Composite Process View Transformation

David Schumm[1], Jiayang Cai[1], Christoph Fehling[1], Dimka Karastoyanova[1], Frank Leymann[1], and Monika Weidmann[2]

[1] Institute of Architecture of Application Systems, University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
{Schumm,Fehling,Karastoyanova,Leymann}@iaas.uni-stuttgart.de
[2] Fraunhofer-Institut für Arbeitswirtschaft und Organisation (IAO), Competence Center Electronic Business, 70569 Stuttgart, Germany
Monika.Weidmann@iao.fraunhofer.de

Abstract. The increasing complexity of processes used for design and execution of critical business activities demands novel techniques and technologies. Process viewing techniques have been proposed as means to abstract from details, summarize and filter out information, and customize the visual appearance of a process to the need of particular stakeholders. However, composition of process view transformations and their provisioning to enable their usage in various scenarios is currently not discussed in research. In this paper, we present a lightweight, service-oriented approach to compose modular process view transformation functions to form complex process view transformations which can be offered as a service. We introduce a concept and an architectural framework to generate process view service compositions automatically with focus on usability. Furthermore, we discuss key aspects regarding the realization of the approach as well as different scenarios where process view services and their compositions are needed.

Keywords: Process View, Service Composition, BPM.

1 Introduction

Increasing adoption of Business Process Management (BPM) technologies in industry over the last decade revealed that managing process complexity is a key issue, which needs to be addressed. A large business process may contain hundreds of activities [2], requiring advanced methods and techniques for managing such complexity. Process view transformations have been proposed by various research groups as a means to address this problem. In previous work [4], we have assembled the existing concepts and approaches in the field of process view transformations and distilled them into a unified generic representation in terms of commonly used transformation patterns. As a consequence, we understand a *process view* as the graphical presentation of the result obtained after specific process view transformations have been applied to a process model. The purpose of these transformations is manifold. It ranges from summarizing information in order to reduce complexity, filtering information to abstract from details that are irrelevant for a particular analytical task, translating information to provide a

C. Huemer and T. Setzer (Eds.): EC-Web 2011, LNBIP 85, pp. 52–63, 2011.

perspective for a particular stakeholder, up to linking information to augment a process with related data like runtime information about the execution status.

While algorithms and concepts for process view transformations have been well-established in business process management research [5, 8, 10, 11], there is a lack of investigation of their applicability in practice, their composability, and their integration into given toolsets. We identified approximately 20 different process views so far [4, 6, 7], which provide advanced functions to support process design, process deployment, process monitoring, and process analysis. Based on self-experience as scientific methodology, we observed that these process views have two fundamental aspects in common, which are essential for the work discussed in this paper. The first aspect is that these *process views can be composed* to form complex view transformations. For example, a process can be organized according to the distribution of participants (both human beings and services). This process view can be used as input to another transformation that includes the current status of a particular instance of this process. The output of this transformation can be further transformed to show only the activities which are incomplete. Figure 1 illustrates this composition of process views. The second, fundamental aspect concerns *the way in which process view compositions are defined*: There is little need for complex control constructs like conditions, loops or parallelism. Instead, a sequence of process view transformations typically is being performed, as exemplified in Figure 1. Therefore, we propose defining process view compositions by specifying sequences of service invocations, each representing a particular process view transformation.

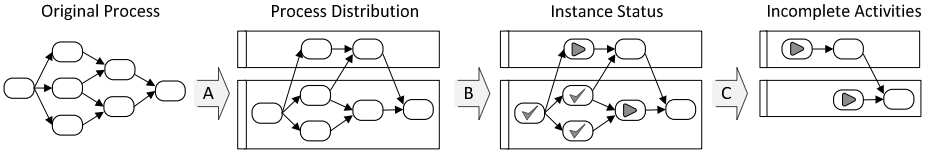

Fig. 1. The result of an exemplary composition of process views: Distribution of participants involved in an input process (A), augmented with the current status of an instance (B), reduced to incomplete activities (C)

The key contribution of this paper is a concept for high-level definition and automatic enactment of service compositions used for composite process view transformation. The concept is intended to empower non-expert users to create pipeline-like service compositions as sequences of service invocations. The approach is to limit the expressiveness of the Business Process Execution Language (BPEL) [1] to a small subset, which allows automatically generating compositions of process view services, out of user-defined composition specifications. Thereby composite process view transformations can be defined that are tailored to the information needs of the different process stakeholders. Moreover, these composites can be provisioned automatically, which is of great advantage. We advance the state of the art regarding the applicability of process view transformation in practice by means of corresponding methods, concepts, and tool support.

The paper's further structure is the following: In Section 2 we introduce a general architecture for service-based composition of process view transformations on a high

level. Based on this architecture, Section 3 describes a detailed walk through the different development stages of process view service compositions. These stages embrace building elementary process view services, defining how to compose them, and generating an executable service composition. We discuss advanced aspects and challenges in Section 4. In Section 5 we point out related work in this field. Section 6 concludes the paper.

2 Architecture for a Process View Management Framework

In this section, we present an architecture for a process view management framework, the platform for composition of process view services. We list contained components, describe their interrelation, and give a brief overview of their functionality and purpose. A walk through the key realization aspects of this architecture can be found in Section 3.

We assume three basic roles we target our framework at. The *Process View Service Developer* is responsible for designing and implementing the core functions of the approach, i.e. the process view services. The *Information Designer* is the user and operator of the process view management framework. He/she registers available process view services and creates meaningful view definitions which describe composite process view transformations on a high level of abstraction. Out of these view definitions, executable service compositions are generated automatically by the framework. The *Process View Consumer* finally uses the (composed) services for the creation of views on concrete processes for his/her particular information needs.

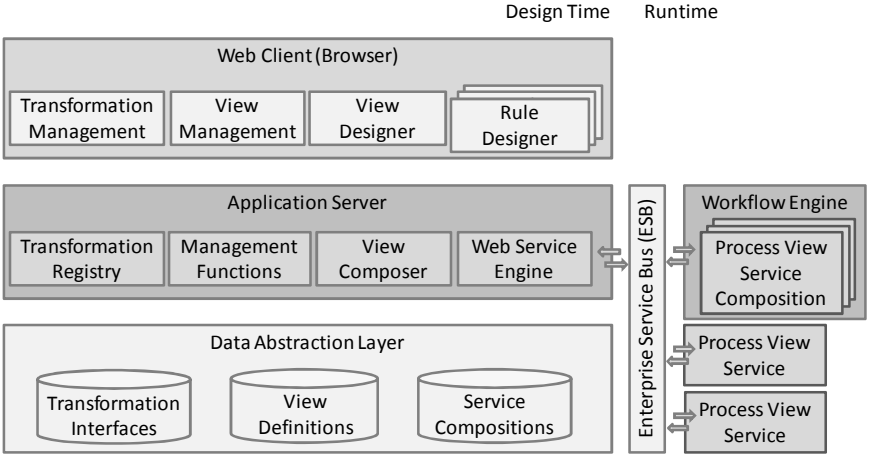

Fig. 2. Architecture for a process view management framework

As shown in Figure 2, we use a three-tier setting for the design time of process view service compositions. Design time is shown on the left part and runtime is shown on the right part of the figure. The upper tier in the design time part of the architecture provides Web-based functions for managing composite process view

transformations. In this tier, the *Transformation Management* provides functions for registering and deregistering *Process View Services* (see the runtime part in Figure 2). The *View Management* provides selection menus for creating, deleting, opening and deploying existing view definitions. It may also provide an interface for invocation of process view service compositions which have been deployed to the workflow engine. A view definition represents a composition of process view services on a high level. This definition is abstract and not executable. A view definition is basically a sequential ordering of selected operations of registered process view services. The *View Designer* is the component which is actually used to design and modify view definitions. As process view services need to be parameterized, a set of *Rule Designers* is required. To support the information designer in coping with a diversity of formats, we propose to use the concept of domain-specific languages (DSL) here.

The middle tier represents the backend. The *Transformation Registry* handles requests related to transformation management, extracts interface information and passes them to the *Data Abstraction Layer*; the *Management Functions* provide analogous functionality for requests related to view definitions; the *View Composer* is one of the core components of the framework, responsible for generating an executable *Process View Service Composition* out of a view definition. This composition orchestrates the core of the approach, the *Process View Services*. We propose the use of BPEL [1] as format for executable service compositions.

The generated service compositions can be stored locally, can be registered as process view services for recursive compositions, and can be deployed using the *Web Service Engine* component. This engine integrates with the Web service interfaces provided by the *Workflow Engine*, shown in the right-hand side in Figure 2. The runtime performs the execution of the generated *Process View Service Compositions*.

3 Key Realization Aspects

In this section, we examine the key aspects of our approach from a realization point of view. These aspects concern the development of process view services (Section 3.1), the creation of view definitions (Section 3.2), and the generation of process view service compositions (Section 3.3).

3.1 Development of Process View Services

Process view services are the components which implement process view transformation functionality. They are exposed to the outside using an interface description language like the Web Services Description Language (WSDL). In the following we abstract from the inner implementation of these services and focus on their exposure to the outside and how to control the transformations they perform.

As proposed in our previous work [4] and depicted in Figure 3, the following terms are essential in process view transformations: The *Original Process* is the process model that is subject to a *View Transformation* which results in a *Process View*. We use the term *Target Set* to indicate the process structures in the input process model which should be affected by an elementary transformation *Action.* The action represents the transformation function to be applied. Examples for such functions as described in [4] are structural transformations (aggregation, omission, alteration,

abstraction, insertion), data augmentation transformations (runtime, human-assisted, calculated), presentation transformations (appearance, scheme, layout, theme), and transformations with multiple input processes.

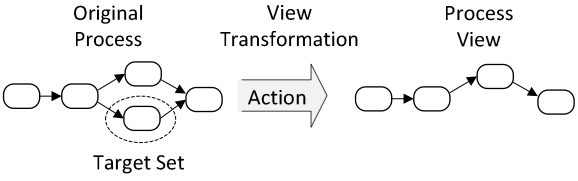

Fig. 3. Process view transformation terminology

Together, a target set and an action make up a *Transformation Rule*. Multiple rules can be applied after one another as in batch processing. For example, a first rule may state to omit all activities for variable assignment. A second rule may state to make service invocation activities "opaque" to state that something is happening at that place, while hiding detailed information. A global *Configuration* is useful to set general parameters valid for all rules. For instance, a parameter in the configuration can switch "executability" on or off. This parameter refers to the preservation of process structures and artifacts that are mandatory for executability, like an instance-creating `<receive>` in BPEL. To support the exposure of process view transformation functionality as a service, as well as to ease their composition, we propose a common structure of transformation instructions as depicted in Figure 4.

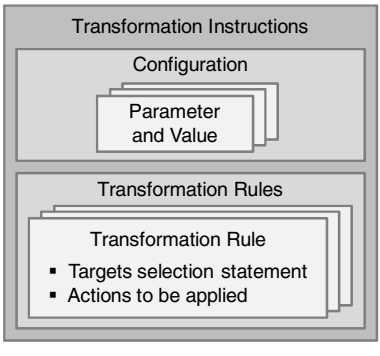

Fig. 4. Common structure of instructions for a process view service

However, our main finding with respect to realization of process view transformations, their exposure as a service, their consumption, and their composition, is that there is no ultimate format or language for describing concrete transformation rules and configuration parameters. Instead, each process view service will likely have a different set of parameters, and will likely use different languages to control the transformation. For example, the target selection statement for a service which removes a process fragment from a given input process will likely be a process

fragment itself, while a process view service that provides general filtering functionality will more likely use regular expressions or SQL-like statements. Also, if the same functionality is offered by different vendors, the parameter formats of services may differ. Furthermore, the vocabulary of transformation actions that can be performed will probably differ. As a conclusion, we argue that the concept of DSLs applies here, so each service may use different formats and types of parameters. The architecture presented in Section 2 considers this conclusion with multiple *Rule Designer* components, generated automatically from the service interface description, or directly provided by the process view service vendor.

3.2 Creation of View Definitions

For the presented approach, the main interest lies in the mere use of service offerings as well as in the ability to create own, custom compositions of available services which are possibly provided by different vendors. Besides the functionality that the service needs to offer, the selection of services can be based on process view transformation quality constraints like guarantee of the executability of the process view, or by cost, processing speed, etc. as described in the vision of Web service ecosystems [15] which makes the notion of service procurement explicit. According to [15], a Web service ecosystem is envisioned as a "logical collection of Web services whose exposure and access are subject to constraints characteristic of business service delivery."

Process view services need to be registered in the process view management framework before they can be used in the definition of process view service compositions. As service registration is a common feature in service-oriented application design, we do not discuss this aspect in detail here. The registration of available process view services and hence the information about their input parameter types allows specifying a composition of these services. Parameter and type information is essential for parameterization and configuration of the process view services on a high level. With the term "View Definition" we denote a quite simple form of such composition, with ease-of-use as focal point. We fundamentally constrain the expressiveness of Web service compositions by only allowing the definition of a linear sequence of process view service invocations. The flexibility we provide is focused on the interconnection of output and input parameters of consecutive service invocations. However, process view service compositions which require complex control structures, cycles, and conditional branches cannot be defined in this high-level manner. For such cases the direct usage of process languages like BPEL without an abstraction level on top as we propose here is one possibility (see also Section 4.1). Nevertheless, a lightweight, pipeline-like composition approach may be beneficial for all those cases in which a linear sequence of service invocations is sufficient. Process code can be generated automatically out of the high-level view definition, which is much easier to create than executable process models.

A view definition can be created by iteratively searching and selecting a registered process view service to be used. From this selection one of the operations offered by that service can be chosen. Thereby, a list of process view service invocations comes into being, see Figure 5 (left). The outputs produced by these process view services can be used as input in subsequent service invocations. Thus, the services can be connected by defining data flow between them.

In the creation of view definitions, we can distinguish *dynamic* and *static* parameters. Dynamic parameters are used to make a view definition (and the resulting process view service composition) configurable. This allows adjusting the behavior of the composite view transformation in each invocation without changing and re-deploying its original definition. In contrast, static parameters are used to define constant settings which are valid for all invocations of the resulting process view service composition. For example (see also Figure 5), the first service invocation may augment a process (provided as dynamic parameter "Original Process") with information related to the recognition of a process fragment that is critical for security (customized through the dynamic "Parameter A"). The second service invocation shall extract this fragment, using static transformation instructions specified in the static "Parameter B". The subsequent service invocation takes the original process and the extracted fragment as input, and produces a process view in which this fragment is omitted. The final service invocation in this exemplary view definition produces an SVG rendering of this process, configured statically with "Parameter C". The output "Process View" is finally returned.

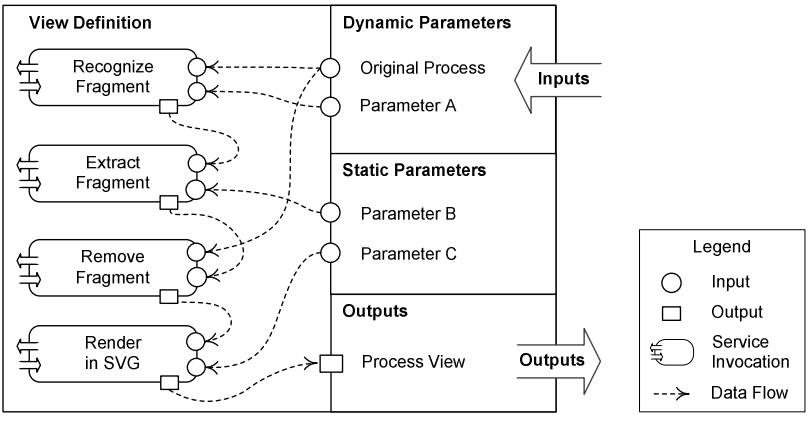

Fig. 5. Definition of a composition of process view services on a high level

Service invocations can be reordered to obtain a view definition that is free of forward dependencies which would make the view definition invalid. When this dependency criterion is met and all service invocation parameters are either connected to dynamic parameters, static parameters, or previous outputs, then this view definition can be used to generate an executable process view service composition.

3.3 Generation of Executable Process View Service Compositions

For the generation of an executable process view service composition several artifacts are necessary. The view definition describes the sequencing of service invocations and the connection of inputs, outputs, and parameters. The WSDL documents of involved process view services contain type definitions and addresses of the services required for execution. Furthermore, as the generated service compositions all have

the same basic structure, a template for the service composition is useful. This template consists of a BPEL process skeleton and a WSDL skeleton. The template is instantiated during the generation of executable code from the view definition. The deployment descriptor, which is also required for execution, is rather dependent on the selected services and therefore needs to be generated dynamically.

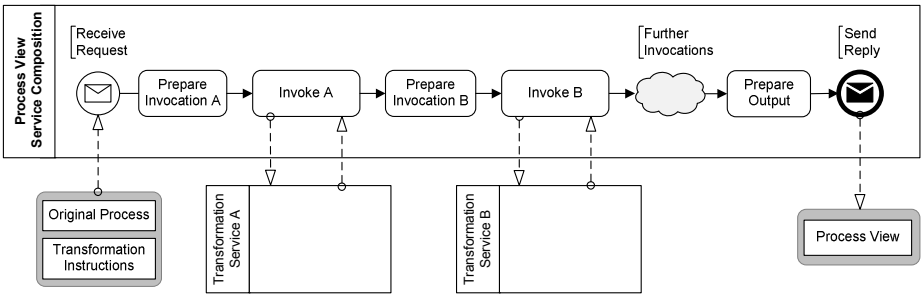

Fig. 6. Generated composition of process view services

The structure of a generated process view service composition is illustrated in Figure 6. In this figure, the Business Process Model and Notation (BPMN) standard [16] is used to visualize the BPEL process, though considering implicit data flow as used in BPEL. The illustration in BPMN is intended to explain the concept, only BPEL code needs to be generated (or: executable BPMN). The process view service composition can be invoked with a request that contains all dynamic parameters, represented by a `<receive createInstance="yes">`. Static parameters are set at process instantiation. For each service invocation specified in the view definition, an `<assign>` activity prepares the input parameters and a subsequent `<invoke>` activity invokes the operation of a service. Finally, the output - the process view - is returned to the service composition consumer by a `<reply>`. Such a generated service composition can be packaged by the *View Composer* component (see Section 2) and be stored in a database, or deployed to a workflow engine to enable execution. A service composition that has been deployed to a workflow engine can also be registered as a new process view service and thus enable recursive compositions.

We developed a prototype of a framework for the management of composite process view transformations on BPEL processes. In comparison to the concept of view definitions presented in Section 3.2, our prototype does not support arbitrary connection of inputs and outputs of services yet. Thus, data mediation is not considered. Currently, one can only configure that a service should use the output produced by the directly prior service invocation as an input for one of its particular parameters. Experiments with our process view services and evaluation of the framework for generating executable service compositions based on BPEL showed that arbitrary connection of output parameters is not necessary in many cases. For instance, the view definition described in Section 3.2 can be implemented that way. Such lightweight compositions can be used to refine a process view step-wise, forming pipeline-like service compositions.

4 Advanced Aspects and Challenges

There are advanced aspects and challenges that need to be addressed before a productive use of (composite) process view services is possible. One aspect is related to the expressiveness of languages involved in the approach (Section 4.1). These languages have significant impact on the flexibility, ease-of-use, and configurability of composite process view transformations. The other aspect we discuss is related to security and privacy (Section 4.2).

4.1 Expressiveness of Involved Languages

Process view services may offer domain-specific languages (DSLs) that allow their parameterization and configuration. To ease usability and to make the approach accessible to a large user group, we also proposed to have a view definition language on top of the execution language which can be used to easily describe sequences of process view service invocations and wire outputs and inputs of these invocations. The question is: How much expressiveness of the involved languages can be provided while still considering ease of use?

Domain-specific languages – The concept of DSLs for parameterizing process view services we presented in Section 3.1 could be extended to provide more flexibility. For example, a rule could conditionally be executed based on the number of activities, control links, or variables contained in an input process. Furthermore, a process view service provider could offer a Web-based rule designer tool to ease the specification of transformation parameters and configuration. Such tools could also be an aid to avoid the definition of inconsistent transformation instructions. A challenge in this lies in the usage of dynamic parameters in a view definition (see Section 3.2). If dynamic parameters have been specified for the view definition, then a new DSL needs to be created to ease invocation of the newly created service composition. This DSL may be composed out of components of the DSLs of the services that are involved in the composition, defining a composition of language profiles.

View definition language – We proposed a view definition to be a sequence of service invocations, where only data flow can be specified in a flexible manner. A major issue in the specification of the data flow is the data mediation that is needed when parts of complex outputs produced by services are used later on as input parameters in invocation of other process view services. To be able to deal with this issue without in-depth technical expertise, a graphical editor is needed to support assigning input and output values. Furthermore, to make the approach more powerful, invocations of process view services could be made conditional, e.g., based on the properties of the process to be transformed. Another feature would be to allow a service invocation to be performed multiple times, for instance invoking an abstraction service until a process contains less than 50 activities. However, if such features are provided there also need to be mechanisms that assure that (i) parameters are properly initialized before any service invocation and (ii) the process is properly routed through the composition, also considering “dead paths” which may arise from conditional service invocations. Standardized process view service parameters which form some kind of basic format for inputs and outputs would make the realization of such features easier.

4.2 Security and Privacy

Well-designed and efficient business processes are an important competitive advantage. Therefore, the corresponding process models are critical intellectual property. If a company uses process view services of third-party providers, business secrets have to be protected. In the following, we discuss three methods to secure the invocation of third-party process view services by (i) hosting them in secure environments, (ii) obfuscation of business process models, and (iii) establishment of a trust relationship between process view service providers and the company using them.

Hosting in secure environments – providing a process view service as an installable package, for example on a CD, allows hosting the service in a private, secure environment. However, service users have to invest in licenses upfront and have to manage updates and patches. Especially, if the service is used seldom, on-demand access and pay-per-use is more desirable.

Obfuscation of business process models – prior to sending process models to insecure process view services, other, internal process view services could be used for process model obfuscation. For example, activity names can be replaced with random identifiers, additional activities and control flow can be added etc. After transformation, an internal deobfuscation service needs to be invoked. This approach can be employed to securely use untrustworthy services. A shortcoming is that it is limited to view transformations that do not require information about process model semantics. Examples for transformations applicable for this approach are aggregation of sequential activities or filtering of particular activity types.

Establishing trust relationships – trust relationships can be established through contracts making providers liable to ensure a certain degree of privacy and security [14]. This method is most likely to be used in practice.

5 Related Work

From academia, significant progress has been made in the field of process views. Process views are applied to various different languages like Event-driven Process Chains (EPC), Petri Nets, BPMN [16], and also to the BPEL [1]. Typically, scientific works on process views concentrate on one particular application scenario. For instance, in [5] process views are used to support service outsourcing by generating "public views". The work presented in [12] focuses on aggregation of activities by making use of part-of relations between activities. A work by Reichert et al. [13] discusses the application of process views to provide access control for process models. In Web service environments, process views can be applied to simplify Web service orchestrations specified in BPEL by omission of activities and aggregation of structures of a BPEL process, as discussed for example in [8]. Our own process view implementations also operate on BPEL processes – in [7] we proposed a process view to remove or extract process structures. However, composition of process views and their provisioning as a service is currently not discussed in research. We argue that all these mentioned process view approaches are well applicable for usage as a software service in the manner we proposed. For instance, a generated public view on a process can subsequently be transformed with advanced aggregation techniques.

Most of the approaches proposed in the field of process views so far have their focus on structural changes of a process model. Recently, graphical aspects and process appearance are taken more and more into account in order to create perspectives which are tailored to the needs of particular stakeholders and scenarios. In this manner, the authors of [17] distinguish between the concrete syntax (appearance) and the abstract syntax (structure) of a process. They argue that changes of the concrete syntax are well-suited to cope with the increasing complexity of process models. Their findings build on literature study, tool and language evaluation, and remarkably, on works related to human perception such as [18]. However, further research is required to cover all aspects of a service-based composition of functions that especially provide transformations of the concrete syntax.

Regarding service composition, the term Composite as a Service (Caas) [9] or Composition as a Service [3] denotes the concept of having a layer on top of Software as a Service (SaaS), which applies process-based application design principles. Defining or executing a composition can be provided as a service which can be offered by a vendor or by a third party. By specifying own compositions, vendor offerings can be combined with services developed in-house. For example, the augmentation of a process with information related to the distribution of activities to the sites of a company may be performed by an in-house service, while an advanced graphical rendering may be provided by a third party.

6 Conclusion

In this paper, we presented an approach for defining and enacting lightweight, service-based applications that form complex process view transformation functionality which can be offered as a service. We introduced an architectural framework and discussed key aspects regarding the realization of such an architecture as well as different scenarios where process view services and their compositions apply. We see our approach as an aid to find a balance between simplicity-of-use on the one hand, and providing flexibility and expressiveness on the other hand, when defining composition of process view services in particular and also when defining service compositions per se. While BPEL provides full flexibility which may be required for specific service compositions, the lightweight approach we presented in this paper is limited, but easy to apply even with little technical skills.

Acknowledgments. The authors D.S. and D.K. would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC 310/1) at the University of Stuttgart. Many thanks go to the EC-Web reviewers for their valuable feedback.

References

1. OASIS: Web Services Business Process Execution Language Version 2.0 (2007)
2. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and more focused control-flow analysis for business process models through SESE decomposition. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 43–55. Springer, Heidelberg (2007)

3. Blake, M., Tan, W., Rosenberg, F.: Composition as a Service. IEEE Internet Computing 14(1), 78–82 (2010)
4. Schumm, D., Leymann, F., Streule, A.: Process Viewing Patterns. In: Proceedings of the 14th IEEE International EDOC Conference, EDOC 2010. IEEE Computer Society Press, Los Alamitos (2010)
5. Eshuis, R., Grefen, P.: Constructing Customized Process Views. Data & Knowledge Engineering 64(2), 419–438 (2008)
6. Schumm, D., Anstett, T., Leymann, F., Schleicher, D.: Applicability of Process Viewing Patterns in Business Process Management. In: Proceedings of the International Workshop on Models and Model-driven Methods for Service Engineering (3M4SE 2010). IEEE Computer Society Press, Los Alamitos (2010)
7. Schumm, D., Leymann, F., Streule, A.: Process Views to Support Compliance Management in Business Processes. In: Buccafurri, F., Semeraro, G. (eds.) EC-Web 2010. Lecture Notes in Business Information Processing, vol. 61, pp. 131–142. Springer, Heidelberg (2010)
8. Zhao, X., Liu, C., Sadiq, W., Kowalkiewicz, M., Yongchareon, S.: Implementing Process Views in the Web Service Environment. WWW Journal 14(1), 27–52 (2011)
9. Leymann, F.: Cloud Computing: The Next Revolution in IT. In: Proceedings of the 52th Photogrammetric Week (2009)
10. Polyvyanyy, A., Smirnov, S., Weske, M.: Business Process Model Abstraction. In: International Handbook on Business Process Management. Springer, Heidelberg (2009)
11. Bobrik, R., Reichert, M., Bauer, T.: View-based process visualization. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 88–95. Springer, Heidelberg (2007)
12. Smirnov, S., Dijkman, R., Mendling, J., Weske, M.: Meronymy-Based Aggregation of Activities in Business Process Models. In: Parsons, J., Saeki, M., Shoval, P., Woo, C., Wand, Y. (eds.) ER 2010. LNCS, vol. 6412, pp. 1–14. Springer, Heidelberg (2010)
13. Reichert, M., Bassil, S., Bobrik, R., Bauer, T.: The Proviado Access Control Model for Business Process Monitoring Components. Enterprise Modelling and Information Systems Architectures - An International Journal. German Informatics Society (GI) (2010)
14. Spiller, J.: Privacy-enhanced Service Execution. In: Proceedings of the International Conference for Modern Information and Telecommunication Technologies (2008)
15. Barros, A.P., Dumas, M.: The Rise of Web Service Ecosystems. IT Professional 8(5), 31–37 (2006)
16. Object Management Group (OMG): Business Process Model and Notation (BPMN), Version 2.0, OMG Document Number formal/2011-01-03 (January 2011)
17. La Rosa, M., ter Hofstede, A., Wohed, P., Reijers, H., Mendling, J., van der Aalst, W.: Managing Process Model Complexity via Concrete Syntax Modifications. IEEE Transactions on Industrial Informatics 7(2), 255–265 (2011)
18. Moody, D.L.: The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE Transactions on Software Engineering 35, 756–779 (2009)