.ac.at/˜mendling/XML4BPM2006/WS-BPEL%20Standards.pdf, February 2006.
9. J. Mendling, K. Lassen, and U. Zdun. Transformation strategies between block-oriented and graph-oriented process modelling languages. Technical Report JM-2005-10-10, WU Vienna, http://wi.wu-wien.ac.at/home/mendling/publications/TR05-Strategy.pdf, October 2005.
10. C. Ouyang, M. Dumas, S. Breutel, and A. H.M. ter Hofstede. Translating Standard Process Models to BPEL. In *Proceedings of the 18th International Conference on Advanced Information Systems Engineering (CAiSE)*, LNCS, 2006.
11. Wil M.P. van der Aalst, Jens Bæk Jørgensen, and Kristian Bisgaard Lassen. Let's Go All the Way: From Requirements via Colored Workflow Nets to a BPEL Implementation of a New Bank System. In R. Meersman and Z.Tari, editors, *Proceedings of CoopIS/DOA/ODBASE 2005, Cyprus*, LNCS 3760, pages 22–39, 2005.
12. S. A. White. Business Process Modeling Notation. Specification, BPMI.org, 2004.
13. J. Ziemann and J. Mendling. EPC-Based Modelling of BPEL Processes: a Pragmatic Transformation Approach. In *Proceedings of MITIP 2005, Italy*, 2005.