

# Name-based view integration for enhancing the reusability in process-driven SOAs

Huy Tran<sup>1</sup>, Uwe Zdun<sup>1</sup>, and Schahram Dustdar<sup>1</sup>

Distributed Systems Group  
Information System Institute  
Vienna University of Technology, Austria.  
htran, zdun, dustdar@infosys.tuwien.ac.at

**Abstract.** Many companies opt for reusing existing software development artifacts due to the benefits of the reuse such as increasing productivity, shortening time-to-market, and spending less time for testing, debugging, to name but a few. Unfortunately, reusing artifacts in existing process-driven SOA technologies is cumbersome and hard to achieve due to several inhibitors. First, the languages used for business process development are not intentionally designed for reuse. Second, numerous tangled process concerns embraced in a process description significantly hinder the understanding and reusing of its concepts and elements. Third, there is a lack of appropriate methods and techniques for integrating reusable artifacts. In our previous work, we proposed a view-based, model-driven approach for addressing the two former challenges. We present in this paper a named-based view integration approach aiming at solving the third one. Preliminary qualitative and quantitative evaluations of four use cases extracted from industrial processes show that this approach can enhance the flexibility and automation of reusing process development artifacts.

**Key words:** reuse, business process, SOAs, view-based, model-driven, name-based, tool support

## 1.1 Introduction

Process-driven, service-oriented architectures (SOAs) advocate the notion of process in order to aggregate various business functionality to accomplish a certain goal, such as fulfilling a purchase order, handling customer complaints, booking a travel itinerary, and so on. A typical business process consists of a number of activities that are orchestrated by a control flow. Each activity is either a communication task (e.g., invoking other services, processes, or an interaction with a human) or a data processing task. Business processes are often designed by business and domain experts using high-level, notational languages, such as Business Process Modeling Notation (BPMN)<sup>1</sup> and UML Activity Diagram<sup>2</sup>. Process designs in the aforementioned languages are mostly non-executable, and therefore, have to be translated into or implemented in low-level, executable languages such as Business Process Execution Language (BPEL)<sup>3</sup>. After

<sup>1</sup> <http://www.bpmn.org>

<sup>2</sup> <http://www.uml.org>

<sup>3</sup> <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf>

that, process implementations can be deployed in a process engine for executing and monitoring.

The IEEE Glossary of Software Engineering Terminology defines reusability as “*the degree to which a software module or other work product can be used in more than one computer program or software system*” [1]. The significant benefit of reuse is to improve software quality and productivity [2, 3]. There are several types of reusable aspects in software projects such as architectures, source code, data, design, documentation, test cases, requirements, etc. [4, 5]. The state-of-the-art software reuse practice suffers from several technical and non-technical inhibitors [6, 7]. Reuse in business process development is not an exception. We identify the most important factors that hinder the reuse of artifacts during the process development life cycle as:

- Most of the languages used for modeling and developing processes, such as BPMN, UML Activity Diagram, EPC, WS-BPEL, etc., are not intentionally designed for reuse. As a result, none of the plethora of existing tools for business process design and development offers adequate support for reusing development artifacts.
- A process description based on the aforementioned languages is often suffering from various tangled concerns such as the control flow, collaborations, data handling, transaction, and so on. As the number of services or processes involved in a business process grows, the complexity of the process increases along with the number of invocations, data exchanges, and therefore, multiplies the difficulty of analyzing, understanding, and reusing any artifacts.
- The lack of adequate method support for flexibly integrating and composing reusable artifacts also contributes to the difficulty of reusing process artifacts.

In our previous work we proposed a novel approach for addressing the complexity of business process development [8, 9, 10, 11, 12]. Our approach explored the notion of views and the model-driven stack in order to separate process representations (e.g. process designs or implementations) into different (semi-)formalized view models. This way, stakeholders can be provided with tailored perspectives by view integration mechanisms [10, 8] according to their particular expertise and interests. View models are also organized into appropriate levels of abstraction: high-level, abstract views are suitable for business experts while low-level, technology-specific views are mostly used by technical specialists. In this paper we focus on providing a solution for the third issue mentioned above, i.e., supporting methods for reusing and integrating process artifacts in a flexible manner. In particular, we introduce a name-based matching approach for view model integration and show that this approach can enhance the flexibility and automation of process artifacts (i.e., process views and view elements) reuse via industrial case studies.

This paper is organized as follows. In Section 1.2 we briefly introduce the View-based Modeling Framework [8, 10, 9] that realizes the view-based, model-driven approach. Next, Section 1.3 presents a name-based view integration approach which is simple, efficient, and flexible for improving the reusability. Processes extracted from four case studies are exemplified to illustrate our approach in Section 1.4 along with a quantitative study to evaluate this approach in industrial context. Then Section 1.5 discusses the related work. Finally, Section 1.6 summarizes our main contributions.

## 1.2 View-based Modeling Framework

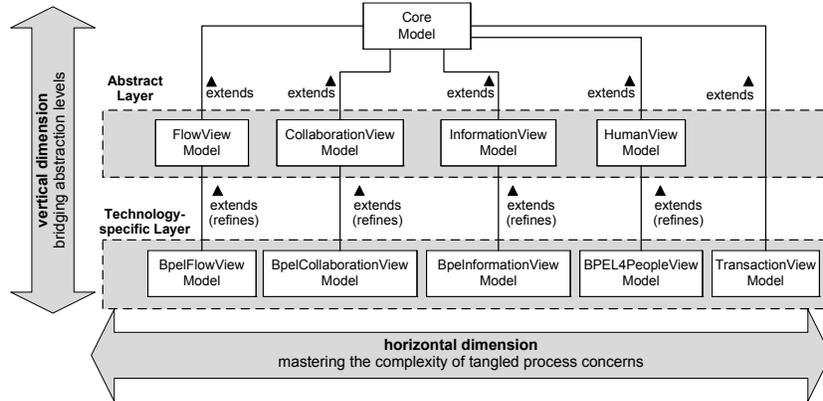


Fig. 1.1: Overview of the View-based Modeling Framework ([8, 9])

In this section, we briefly introduce the View-based Modeling Framework (VbMF), which is an implementation of our view-based, model-driven approach [8]. VbMF exploits the notion of process views to separate tangled process concerns in order to reduce the complexity and enhance the flexibility and extensibility in process-driven SOA development. Each process concern, i.e., a particular perspective of business processes, is (semi-)formally described by a view model that comprises a number of elements and their relationships. VbMF view models are organized into abstract and technology-specific layers. As such, business experts, who mostly work with the high level view models, can better formulate domain- and business-oriented concepts and knowledge because the technical details have been abstracted away. For particular process-driven technologies, such as BPEL, VbMF provides extension models that add details to the abstract models that are required to depict the specifics of these technologies [8]. These extension views belong to the technology-specific layer shown in Figure 1.1.

VbMF initially provides stakeholders with basic (semi-)formalizations, which are the **FlowView**, **CollaborationView** and **InformationView** models, for describing a business process. The **FlowView** model specifies the orchestration of process activities, the **CollaborationView** model represents the interactions with other processes or services, and the **InformationView** model elicits data representations and processing. Nonetheless, VbMF is not bound to these view models but can be extended for capturing many other concerns, for instance, human interaction [9], data access and integration [13], and traceability [12]. View models of VbMF are derived from fundamental concepts and elements of the Core model. Thus, the concepts of the Core model are the extension points and integration points of VbMF [8].

We implemented VbMF as Eclipse plugins based on the Eclipse Modeling Framework<sup>4</sup>. To illustrate how VbMF works in reality, we exemplify parts of the billing and

<sup>4</sup> <http://www.eclipse.org/emf>

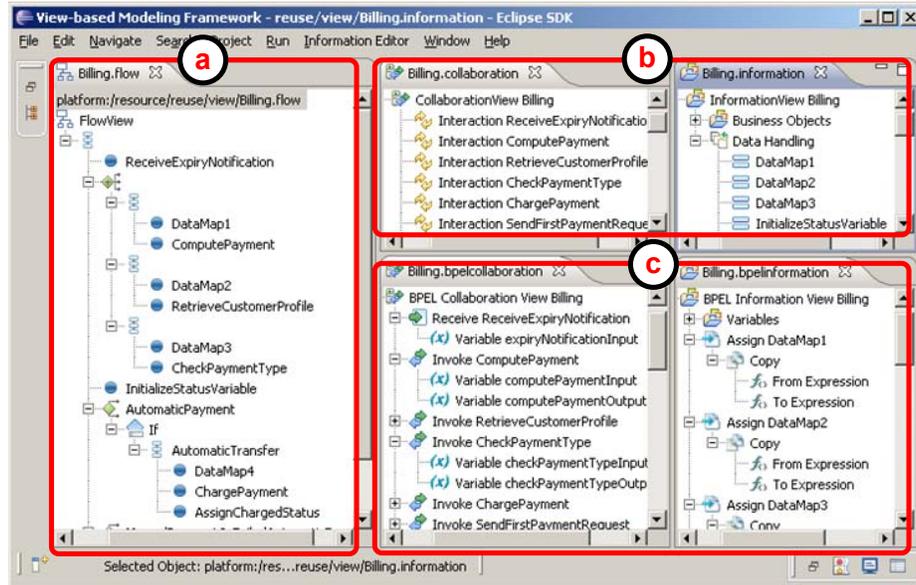


Fig. 1.2: Billing Renewal process development using VbMF

provisioning system of a domain registrar and hosting provider [14]. The billing system comprises a wide variety of services including: credit bureau services (cash clearing, card validation and payment, etc.), domain services (whois, domain registration and transfer, etc.), hosting services (Web and email hosting, provisioning, etc.), and retail services (customer service and support, etc.). The company has developed a business process, namely, Billing Renewal process, in order to integrate and orchestrate core functionality and the services. We present the VbMF views of the Billing Renewal process that are the FlowView (Figure 1.2(a)), the high-level CollaborationView and InformationView (Figure 1.2(b)), and the low-level BpelCollaborationView and BpelInformationView (Figure 1.2(c)). For further details of VbMF, we would like to refer the readers to [8, 10, 11, 12, 9].

### 1.3 Name-based View Integration Approach

In our view-based, model-driven approach, the FlowView – as the most important concern in process-driven SOA – is often used as the central view. Views can be integrated via integration points to produce a richer view or a thorough view of the business process. We propose a name-based matching algorithm for realizing the view integration mechanism (see Algorithm 1). This algorithm is simple, but effectively used at the view level (or model level) because from a modeler’s point of view in reality it makes sense, and is reasonable, to assign the same name to the modeling entities that pose the same functionality and semantics. Nonetheless, other view integration approaches such as those using class hierarchical structures or ontology-based structures are applicable in our approach with reasonable effort as well. Exploring other view integration

mechanisms and comparing them with the name-based matching approach is beyond the scope of this paper. Therefore, we merely focus on the name-based view integration and illustrate its promising advantages contributing to improve the reusability of process artifacts.

Before discussing in detail the name-based view integration, we introduce the definition of conformity of model elements and integration points. Let  $m$  be an element of a certain view model, the symbol  $\hat{m}$  denotes the hierarchical tree of inheritance of  $m$ , i.e., all elements which are ancestors of  $m$ , and  $m.x$  denotes the value of the attribute  $x$  of the element  $m$ .

**Definition 1 (Conformity).** Let  $M_1, M_2$  be two view models and  $m_1 \in M_1$  and  $m_2 \in M_2$ . Two elements  $m_1$  and  $m_2$  are **conformable** if and only if  $m_1$  and  $m_2$  have at least one common parent type in their tree of inheritance or  $m_1$  is of type  $m_2$ , or vice versa.

Using  $m_1 \uparrow m_2$  to denote  $m_1$  and  $m_2$  are **conformable**, Definition 1 is given as:

$$m_1 \uparrow m_2 \iff (\hat{m}_1 \cap \hat{m}_2 \neq \emptyset) \vee (m_1 \in \hat{m}_2) \vee (m_2 \in \hat{m}_1)$$

**Definition 2 (Integration point).** Let  $M_1, M_2$  be two view models and two views  $V_1, V_2$  be instances of  $M_1$  and  $M_2$ , respectively. A couple of elements  $e_1$  and  $e_2$ , where  $e_1 \in V_1$  and  $e_2 \in V_2$ ,  $e_1$  is an instance of  $m_1$ , and  $e_2$  is an instance of  $m_2$ , is an **integration point** between  $V_1$  and  $V_2$  if and only if  $m_1$  and  $m_2$  are **conformable** and  $e_1$  and  $e_2$  have the same value of the attribute “**name**”.

Using  $I(e_1, e_2)$  to denote the integration point between two views  $V_1$  and  $V_2$  at the elements  $e_1$  and  $e_2$ , and  $x \succ y$  to denote  $x$  is an instance of  $y$ , Definition 2 can be written as:

$$I(e_1, e_2) \iff (m_1 \uparrow m_2) \wedge (e_1.name = e_2.name)$$

where

$$e_1 \in V_1, e_2 \in V_2, e_1 \succ m_1, e_2 \succ m_2, V_1 \succ M_1, V_2 \succ M_2$$

The main idea of the name-based matching for view integration is to find all integration points  $I(e_1, e_2)$  between two views  $V_1$  and  $V_2$  and merge these two views at those integration points. The merging at a certain integration point  $I(e_1, e_2)$  is done by creating a new element which aggregates the attributes and references of both  $e_1$  and  $e_2$  (see Algorithm 1). The complexity of the name-based matching algorithm is approximately  $O(k + l + k \times l)$ , where  $k = |V_1|$  and  $l = |V_2|$ . This complexity can be significantly reduced by generating and maintaining a configuration file containing the integration points of every pair of views with tool support. The integration points can be automatically derived from the relationships between two views. Later on, the view integration algorithm only loads the configuration file and performs view merging straightforwardly. This way, the complexity of the view integration algorithm can be reduced to approximately  $O(P)$ , where  $P$  is the number of integration points between  $V_1$  and  $V_2$ . We note that  $P \leq k \times l$ . In reality, the numbers of elements which are used for view integration are often much less the total number of elements of the containing view, and therefore,  $P \ll k \times l$ . Nonetheless, this approach requires additional support, especially tool support, for deriving and maintaining the integration points, which is one of our ongoing endeavors to complete the framework.

**Algorithm 1:** View integration by name-based matching

---

```

Input: View  $V_1$  and view  $V_2$ 
Output: Integrated view  $V_{12}$ 
begin
   $V_{12}.initialize();$ 
   $E_1 \leftarrow V_1.getAllElements();$ 
   $E_2 \leftarrow V_2.getAllElements();$ 
   $V_{12}.addElements(E_1);$ 
   $V_{12}.addElements(E_2);$ 
  foreach  $e_1 \in E_1$  do
     $found \leftarrow \text{false};$ 
    while not found do
       $e_2 \leftarrow E_2.next();$ 
      if  $(e_1.name = e_2.name) \wedge (e_1.superType \uparrow e_2.superType)$  then
         $found \leftarrow \text{true};$ 
         $e_{new} \leftarrow \text{createNewElement}();$ 
         $e_{new}.attribute \leftarrow \text{merge}(e_1.attribute, e_2.attribute);$ 
         $e_{new}.reference \leftarrow \text{merge}(e_1.reference, e_2.reference);$ 
         $V_{12}.addElements(e_{new});$ 
         $V_{12}.removeElements(e_1, e_2);$ 
      end if
    end while
  end foreach
  return  $V_{12};$ 
end

```

---

## 1.4 Case Study

In this section, a typical process development scenario is used to demonstrate how the name-based view integration in VbMF can support a flexible reuse of process artifacts. After that, we present a preliminary quantitative evaluation of our approach based on four use cases extracted from industrial business processes.

### 1.4.1 Process artifacts reuse scenario

As shown in Section 1.2, the Billing Renewal process has been developed using VbMF. Now the company starts develop an Order Handling process such that Internet customers can order the company's products via the Web site. Figure 1.3 shows the core functionality of the Order Handling process in terms of a BPMN diagram. The company opts to reuse existing artifacts as much as possible to develop the Order Handling process rather than starting from scratch. After analyzing the business requirements, the developers identify a number of fragments of process models and services with similar functionality existing across the enterprise. For instance, the Order Handling process requires a task that charges customer payment by invoking the services provided by the credit bureau partner. This task is similar to the *ChargePayment* task of the Billing Renewal process developed before. Therefore, this task should be reused in the Order Handling process rather than being re-developed.

Figure 1.4 illustrates how the developers reuse the existing *ChargePayment* activity for modeling the Order Handling process. The scenario is presented in terms of UML

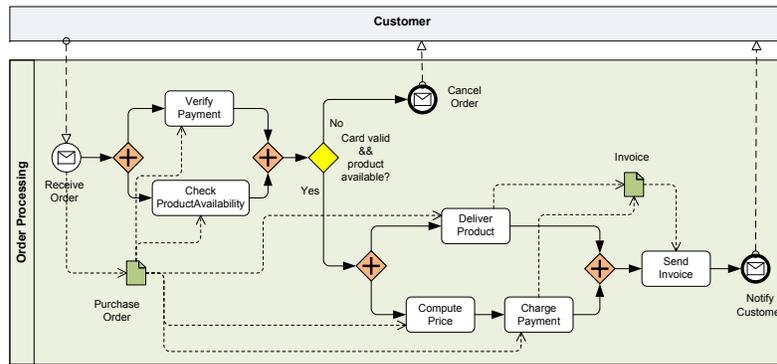


Fig. 1.3: Overview of the Order Handling process

object diagrams. On the right-hand side, we show the CollaborationView and BpelCollaborationView of the Billing Renewal process where the *ChargePayment* activity is defined at high-level and low-level of abstract, respectively. In the Billing Renewal CollaborationView, *ChargePayment:Interaction* – an instance of the *Interaction* class – has relationships with three other objects: *CreditBureau:Partner*, *CreditBureau:Interface*, and *charge:Operation*. The *ChargePayment:Interaction* object is refined in the Billing Renewal BpelCollaborationView by the *ChargePayment:Invoke* object – an instance of the *Invoke* class. The *ChargePayment:Invoke* object has two more associations with the *chargePaymentInput:VariableReference* and *chargePaymentOutput:VariableReference* objects.

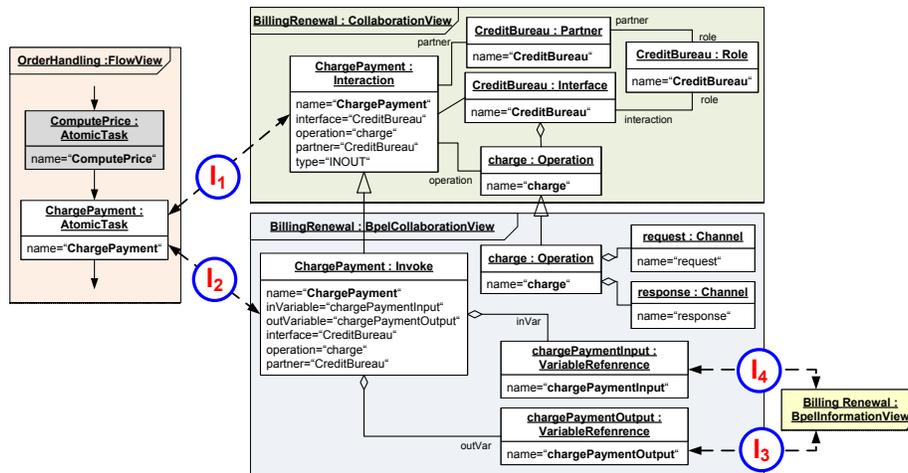


Fig. 1.4: Name-based view integration approach for reusing by referencing the *Charge Payment* element of the Billing Renewal Process in the Order Handling process

In order to properly reuse the *ChargePayment* activity of the Billing Renewal process, the developers perform two steps:

1. Create a corresponding *ChargePayment:AtomicTask* in the Order Handling FlowView as shown in the right-hand side of Figure 1.4.
2. Perform one of the following tasks (note that these tasks can be supported by the framework in a (semi-)automatic manner):
  - a) Explicitly define either an integration point  $I_1$  between the *ChargePayment:AtomicTask* and the *ChargePayment:Interaction* or  $I_2$  between the *ChargePayment:AtomicTask* and the *ChargePayment:Invoke*.
  - b) Explicitly specify the CollaborationView and BpelCollaborationView of the Billing Renewal process are input views of the Order Handling process. As VbMF supports view integration by name-based matching (cf. Section 1.3), the aforementioned integration points can be implicitly resolved by VbMF tooling, for instance, the code generators.

A question might be risen at this point: “How’s about the relationships between the reused elements and other views or elements?”. For instance, the *ChargePayment:Invoke* has associations with *chargePaymentInput:VariableReference* and *chargePaymentOutput:VariableReference* objects which are instances of the *VariableReference* class. In the Billing Renewal process, the actual definitions of these objects belong to the BpelInformationView. Therefore, these objects are part of the integration points  $I_3$  and  $I_4$ , respectively, between the BpelCollaborationView and BpelInformationView of the Billing Renewal process. In this situation, the stakeholders can take any one of two possible approaches which can be (semi-)automatically supported by our modeling framework:

1. Reuse the existing integration points between the BpelCollaborationView and BpelInformationView of the Billing Renewal process: The stakeholders can gain more benefit of reusability but they have to analyze the subsequent dependencies of the reused objects in the BpelInformationView. In addition, these subsequent dependencies also require extra effort to maintain view synchronization when making any change in the reused views. This task is supported by our traceability approach [12].
2. Create new objects in the Order Handling BpelInformationView bearing corresponding names, then  $I_3$  and  $I_4$  can be automatically derived. Although no benefit of reusability gained, there is also no binding to the Billing Renewal BpelInformationView. That is, no extra effort for understanding the subsequent dependencies or maintaining view synchronization is required.

#### 1.4.2 Quantitative evaluation

So far we presented a development scenario to illustrate how our view-based, model-driven powered by the name-based matching can improve the flexibility and automation of reuse process development artifacts. To explore the application and pragmatic usage of our approach, adequate experiments to quantitatively evaluating it in industrial business process development environment are definitely necessary. As the use cases examined in our work are mostly in the preliminary development phase. Thus,

the reuse rate is an adequate factor for the initial assessment of the value of the reuse method [15, 5]. We present in this section our quantitative evaluations of the reuse rate according to the model proposed by Gaffney and Cruickshank (which is called *the proportion of reuse*) [15] as well as by Frakes and Terry (which is called *reuse percent*) [5]. Essentially, the reuse rate  $R_R$  of each view reflects how much of that view can be attributed to reuse and be computed by the formula  $R_R = \frac{E_R}{E} \times 100$ , where  $E_R$  is the number of reusable/reused elements and  $E$  is the total number of model elements of the corresponding view [15, 5].

We have conducted the quantitative evaluation in four processes extracted from industrial use cases. Two of them are the Billing Renewal and the Order Handling processes mentioned in the previous sections. Two other processes are the CRM Fulfillment process [14] and the Travel Booking process [16]. The CRM Fulfillment process is part of the customer relationship management (CRM), billing, and provisioning systems of an Austrian Internet Service Provider. The Travel Booking process is based upon the procedure of making itinerary arrangements. It comprises typical steps for accomplishing a travel reservation: Internet customers submit data about the travel itineraries and receive a confirmation number when the travel itineraries have been booked successfully. These processes are mostly in the modeling and implementation phases. In Table 1.1, we present the reuse rate  $R_R$  of VbMF views, such as CollaborationView (CV), InformationView (IV), BpelCollaborationView (BCV), and BpelInformationView (BIV), of each case study.

Process	CV			IV			BCV			BIV		
	$E_R$	$E$	$R_R$ (%)									
<i>Billing Renewal</i>	49	63	77.78	59	85	69.41	63	132	47.73	407	494	82.39
<i>CRM Fulfillment</i>	60	74	81.08	63	78	80.77	74	131	56.49	448	537	83.43
<i>Order Handling</i>	29	36	80.56	36	44	81.82	36	65	55.38	238	286	83.22
<i>Travel Booking</i>	27	33	81.82	33	43	76.74	33	56	58.93	219	260	84.23
<b>Average</b>	<b>80.31</b>			<b>77.19</b>			<b>54.63</b>			<b>83.32</b>		

Table 1.1: The reuse rate of process view models in four use cases

As illustrated in the previous development and reuse scenario, each element of VbMF process views is potentially reusable artifact. A FlowView purely contains a control flow that defines the business logic, i.e., the execution order of process activities in order to achieve a particular business goal. Note that detailed specification of process activities, for instance, invoking a service, transforming data objects, are not embraced in the FlowView but others such as (Bpel)CollaborationView and (Bpel)InformationView. Therefore, reusing an existing FlowView to develop a new process is still possible but inefficient. Nonetheless, a FlowView can be reused as the documentation of an “*as-is*” process that can be referenced, or even used as a skeleton, for developing new processes. For this reason, we omit the reuse rate of the FlowView in Table 1.1.

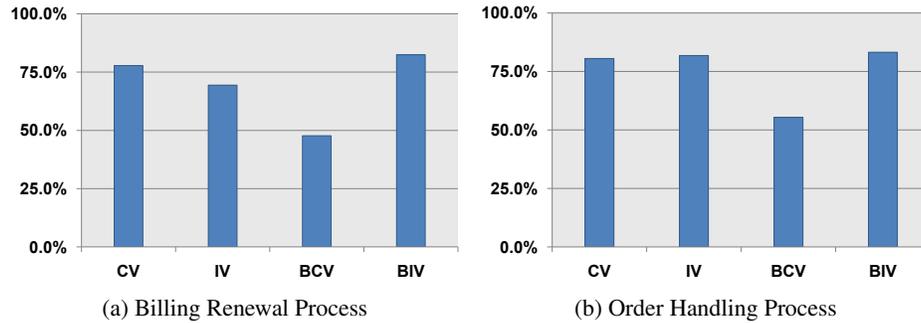


Fig. 1.5: The reuse rate of view models in the Billing Renewal and Order Handling processes

The ratio of reuse also reflects the tendency of integration of VbMF views. That is, *AtomicTasks* of the FlowView are often integrated with the corresponding elements of the CollaborationView and InformationView such as *Interaction* and *Data Handling*, or elements of the BpelCollaborationView and BpelInformationView, such as *Receive*, *Reply*, *Invoke*, and *Assign*. In addition, a number of elements of the (Bpel)CollaborationView have references to corresponding elements of (Bpel)InformationView whilst none of the (Bpel)InformationView's element depends on other views' elements. As a result, the ratio of reuse of the (Bpel)InformationView is much higher than that of the (Bpel)CollaborationView. The ratios of reuse of high-level views are higher than that of low-level ones because the abstract concepts are more reusable than the technology-specific counterparts. The average degrees of reuse over four use cases are very promising: **80.31%** for the CollaborationView (CV), **77.19%** for the InformationView (IV), **54.63%** for the BpelCollaborationView (BCV), and **83.32%** for the BpelInformationView (BIV). Because the reuse rates of view models of each use case is almost identical to those of the others, we only show the visualizations of the evaluation results of the Billing Renewal and Order Handling processes (see Figure 1.5).

## 1.5 Related Work

Software reuse has been an active field of study in software engineering since last three decades that leads many promising results for reusing existing software or software knowledge to build new software [7, 4, 6]. Several work in this field has contributed success stories in various aspects such as reuse libraries, domain engineering methods and tools, reuse design, design patterns, domain specific software architecture, components, generators, and so on [6]. Yet there has been very few investigation of reuse in the area of business process management, in particular, business process development.

As we mentioned above, most of popular languages used for modeling and developing business processes such as BPMN, UML Activity Diagram, EPC, BPEL, etc., are

not intentionally designed for reuse. As a consequence, developers find it hard to reuse a certain excerpt of a process represented in any of these languages. Reuse merely exists in form of “*copy-and-paste*” if the same language is used to model and develop business processes. Otherwise, necessary interpretation and translation must be performed in order to reuse existing processes. All these are however cumbersome and error-prone tasks.

To the best of our knowledge, most of researches on software reuse in the domain of business process management focus on the control flow of the business process. Van der Aalst et al. [17] proposed several so-called workflow patterns, which are reusable control flow structures representing frequently occurring knowledge for constructing workflows. Each pattern has a sound semantic and example usage in various workflow products. These patterns can be applied for specifying, analyzing, understanding the control flow of business processes. Similarly, Schumm et al. [18] present an approach based on the notion of process fragment that enables a flexible method for describing and integrating existing artifacts into business processes. From our point of view, the aforementioned approaches and our work in this paper are nicely complementary. We believe that further exploring and integrating can fully benefit the reuse of the control flow. The distinctive point is that our approach does not solely focus on the reuse of the control flow per se. Facilitating VbMF’s extension mechanisms [8], we aim at supporting the flexible reuse of business processes from different aspects such as collaborations, data handling, etc., considering the control flow as the central notion.

Markovic and Pereira present a preliminary approach based on  $\pi$ -calculus and ontologies to provide richer representations of business process aspects such as function, information, organization, etc., [19]. This approach aims at using ontologies to explicitly specify business knowledge for better manipulating and reusing. However, the authors have not further mentioned or investigated the reuse of these knowledge in the business process life cycle.

## 1.6 Conclusion

In the domain of process-driven SOAs, reusing existing development artifacts is hindered by various factors. First, the languages used for modeling and developing processes are not intentionally designed for reuse. Second, business process representations in these languages are often complex and tangled by various concerns such that it is hard for the stakeholders to analyze, understand, and reuse them. Last but not least, there is still a lack of methods for flexibly integrating reusable artifacts.

In our previous work, we presented a novel solution for addressing the two former challenges. In this paper we focused on a name-based view integration approach aiming at solving the last challenge. Through a qualitative scenario-driven and a quantitative evaluation, we show that promising results on reusing process development artifacts can be achieved using our approach. Nonetheless, further endeavors such as industrial experiments and surveys over several software projects are definitely necessary in order to confirm the application and pragmatic usage of this approach in reality. In addition, exploring other view integration methods, such as those based on concept hierarchies or ontologies, can help fully exploiting the benefit of reuse and enhancing the automation in reusing process development artifacts.

## Acknowledgement

This work was supported by the European Union FP7 project COMPAS, grant no. 215175.

## References

1. IEEE: Standard Glossary of Software Eng. Terminology (December 1990)
2. Gaffney, J., Durek, T.A.: Software reuse: key to enhanced productivity: some quantitative models. *Information and Software Technology* **31**(5) (June 1989) 258–267
3. Fichman, R., Kemerer, C.F.: Incentive compatibility and systematic software reuse. *J. Systems and Software* **57**(1) (April 2001) 45–60
4. Krueger, C.W.: Software reuse. *ACM Comp. Surv.* **24**(2) (1992) 131–183
5. Frakes, W., Terry, C.: Software reuse: metrics and models. *ACM Comp. Surv.* **28**(2) (June 1996) 415–435
6. Frakes, W., Kang, K.: Software reuse research: status and future. *IEEE Trans. Software Eng.* **31**(7) (July 2005) 529–536
7. Morisio, M., Ezran, M., Tully, C.: Success and failure factors in software reuse. *IEEE Trans. Software Eng.* **28**(4) (April 2002) 340–357
8. Tran, H., Zdun, U., Dustdar, S.: View-based and Model-driven Approach for Reducing the Development Complexity in Process-Driven SOA. In: *Int'l Conf. Business Process and Services Computing (BPSC)*. Volume 116 of LNI., GI (2007) 105–124
9. Holmes, T., Tran, H., Zdun, U., Dustdar, S.: Modeling Human Aspects of Business Processes - A View-Based, Model-Driven Approach. In: *4th European Conf. Model Driven Architecture Foundations and Applications (ECMDA-FA)*, Springer (2008) 246–261
10. Tran, H., Zdun, U., Dustdar, S.: View-based Integration of Process-driven SOA Models At Various Abstraction Levels. In: *1st Int'l Workshop on Model-Based Software and Data Integration*, Springer (April 2008) 55–66
11. Tran, H., Zdun, U., Dustdar, S.: View-Based Reverse Engineering Approach for Enhancing Model Interoperability and Reusability in Process-Driven SOAs. In: *10th Int'l Conf. Software Reuse (ICSR)*, Springer (2008) 233–244
12. Tran, H., Zdun, U., Dustdar, S.: VbTrace: Using View-based and Model-driven Development to Support Traceability in Process-driven SOAs. *J. Softw. Syst. Model.* (2009) <http://dx.doi.org/10.1007/s10270-009-0137-0>.
13. Mayr, C., Zdun, U., Dustdar, S.: Model-Driven Integration and Management of Data Access Objects in Process-Driven SOAs. In: *ServiceWave*. (2008) 62–73
14. Evenson, M., Schreder, B.: SemBiz Project: D4.1 Use Case Definition and Functional Requirements Analysis. <http://sembiz.org/attach/D4.1.pdf> (August 2007)
15. Gaffney, J.E., Cruickshank, R.D.: A general economics model of software reuse. In: *14th Int'l Conf. Software Eng. (ICSE)*, ACM Press (1992) 327–337
16. IBM: Business process use cases. <http://publib.boulder.ibm.com/bpcsamp> (2006) (accessed 2008/01/05).
17. van der Aalst, W., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. *Distributed and Parallel Databases* **14**(1) (2003) 5–51
18. Schumm, D., Leymann, F., Ma, Z., Scheibler, T., Strauch, S.: Integrating Compliance into Business Processes Process Fragments as Reusable Compliance Controls. In: *Multikonferenz Wirtschaftsinformatik (MKWI)*, Universitätsverlag Göttingen (2010) 2125–2137
19. Markovic, I., Pereira, A.C.: Towards a Formal Framework for Reuse in Business Process Modeling. In: *BPM Workshops Advances in Semantics for Web services 2007 (semantics4ws'07)*, Springer (2008) 484–495