Institute of Architecture of Application Systems

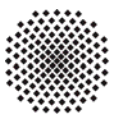
Process Viewing Patterns

David Schumm, Frank Leymann, Alexander Streule

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{schumm, leymann, streule}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings{SchummLS10,
  author    = {David Schumm and Frank Leymann and Alexander Streule},
  title     = {Process Viewing Patterns},
  booktitle = {Proceedings of the 14th IEEE International EDOC Conference,
               EDOC 2010, 25-29 October 2010, Vitória, Brazil},
  year      = {2010},
  pages     = {89--98},
  doi       = {10.1109/EDOC.2010.16},
  publisher = {IEEE Computer Society}
}
```

Universität Stuttgart
Germany

Process Viewing Patterns

David Schumm, Frank Leymann, Alexander Streule
University of Stuttgart,
Institute of Architecture of Application Systems,
70569 Stuttgart, Germany
{ Schumm, Leymann, Streule }@iaas.uni-stuttgart.de

Abstract— **Business processes represent a fundamental asset of a company as they describe the core knowledge underlying its competitive advantage. Tools for modeling and analysis of business processes have to cope with the increasing complexity of these processes. A view on a process intends to abstract from details and make complex processes easier to understand. A process view results from specific transformations applied to a process model. In this paper we introduce a metamodel for process views as well as process viewing patterns which specify elementary transformations to alter an existing process. The patterns are presented in a technology independent manner and can be applied to any process language that can be represented by a process graph.**

Keywords: Process View, Pattern, Model Transformation, Process Analysis.

I. INTRODUCTION

Advances in the principles, methods and tools for creation and management of software are often driven by the necessity of mastering increasing complexity. Several principles have been well established, such as separation of the application from the data it processes (data independence), separation of the application from its presentation and also abstraction of program code in the form of object-orientation. As complexity is still increasing, novel concepts attract more and more interest. The separation of the application functions from the process logic that interconnects them seems to be a promising concept. This separation allows building applications in a loosely coupled, component-oriented manner. Application functions can be bundled and offered as a service, either for internal use or as offer to the outside. This concept shifts the focus from programming in the small towards programming in the large [9] and to the process logic respectively. In recent years, various new terms and approaches have evolved around this topic, subsumed as business process management (BPM) [37] and the technical implementation of business processes, which is referred to as workflows [20]. However, increasing complexity is a problem in this field as well. In practice, business processes may have several hundred activities [35] demanding new methods and concepts for making complex processes still manageable.

Process views, often also called views on processes (or views for short) represent a promising set of approaches addressing this problem. Depending on the problem focus and interpretation of the particular authors, a process view allows an abstraction from undesired details [28], is a separation of concerns [34], or it provides a perspective on a process which is personalized for a specific user [4]. We have discovered that process views serve various further purposes, for instance they can be used for information filtering, information summarization, information hiding or for linking of information to a process. One thing the various approaches on process views have in common is the usage of model transformation techniques [32] and custom-made visualization. Most approaches operate on 'process graphs', where nodes represent activities of a business process and edges represent control dependencies between them.

The meaning of patterns slightly varies in the literature. In [41] architectural patterns have been proposed which discuss problems that occur over and over again in the environment. For each pattern (e.g. "Promenade") the core of the architectural solution is described. Additionally, pictures and diagrams are provided to ease the understanding. Related to programming, patterns usually document programming techniques being described as "simple and elegant solutions to specific problems" [12]. According to [40], a pattern should capture the problem and the solution, as well as the reasons why the solution is applicable. In [16] patterns also cover guidance through the decision-making process for finding an appropriate solution to a particular problem. As patterns are usually elaborated by experienced practitioners, they come with additional information about the motivation for applying a pattern, the consequences when it is applied, practical examples, and an implementation.

In [1] however, patterns have a different meaning. 'Workflow patterns' are used to capture the expressivity of different aspects of a workflow language, in other words which language constructs can be expressed in a particular workflow language. The patterns presented in our work can be seen as a synergy between those different meanings. On the one hand, process viewing patterns describe elementary solutions to simplify the management and analysis of complex business processes. On the other hand, they allow capturing the expressivity of different process view approaches. Furthermore, they enable benchmarking by allowing one to systematically check which patterns or combinations thereof a particular tool supports. For space reasons we refrain from a complex pattern description as it is done for instance in [16].

Although process views are gaining momentum, we think that more work needs to be done concerning the fundamentals of the overall concepts. Therefore, we have assembled the existing concepts and approaches and have

distilled them into a unified and readily understandable representation. The process viewing patterns presented in this paper have their origin spread over various works and tools in which they are *implicitly* applied, i.e. we identified and observed them, but we did not invent them. The circumstance that they are frequently used to solve particular problems qualifies them as meaningful patterns. The relevance of the patterns is confirmed by their actual application in research and practice. We make no claim to provide a complete list of patterns for process viewing. The patterns we describe represent our abstraction from the current literature, product evaluations and our results from own work. Due to space limitations we can neither provide a complete list of literature nor a list of tools that we have evaluated.

We argue that the viewing patterns and the generic metamodel of process views presented in this paper provide basic principles for process view creation. In the following we would like to sketch an example of a process view which can be created using the patterns which we propose in this work. Let us name this example view 'Business Perspective'. This view intends to show a business perspective on a process by leaving out all technical things and reducing the process to its essence.

For creating the view we need to define a few transformation rules. Firstly, we apply some filtering operations to the process. Therefore we *omit* all nodes which are technical, for instance nodes for variable assignment or input data validation. Next, we omit all parts in the process related to exceptions (exceptional paths, fault handling, and compensation handling). In other words we *preserve* those structures which are business relevant, i.e., invocations of programs, human tasks as well as meaningful control structures. Secondly, we apply operations to summarize the information. Therefore, we *aggregate* fully automated structures (e.g., microflows). Depending on the availability of *augmented* (i.e., linked) information, further transformations can be applied. Human-assisted augmentation (i.e., tagging) allows the recognition and subsequently the aggregation of known structures (e.g., an approval chain). Moreover, runtime augmentation allows identifying and subsequently omitting paths which are rarely executed. Thereby, the most frequently taken paths remain in the process. The actual *application* of those transformation rules to a process finally results in the 'Business Perspective'. This view can be presented using a graphical representation, setting the focus on business people's needs and fading out technical details or parts only having an effect in exceptional cases (see Fig. 1). We discuss further application scenarios for process views in [39].

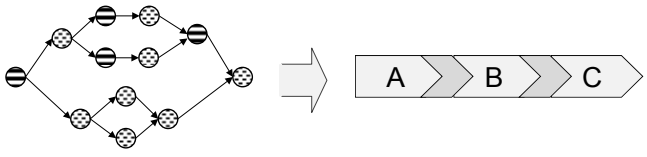


Figure 1. Process View Transformation Example: The Business Perspective.

This paper is organized as follows: Section II specifies a metamodel for use of process views and subsequently describes process viewing patterns. In Section III, application scenarios of the patterns are discussed. Next, Section IV describes an exemplary application of the metamodel and the patterns in the context of a particular process language and viewing purpose. In Section V, state of the art in the field of process views and related work is presented. Finally, Section VI summarizes the paper and characterizes future work.

II. PROCESS VIEWING PATTERNS

In this section, the terminology and metamodel for process views are presented and elementary process viewing patterns are introduced. The elementary process viewing patterns can be composed to obtain more complex transformations. For instance, the composition of the structure omission pattern and the structure preservation pattern described in Section II-B would result in a transformation that could be named 'structure extraction'. We distinguish between four pattern groups that concern transformations of (i.) process structure, (ii.) presentation, (iii.) inter-view relation and (iv.) augmentation. This section does not make any assumptions about the specifics of a process language. The patterns describe the principle, but give no information neither how it can nor how it should be implemented. The description of the patterns is on purpose in an abstract manner, for allowing an easier transfer to different process languages and tools. Beyond that, various different forms of implementation would be conceivable due to distinct characteristics of different process languages and ambiguities that occur during the application of the patterns.

A. Process View Metamodel

A metamodel defines constructs and the associated functions that are supported on them. We distinguish between the terms regular process *R*, original process *O* and process view *V*. A regular process complies with the metamodel of regular processes *Mr*. The metamodel of process views *Mv* is extending the metamodel of regular processes with additional node and edge types which are specific for viewing purposes. An original process *O* represents the process model that is used as input for the transformation *T*, which in turn results in a process view *V*. A regular process model is defined by a tuple

$$R = (Nt, Ec, Ed) \quad (1)$$

where N_t is a set of typed nodes, E_c is a set of control edges that define control dependency between the nodes of N_t. Nodes represent activities that are carried out in a business process. The activity is either executed by a human being ('human task') or by a program (e.g., a Web service). E_d is a set of data edges that define data dependencies between the nodes of N_t. Each edge has exactly one node as source and exactly one node as target. Nodes have properties such as type, identifier and further characteristics which are of minor importance for this work. Edges have properties such as name, source, target, and possibly a transition

condition that defines the condition under which control is passed on. A transition condition is a construct that is present in various process languages and is required for implementing 'dead-path elimination' [20]. The metamodel does not contain concepts which are different across the various process languages (e.g., scoping concepts, fault propagation, event handling, compensation handling). By not assuming a specific model for this, the description of the patterns is made more abstract and easier to transfer to different languages. Those concepts have to be considered when implementing support for the patterns for a specific process language. The metamodel of a process view (see Fig. 2) extends the metamodel of regular processes. It contains all constructs that are required in order to support all process viewing patterns presented in this paper. Thus, if only a subset of the patterns shall be supported, some of the constructs are obsolete. A process model of a process view *V* is defined by a tuple

$$V = (N_t, N_{ab}, N_{ag}, N_i, E_c, E_d, E_i) \quad (2)$$

where N_t, E_c and E_d have the same meaning as in the process model of a regular process. N_{ab} is a set of abstract nodes, i.e. nodes in which all properties are erased. N_{ag} is a set of aggregate nodes, i.e. nodes that consist of multiple nodes and edges, yet treated as atomic. N_i is a set of inserted nodes, i.e. nodes that are not contained in the original process. Inserted nodes can be used as a 'helping' construct, when it is necessary to have a node contained in a process view that is not reflected by any node in the original process. E_i is a set of inserted edges, i.e. control edges that are not contained in the original process. Nodes and edges of M_v are further extensible by an arbitrary set of additional properties. Similar to M_r, all kinds of nodes can be connected to each other with any kind of edges. Depending on the requirements for consistency (see Subsection II-B) fewer restrictions on the metamodel are possible by allowing loose edges, i.e. edges that have either no source or no target that connects them.

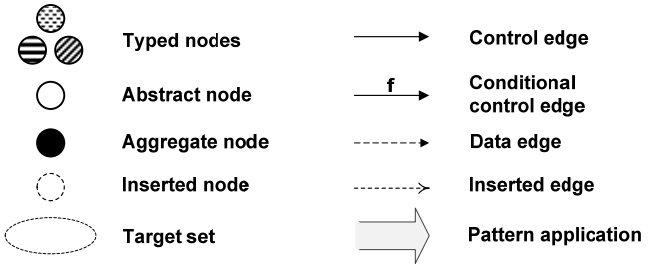


Figure 2. Process View Metamodel Constructs.

A transformation *T* describes the map of an original process *O* into a process view *V*. The original process can either comply to M_r (i.e., be a regular process) or to M_v (i.e., be a process view already), the latter for creating a view on a view. For this reason we distinguish between the terms original process and regular process. A transformation *T* is defined as function

$$T: O \times P \times G \rightarrow V \quad (3)$$

where *O* is the original process model. *P* is an ordered list of transformation step items *I (t, p, c)*, where *t* describes the target set specifying to which constructs in *O* the pattern *p* shall be applied (to which nodes or edges) while using the configuration *c* (e.g., consistency required). The target set *t* can be specified by a query on the process model or on augmented information (see Subsection II-E). The order of the items in *P* is of importance as *T* is non-commutative, i.e. the result of applying patterns in different order can result in different views. For example, if structures are omitted in a first transformation step, then they are not contained in the view which is input for the following transformations. There is a certain difference to Model-driven Architecture (MDA) terminology - in MDA a source model is transformed to a target model. A source model in MDA terminology corresponds to an original process *O* in our conceptual model, and a target model in MDA corresponds to a process view *V*. However, we use the term "target" to indicate the structures in the original process which are to be affected by a transformation. *G* is a set of graphical functions that map a process model to a graphical representation. Basically, *G* represents the presentation layer of a process view. In summary, a process view can be described as presentation of the result from specific transformations applied to a process model. In this sense we understand a process viewing pattern as elementary form of such model transformations.

The graphical functions *G* are separated from the other forms of transformations *P* in order to account for process view composability. The composition of multiple views can be performed by successively applying all view transformations *P* (i.e., transformations concerning structure, augmentation and inter-view), before subsequently presenting the overall result of the transformations. We assume that tools for presentation of a process model have a default set of graphical functions that define how a process has to be visualized. We also assume that we can overwrite these functions with custom visualization functions, e.g., for highlighting particular activities. The graphical functions can thus be composed by subsequently overwriting them in case they overlap. For instance, a view transformation that defines that the border of specific nodes should be painted in green color can be overwritten by another view transformation which defines that some node borders should be painted in red color. The graphical elements which are used in the illustrations of the viewing patterns are shown in Fig. 2. This figure also explains the semantics of shading and different style of control edges and data edges, which are used in the subsequent figures. This is not meant to be a definitive graphical notation for process views. It is used here informally as an aid to ease understanding. For easier referencing, the patterns described in the following are numbered, for instance (P3) stands for pattern number three.

B. Structure Patterns

Structure patterns describe the most common forms of process model transformation. They are destructive, i.e. the process model is physically changed. Therefore, the patterns

are typically applied to a copy of the original process. The patterns refer to the constructs (nodes and edges) contained in the metamodel. The corresponding figures illustrate the application of the patterns on nodes and control edges. The author of [14] even shows that structural view transformations can also be applied on data edges and variables, respectively.

Omission (P1) describes the removal of nodes and edges (see Fig. 3). By omission of a node, omission and re-curving of edges connected to this node are implicitly required. That is why the omission pattern has inherent ambiguities, which we discuss in Section II-F. Single edges can be omitted as well when consistency preservation allows this. Consistency preservation is discussed at the end of this section.

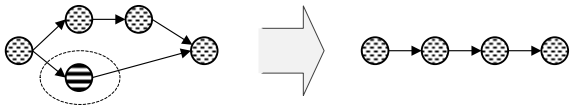


Figure 3. Omission Pattern.

Abstraction (P2) of a node means erasing all properties, including the type (see Fig. 4). An abstract node allows stating in a view that something is happening while hiding on purpose what exactly.

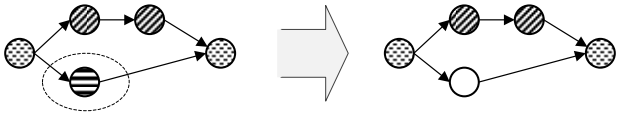


Figure 4. Abstraction Pattern.

Insertion (P3) introduces new (inserted) edges and new (inserted) nodes that are not contained in the original process (see Fig. 5). The application of structure patterns, e.g., omission, requires in some cases adding new nodes or control edges (see also Section II-F). This pattern is necessary for being able to distinguish between original and new nodes and control edges. Besides, inserted nodes allow stating in a view that something shall or may happen which is not happening in the original process model. This is for instance useful in the context of aspect weaving for viewing or planning existent or potential variants of a process model. The pattern for insertion is not common in the field of process views. However, this pattern is frequently applied in approaches that are concerned with the creation and management of constrained process variants, such as described in [22].

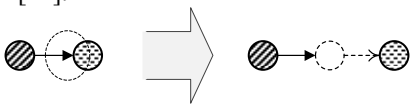


Figure 5. Insertion Pattern.

Aggregation describes the summarization of a set of nodes to a node of higher order, i.e. an aggregate node. All edges entering or leaving this set have their target or source in the aggregate node after the transformation. A difficulty of applying this pattern is how to define the semantics of an aggregate node and how to create a meaningful label for it [15]. For instance, the name of an aggregate can either be calculated by heuristics, by recognition of structures (e.g., using an ontology) or defined manually. We can distinguish between two aggregation forms depending on the nature of the target set *t*.

i.) Connected aggregation (P4) denotes the aggregation of a connected subgraph into an aggregate node (see Fig. 6a). This pattern focuses mainly on (but is not limited to) single entry single exit (SESE) structures, i.e. connected subgraphs with exactly one entry edge and exactly one exit edge. For a formal definition of SESE structures please see [35]. If a subgraph has multiple entries or exits, the outcome of the transformation may contain control cycles which may be prohibited by consistency constraints for a particular process language.

ii.) Disconnected aggregation (P5) describes the aggregation of multiple arbitrary subgraphs into one aggregate node (see Fig. 6b). Depending on consistency requirements the application of this pattern may require adding inserted edges for viewing transitive control dependency. Inserted edges represent control dependency in a view and imply that an edge is not contained in the original process. This pattern reveals ambiguities which we discuss in Section II-F.

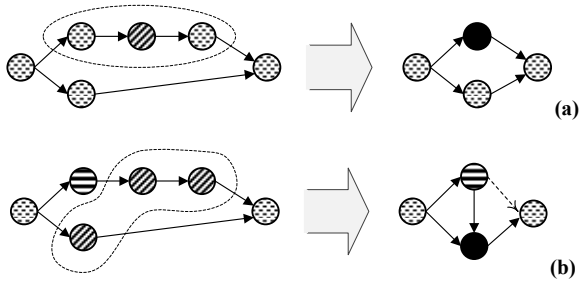


Figure 6. Connected (a) and Disconnected (b) Aggregation Pattern.

Alteration (P6) allows changing properties of nodes and edges, such as type, name, identifier, transition condition or other properties (see Fig. 7). This pattern can be used as fine-granular method for abstraction and information hiding.

Figure 7. Alteration Pattern.

Preservation is a set of patterns that is cross-cutting the patterns concerning structural transformation. Basically, it is part of the configuration *c* which specifies how a pattern should behave. Preservation restricts the structure transformations, i.e., in some situations the patterns cannot or can only partially be applied, as otherwise the outcome would be inconsistent. Implementation of the structure patterns is easiest when no kind of preservation is required. If any kind of preservation is required, complexity of the implementation increases considerably [33]. We distinguish three kinds of preservation patterns (see Fig. 8).

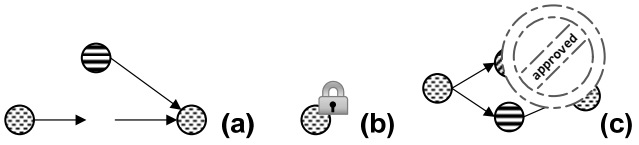


Figure 8. Preservation Patterns.

Consistency preservation (P7) shall ensure consistency of the outcome of a pattern application (compare inconsistency in Fig. 8a). The definition of consistency varies among different process languages, for instance some languages require directed, acyclic graphs (DAGs) for consistency while others also allow non-DAGs. Consistency may also determine the order of nodes or specify whether loose edges and unconnected nodes are allowed.

Construct preservation (P8) has implications on the easiness and flexibility from a user's point of view, as it allows setting constraints to structure transformations by preserving particular constructs (see Fig. 8b). The omission pattern for instance can thereby also be used for extracting parts out of a process.

Executability preservation (P9) is desirable when the process model of the process view shall be executable. This is for instance the case for a view in which all nodes shall be omitted that are relevant for debugging only. Preserving executability includes consistency preservation and furthermore the process model of the view may only contain constructs of M_r. As some patterns introduce constructs that are only contained in M_v, they have to be replaced by constructs of M_r for preserving executability. For instance, an abstract node could be replaced by a regular node that represents a no-operation statement. Fig. 8c shows an icon for this pattern.

C. Presentation Patterns

In contrast to structure patterns, presentation patterns operate on the presentation layer of the process model, i.e. they represent the graphical functions G. Visualization techniques, such as shown in [17], make the process model look differently (e.g., to emphasize specific characteristics or to improve clarity) without physically changing it, i.e., they are non-destructive. Presentation patterns are applied subsequently after all other kinds of patterns. In practice, process languages have many more constructs than shown in the metamodel in Section II-A, e.g., variables, message types and a considerable number of properties on the nodes. Those constructs can be made visible and integrated into the presentation as well.

Appearance (P10) refers to size, color, levels of grey, caption, patterning, decoration, shape, border style, contrast, brightness and transparency of nodes and width, length, color, levels of grey, caption, decoration, style, arrowhead, contrast, brightness and transparency of edges of M_v. Further features are conceivable though. For each of the appearance features a function α can be defined which specifies the mapping of node or edge properties to the presentation characteristics. In former work [31], we have demonstrated a practical implementation of this pattern for visualizing a process with the look and feel of a different language. The example shown in Fig. 9 presents nodes (i.e. activities) executed by humans with bigger size than program calls and in addition decorators are added. Light grey shading represents which of the activities are running in a secured mode. Furthermore, the shapes for the start node (incoming edges = 0) and the end node (outgoing edges = 0) are modified.

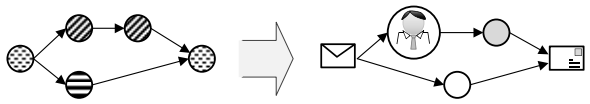


Figure 9. Appearance Pattern.

Scheme (P11) considers the whole representation of a process (see Fig. 10). In many cases a presentation as graph (a) is useful, whereas for some cases a tree (b) might be better, although less expressive. The block scheme (c) is well applicable for block-structured process languages, and the plain textual scheme (d) is indispensable with respect to execution details. In [18] it is pointed out that control flow in block-structured process languages is defined by using block-structures such as 'if' or 'while' for branching and looping, similar to existing programming languages. In contrast, control flow in graph-oriented process languages is explicitly defined through control edges between activities. All those schemes have in common that they still represent the process model. Other schemes such as a performance dashboard or a pie chart abstract from the process model beyond recognition and are thus not listed here.

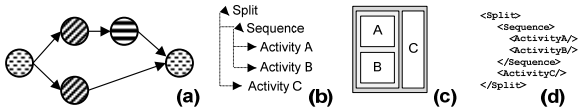


Figure 10. Scheme Patterns.

Layout (P12) describes how to arrange the artifacts of the process model in the particular scheme. For graphs at least three forms are conceivable (see Fig. 11): From left to right layout (a) is frequently used in applications near to business and management, while in technical applications from top to bottom (b) is more common. From right to left (c) could be requested in areas where written language goes also this direction, e.g., in Arabic countries. From bottom to the top (d) is possibly also useful in some scenario, for instance to show a process in a hierarchy.

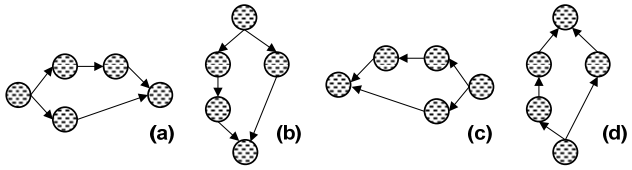


Figure 11. Layout Patterns.

Theme patterns specify the theme of information that should be shown. A process model contains *logical information (P13)* about control flow and data flow. In some process languages data flow is implicitly contained by variables and access to them, nevertheless it can be made

explicitly visible. In some workflow management systems even control flow is implicit, for example in Triana which is data-driven (see http://www.trianacode.org/ for details). This pattern allows visualizing just control flow, just data flow or both (see Fig. 12a) at a time. A process model may also contain *organizational information (P14)*, e.g., the department of a performer of a human task or a particular category of a program that is called in the process. The organizational information can be visualized by categorizing nodes into distinct pools and swim lanes. This can be achieved by binding pools and lanes to particular node attributes. In addition, a process can be further structured into different phases (see phase A, B, C in Fig. 12b). Other forms than pools and swim lanes are conceivable for visualizing *custom categories (P15)*. For instance, when taking augmentation patterns into account (see Section II-E) the location of a process participant (see Fig. 12c) or service deployment information could be used as node positioning or grouping criterion.

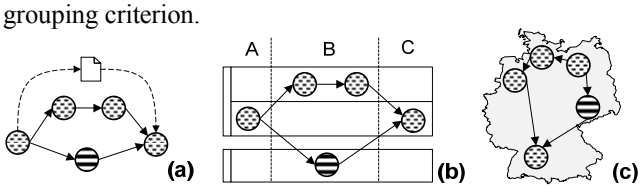


Figure 12. Theme Patterns.

D. Inter-view Patterns

In business process management, the notion of orchestration and choreography is frequently utilized to distinguish between control and data flow within a process (orchestration), and the interaction among processes (choreography). Inter-view patterns address both by abstractly describing operations on process views of orchestrations and choreographies. They have different expressivity depending on the *set-based operations* they support. Therefore, the set-based support for (i.) union, (ii.) intersection and (iii.) complement can be employed as sub-categories of the inter-view patterns.

Orchestration inter-view (P16) describes transformations of multiple process views that share the same original process O. In the literature this pattern is also denoted as view integration [23]. Union combines two process views according to their inherent relation in the original process. Intersection can be seen as lowest common denominator between two process views. Complement creates a process view by omitting all parts of the original process that match the input process view. It is important to note that the original process may be a process view itself. Fig. 13 shows the inter-view union pattern on orchestration level.

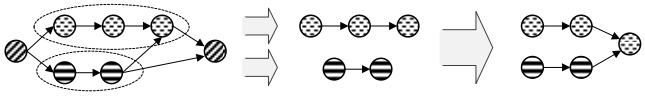


Figure 13. Inter-view Union on Orchestration Level.

Choreography inter-view (P17) represents transformations of process views stemming from different original processes. Fig. 14 shows an example of the inter-view union on choreography level. In terms of process research, union is equivalent to process merging [19]. Intersection describes the effective business protocol between two processes. Taking the complement on choreography level is not applicable.

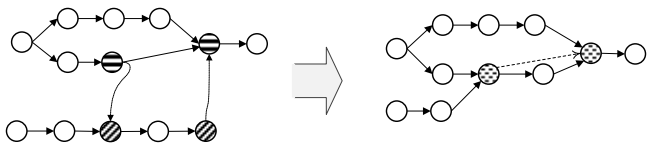


Figure 14. Inter-view Union on Choreography Level.

E. Augmentation Patterns

Besides the information already contained in a process model (nodes, node properties and edges), additional information is helpful for viewing a process tailored to specific needs. We can distinguish three patterns (see Fig. 15) for augmenting a view with additional information.

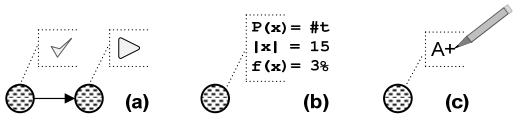


Figure 15. Augmentation Patterns.

Runtime information (P18) describes the augmentation with information related to current or former execution(s) of a process (see Fig. 15a). This may encompass mined information (number of executions, average execution duration, cost, etc.), monitoring information (current status of instance x1, status of all instances of process model x, etc.), organizational details about the performer of a task, deployment information, or context information (workload, service or human geolocation, network traffic, etc.).

Calculated information (P19) considers automatic techniques such as pattern recognition, deadlock detection, particular heuristics and so forth (see Fig. 15b). Technically, the augmentation pattern is likely to be realized by creating a set of references between the node or edge identifier and the identifier of the additional information.

Human-assisted augmentation (P20) addresses human knowledge about the process (see Fig. 15c). This ranges from semantic annotations, the classification of confidentiality levels over organizational information like roles of process participants up to annotation of compliance constraints [8]. These constraints might for instance state for which activities encrypted interaction is required. Our research suggests that especially the usage of ontologies has manifold benefits for fully automated view transformations.

F. Ambiguities

The application of some patterns may comprise ambiguities of the resulting view. Omission, for example, may produce various different views when the omission targets synchronization or branching nodes and control dependency shall be preserved (see Fig. 16a). The problem becomes even harder when conditional control edges have to be handled. The outcome of the transformation is depending

on consistency constraints, user's choice (driven by intended use), or pre-defined settings (i.e. best practice).

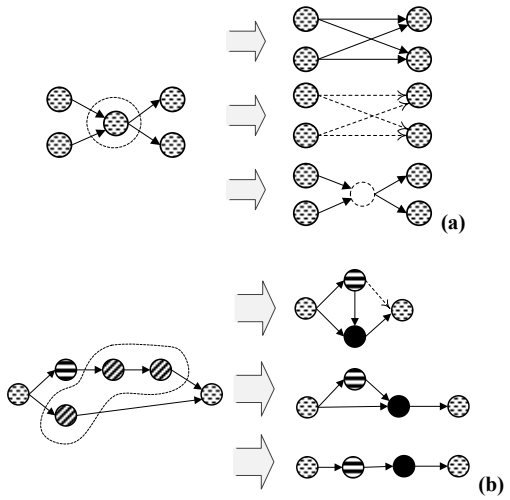


Figure 16. Ambiguous Transformation Results of Omission (a) and Aggregation (b).

The application of the aggregation pattern reveals ambiguities as well, as shown in Fig. 16b. The number of possible solutions increases when conditional control edges are affected, demanding for an efficient method for disambiguation. In addition, the combined application of patterns is problematic: when runtime augmentation for showing the execution state of a process instance (see Fig. 15a) is applied in combination with aggregation (see Fig. 6) or omission (see Fig. 3), it might be unclear how to present the current execution state of the nodes, or require a relaxation of semantics. Another issue arises due to various possible forms of implementation. The inter-view patterns, for instance, give a lot of room for interpretation, and for implementation respectively. Achieving unique transformations across different implementations and therefore defining predictable transformation results is quite improbable as two different implementations (and users) will most likely disambiguate differently. The discussed ambiguities can either be resolved automatically using pre-defined resolution mechanisms or manually by user enquiries.

III. Application Scenarios

The patterns presented in this paper are elementary forms of process view transformations. In the following we present a concrete scenario for their application, namely a process view for process instance management: When it becomes clear that a process instance is "not in good shape", which can for example be measured with Key Performance Indicators, the question "Who is to blame?" arises. Another scenario is that for some cases it is also interesting to know who is in charge of the technical support for a service that faulted in the process. A viewing scenario that provides a solution to this requires first of all augmentation of the process with runtime information (P18) that provides the status of a particular instance. This augmentation step also needs to augment the nodes in a process with the "owners". For service calls an owner is the bound provider as well as the corresponding technical support; for human tasks it is a staff contact and the corresponding supervisor contact (e.g., mail or phone). The presentation of this augmented process shows the status of the instance using decorators (P10). The responsibilities can be shown using swimlanes which are determined by the responsible owners (theme, P14). Advanced applications can provide additional interactive functionality, e.g., when right-clicking a decorator, a menu for notification and escalation could pop up as illustrated in Fig. 17.

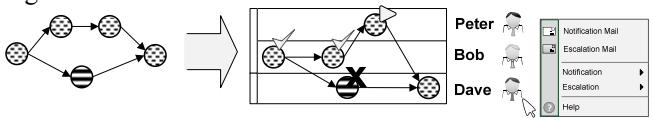


Figure 17. Exemplary Process View: Who is in Charge?

In this paper, we did not explicitly state which of the presented patterns can be used to address which particular problem. In fact, these patterns present solutions for reoccurring problems arising in many different contexts. However, there is no one-to-one relationship between the patterns we proposed and particular problems. The patterns can be combined in many different ways in order to create process views for various different tasks. Especially when augmenting a process model with additional information, it makes sense that the combination of the elementary patterns enables the creation of a flexible and powerful instrument. Let us assume that we define a process view in an ad-hoc manner, in order to find out which activities are most expensive and therefore likely to be outsourced to a company performing them cheaper: In a first step we classify 'internal' activities which cannot be outsourced anyway (human-assisted augmentation, P20), see Fig. 18 (left). Following that, we omit these activities (P1), see Fig. 18 (center). Then, we set the function that calculates the visible size of the nodes (presentation, P10) in proportion to the effective costs of the activities (runtime augmentation, P18). The resulting ad-hoc view (see Fig. 18, right) helps to reveal the activities which are feasible for outsourcing.

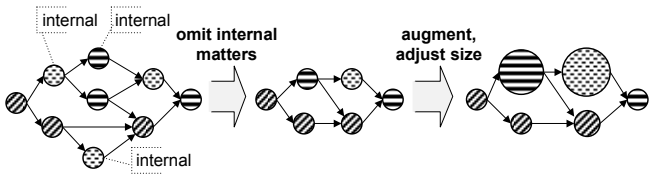


Figure 18. Ad-hoc Process View: Which Activities could be Outsourced?

Many application scenarios for process viewing patterns in business process management are conceivable. Basically, they can be used to support the management of business processes in all stages of the business process life cycle.

In process design, view transformations can be applied to simplify a process by filtering information (omission, P1) or summarizing information (aggregation, P4, P5). In order to

ensure a consistent and well-formed outcome the preservation of particular properties is essential (consistency, P7; construct, P8; executability, P9). Process views can for instance be used for abstract process modeling, for the creation of public views on a private process (see Section IV), and for the extraction of process structures. *In process deployment*, view transformation can simplify the management of configurations by information linking. Deployment configurations (e.g., related to security) are typically separated from the process. An augmentation step (runtime augmentation, P18) can combine this information with the process and present this information to the user (theme, P13). *In process execution*, to the best of our knowledge, view transformations are rarely used at the moment. Structure patterns as well as augmentation patterns can be applied to customize a running instance of a process though. However, the application of these patterns during process execution reveals problems known from process instance migration. *In process monitoring and analysis*, view transformations can be used to link information about the current status of single or multiple instances to the model of the process (runtime augmentation, P18). Additionally, analysis algorithms can provide further information, e.g. about frequently taken paths (calculated augmentation, P19). Presentation patterns (P10-P15) provide a means to visualize this linked information. In addition, *general purpose views* can be composed with other view transformations. This means that they are applicable as views on other views.

In this section we have shown a usage context for some of the patterns we proposed. Furthermore, we have discussed how viewing patterns can be used to provide assistance in the tasks related to process management. For further details concerning application scenarios of process viewing patterns please refer to [39], in which we elaborate known uses of the patterns along the business process management life cycle.

IV. Application Example: Public Views for Business Process Outsourcing

This section discusses an exemplary application of the presented metamodel and the patterns in the context of a particular process language and viewing purpose. The Business Process Execution Language (BPEL) [27] already defines a metamodel for public views, namely 'Abstract BPEL', which is intended for creating templates, and for publishing visible behavior and constraints for valid interactions while hiding complexity and internal information. The metamodel of Abstract BPEL introduces 'opaque' activities and tokens ('`##opaque`') which are in terms of process viewing patterns equivalent to abstract nodes and alteration to pre-defined values. We can utilize a subset of the presented patterns to support this viewing scenario: *omission*, *abstraction*, *alteration*, *consistency preservation*, *construct preservation* and *human-assisted augmentation*. In the work of [33] we created a process viewing framework which supports the generation of Abstract BPEL processes to support business process outsourcing, based on these patterns. In the following we discuss the key aspects of this framework and its relation to the presented metamodel and process viewing patterns.

A. Public View Generation Principle

The generation of a public view for process outsourcing requires two preparation steps before the transformation can be carried out. The first preparation step implements the human-assisted augmentation pattern. In this step constructs (i.e. any artifact of the process model) of the original process *O* are manually tagged. A taxonomy of pre-defined tags allows classifying the confidentiality level of the artifacts in a process (e.g. confidential, private, public). This information can be used to specify the target set *t* in an easy to use manner that requires no in-depth technical knowledge. Thus, every construct which is meant to be affected by this high-level mechanism has to be manually tagged.

The second step comprises the definition of the transformation items *I (t, p, c)* which make up *P*. As a reminder, the target set *t* indicates which constructs should be transformed, the pattern *p* specifies which transformation pattern to apply, and the configuration *c* specifies the settings for the transformation. Eventually, the transformation *T* that implements the patterns takes the tagged original process and the transformation items as input. It applies the particular transformations to the target constructs as specified in the transformation items and thereby creates a process view *V*. For instance, the transformation items could state that all constructs which have been tagged as confidential or private (P20) have to be omitted (P1) during the view transformation. The resulting view hides internal information and can be shared with a business partner. For each business partner a different public view can be generated which only contains the parts which are relevant for that particular partner.

B. Process Model Transformation

The transformation step evaluates the transformation items and performs the requested transformation functions, i.e., it applies the patterns to the targeted parts of the input process. A frequent requirement is the alteration of attributes of constructs. For instance, the name of an activity can reveal information (e.g. about the organizational structure of the company) which should be hidden from a business partner. Our implementation supports the alteration pattern for fine-granular modification of attributes or values of attributes. Furthermore, the BPEL specification supports so-called opaque tokens. The value '`##opaque`' can therefore be used in alteration to hide only the value of an attribute but not the attribute definition itself. Our implementation also accounts for the abstraction pattern by transforming regular activities into so-called opaque activities, when requested in the transformation items (e.g. to show a business partner that "something" happens). For obtaining clean processes cleaning functions are necessary. Probably some constructs contained in an abstract view do not serve any purpose. For example, some variables might no longer be required because corresponding activities have been omitted or abstracted. Our implementation foresees a set of pre-defined post-processing functions to remove such constructs. In [33] we discuss more technical details and show a full view generation in a travel agency scenario. There a travel agency generates different public views for each of its business

partners (hotels, airlines). Our prototype is based on the open source modeling tool Eclipse BPEL Designer [10]. The code changes for supporting 'Abstract BPEL' have already been contributed to the community.

V. Related Work

The notion of process views has originally been introduced for the management of software processes [7]. Due to the growing acceptance of business process management, nowadays a large body of works considering process views exists. Process views are applied for several use cases, such as for providing a perspective on a process that is personalized for specific needs of a user [4]. The authors of [2] show an application of process views in a mobile communication scenario. They propose using views for providing alternative presentations, data views, and customized processes depending on the target mobile device. In proposing 'architectural views' for the separation of concerns, [34] introduces process views in a manner quite different to other approaches, which typically make use of omission and aggregation of the nodes in a process graph. Process views for inter-organizational collaboration are investigated in [5]. A cross-cutting problem when creating a view is the preservation of consistency during transformation. The authors of [21] for example elaborate on preservation of activity order. Recently, also automatic techniques for targeting of transformations are being investigated, as shown in [28]. Commercial vendors offer support for views as well: [30] for instance provides views on functions, organization, data and control. In [6] basic classifications of process views have been proposed. Most notably in this work is the distinction between ready-only and updateable views, and between intra-process and inter-process views. Regarding the classification of process views also works related to model transformation in general are relevant. In particular, taxonomies for model transformations (such as discussed in [42] or [36]) can be used to classify a view transformation, e.g. to state whether a process view can be created automatically or if user intervention is required.

Concepts of views are investigated for a growing number of process languages. In [26] they are applied to the block-structured parts of the Business Process Execution Language (BPEL) [27], whereas in [13] viewing concepts are transferred to Event-driven Process Chains (EPC). The application of viewing concepts for managing access control within process diagrams in the Business Process Modeling Notation (BPMN) [24] is presented in [6]. In [3] views are applied on software processes represented by Petri Nets. In this work the authors generate reusable building blocks using the patterns for omission and inter-view union on orchestration level. The authors of [11] apply process views for generating business protocols for block-structured process languages (similar to UML Activity Diagrams).

Other fields of research such as visualization techniques, especially process visualization techniques [17], are slightly moving towards the concept of process views. The community around generic graph transformation is gradually stepping up to process views as well [25]. Mining techniques, such as shown in [38], also provide a source of ideas for this field of research by identifying recurring templates of control flow. Further reading can be found in analyses of process views [29], [43]. In addition, an analogy to views in data bases is drawn in [23].

VI. Conclusion and Future Work

The patterns presented in this paper describe elementary forms of process model transformations. The fundamental contributions of this work comprise an expressive metamodel for process views and a clear illustration of the elementary process viewing patterns. Moreover, we have discussed application scenarios and we have exemplified a mapping of patterns and the metamodel to a concrete process language. The abstraction into patterns makes the discussion of the features of a process view approach easier and it makes the specification of requirements for an implementation more precise. It also allows characterizing the expressivity of an approach or benchmarking of a tool.

A central problem in our research is how to handle compliance [8]. Managing compliance requires performing profound and traceable changes on process models and providing an according visualization for process management and auditing reasons. Therefore, we investigate methods and concepts for extracting, highlighting and fading out particular parts of a process that are subject to compliance. Process viewing patterns provide the fundamentals for this task.

In this work, we have shown the application of process viewing patterns to process graphs and have demonstrated their feasibility in practice (BPEL-based transformation). In our future work we are going to investigate the use of the presented patterns and combinations thereof in business scenarios concerning process analysis, modeling, deployment, and monitoring. Further usage scenarios like the analysis of service networks or other graph-based applications are conceivable as well.

Acknowledgment

The work published in this article was partially funded by the COMPAS project (FP7-215175, www.compas-ict.eu) under the EU 7th Framework Programme ICT Objective. Many thanks go to Tobias Anstett, Oliver Kopp and Mirko Sonntag for the helpful discussions.

References

[1] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, A. Barros. Workflow Patterns. In: Distributed and Parallel Databases, 14(1):5–51, Springer, 2003.

[2] D.K.W. Chiu, S.C. Cheung, E. Kafeza, H.F. Leung. A Three-Tier View-Based Methodology for M-Services Adaptation. In: IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, IEEE Systems, Man, and Cybernetics Society, 2003.

[3] D. Avrilionis, P.Y. Cunin, C. Fernström: OPSIS: a View Mechanism for Software Processes which Supports their Evolution and Reuse. Proceedings of the 18th International Conference on Software Engineering (ICSE), IEEE Computer Society, 1996.

[4] R. Bobrik, T. Bauer, M. Reichert: Proviado - Personalized and Configurable Visualizations of Business Processes. Proceedings of the 7th International Conference on Electronic Commerce and Web Technologies (EC-Web), Springer, 2006.

[5] I. Chebbi, S. Dustdar, S. Tata. The View-based Approach to Dynamic Inter-organizational Workflow Cooperation. In: Data & Knowledge Engineering, 56(2):139–173, Elsevier, 2006.

[6] M. Chinosi. Representing Business Processes: Conceptual Model and Design Methodology. PhD Thesis, Università degli studi dell'Insubria, Varese, Italy, 2008. http://www.dicom.uninsubria.it/~michele.chinosi/files/PhD/MicheleChinosi_PhD.pdf

[7] W. Deiters, V. Gruhn, H. Weber: Software Process Evolution in Melmac. In: The Impact of CASE Technology on Software Processes. pp. 301-326. World Scientific, 1994.

[8] F. Daniel, F. Casati, V. D'Andrea, S. Strauch, D. Schumm, F. Leymann, E. Mulo, U. Zdun, S. Dustdar, S. Sebahi, F. de Marchi, M.S. Hacid. Business Compliance Governance in Service-Oriented Architectures. Proceedings of the 23rd IEEE International Conference on Advanced Information Networking and Applications (AINA), IEEE Press, 2009.

[9] F. DeRemer, H. Kron. Programming in the Large versus Programming in the Small. Proceedings of the International Conference on Reliable Software, ACM New York, 1975.

[10] Eclipse BPEL Project. Eclipse BPEL Designer, 2010. http://www.eclipse.org/bpel/

[11] R. Eshuis, P. Grefen. Constructing Customized Process Views. In: Data & Knowledge Engineering, 64(2):419– 438, Elsevier, 2008.

[12] E. Gamma, R. Helm, R. Johnson. Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley Longman, 1995.

[13] F. Gottschalk, M. Rosemann, W. van der Aalst. My Own Process: Providing Dedicated Views on EPCs. In: EPK, pages 156–175, Gesellschaft für Informatik, 2005.

[14] R. Bobrik: Konfigurierbare Visualisierung komplexer Prozessmodelle. PhD thesis, University of Ulm, 2008.

[15] O. Maqbool, H. Babri: Automated Software Clustering: An Insight using Cluster Labels, In: The Journal of Systems and Software, 79 (2006):1632–1648, Elsevier, 2006.

[16] G. Hohpe, B. Woolf. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Longman, 2003.

[17] S. Jablonski, M. Goetz. Perspective Oriented Business Process Visualization. BPM Workshops: Workshop on Business Process Design (BPD), Springer, 2007.

[18] O. Kopp, D. Martin, D. Wutke, F. Leymann. The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages. In: Enterprise Modelling and Information Systems. Vol. 4(1), Gesellschaft für Informatik, 2009.

[19] J. Küster, J. Koehler, K. Ryndina: Improving Business Process Models with Reference Models in Business-Driven Development. BPM Workshops: 2nd International Workshop on Business Process Design (BPD), Springer, 2006.

[20] F. Leymann, D. Roller. Production Workflow – Concepts and Techniques. Prentice Hall, 2000.

[21] D. Liu, M. Shen. Workflow Modeling for virtual Processes: an Order-preserving Process-view Approach. In: Information Systems, 28(6):505–532, Elsevier, 2003.

[22] B. Weber, M. Reichert: Refactoring Process Models in Large Process Repositories. Proceedings 20th Intl. Conf. on Advanced Information Systems Engineering (CAiSE'08), Springer, 2008.

[23] J. Mendling, C. Simon. Business Process Design by View Integration. BPM Workshops: Workshop on Business Process Design (BPD), Springer, 2006.

[24] Object Management Group (OMG). Business Process Modeling Notation Version 1.2. OMG Standard, 2009.

[25] 8th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT), pre-proceedings, EASST, 2009.

[26] X. Zhao, C. Liu, W. Sadiq, M. Kowalkiewicz, S. Yongchareon. WS-BPEL Business Process Abstraction and Concretisation. Proceedings of the 14th International Conference on Database Systems for Advanced Applications, Springer, 2009.

[27] Organization for the Advancement of Structured Information Standards (OASIS). Web Services Business Process Execution Language Version 2.0. OASIS Standard, 2007.

[28] A. Polyvyanyy, S. Smirnov, M. Weske. Process Model Abstraction: A Slider Approach. Proceedings of the 12th IEEE Enterprise Distributed Object Conference (EDOC), IEEE Computer Society, 2008.

[29] S. Rinderle, R. Bobrik, M. Reichert, T. Bauer. Business Process Visualization – Use Cases, Challenges, Solutions. Proceedings of the International Conference on Enterprise Information Systems (ICEIS), pages 204–211, INSTICC Press, 2006.

[30] A.W. Scheer, K. Schneider. ARIS – Architecture of Integrated Information Systems. Springer, 2005.

[31] D. Schumm, D. Karastoyanova, F. Leymann, J. Nitzsche. On Visualizing and Modelling BPEL with BPMN. Grid and Pervasive Computing Workshops: 4th International Workshop on Workflow Management (ICWM), IEEE Press, 2009.

[32] T. Stahl, M. Völter, K. Czarnecki. Model-driven Software Development: Technology, Engineering, Management. John Wiley & Sons, 2006.

[33] A. Streule. Abstract Views on BPEL Processes. Diploma thesis no. 2889, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2009. ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/DIP-2889/DIP-2889.pdf

[34] H. Tran, U. Zdun, S. Dustdar. View-based and Model-driven Approach for Reducing the Development Complexity in Process-driven SOA. Proceedings of the International Working Conference on Business Process and Services Computing (BPSC), pages 105-124, Gesellschaft für Informatik, 2007.

[35] J. Vanhatalo, H. Völzer, F. Leymann. Faster and more focused Control-flow Analysis for Business Process Models though SESE Decomposition. Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC), Springer, 2007.

[36] K. Czarnecki, S. Helsen. Classification of Model Transformation Approaches. Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, Anaheim, 2003.

[37] M. Weske: Business Process Management: Concepts, Languages, Architectures. Springer, 2007.

[38] C. Günther, W. van der Aalst. Fuzzy Mining – Adaptive Process Simplification Based on Multi-Perspective Metrics. Proceedings of the 5th International Conference on Business Process Management (BPM), Springer, 2007.

[39] D. Schumm, T. Anstett, F. Leymann, D. Schleicher. Applicability of Process Viewing Patterns in Business Process Management. Proceedings of the International Workshop on Models and Model-driven Methods for Service Engineering (3M4SE 2010), in conjunction with the 14th IEEE International EDOC Conference (EDOC 2010), IEEE Computer Society Press, 2010.

[40] G. Meszaros, J. Doble. MetaPatterns: A Pattern Language for Pattern Writing. Proceedings of the 3rd Pattern Languages of Programming Conference (PLoP), Addison-Wesley, 1996.

[41] C. Alexander, S. Ishikawa, M. Silverstein. A Pattern Language: Towns, Buildings, Construction. Oxford University Press, USA, 1997.

[42] T. Mens, P. van Gorp. A Taxonomy of Model Transformations. In Electronic Notes in Theoretical Computer Science, 152:125–142, Elsevier, 2006.

[43] M. Vasko, S. Dustdar. A View Based Analysis of Workflow Modeling Languages. Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP) pages 293–300, IEEE Computer Society, 2006.