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

Huy Tran* and Uwe Zdun

Software Architecture Group,
Faculty of Computer Science,
University of Vienna,
Berggasse 11, 1090 Wien, Austria
Fax: +43-1-4277-39699
E-mail: huy.tran@univie.ac.at
E-mail: uwe.zdun@unvie.ac.at
*Corresponding author

Schahram Dustdar

Distributed Systems Group,
Institute of Information Systems,
Vienna University of Technology,
Argentinierstrasse 8/184-1, Wien 1040, Austria
Fax: +43-1-8801-18491
E-mail: dustdar@infosys.tuwien.ac.at

Abstract: Many companies opt for reusing existing software development artefacts 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 artefacts 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 artefacts. 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 artefacts.

Keywords: reuse; business process; SOAs; view-based; model-driven; name-based; tool support.

Reference to this paper should be made as follows: Tran, H., Zdun, U. and Dustdar, S. (2011) 'Name-based view integration for enhancing the reusability in process-driven SOAs', *Int. J. Business Process Integration and Management*, Vol. 5, No. 3, pp.229–239.

Biographical notes: Huy Tran is a Postdoctoral Researcher working at the Software Architecture Group, Faculty of Computer Science, University of Vienna, Austria. He received his Bachelor in Computer Science and Engineering from Ho Chi Minh City University of Technology, Vietnam in 2002 and PhD in Computer Science specialised in Software Engineering at Vienna University of Technology, Austria in 2009. He has participated in a number of European projects. His current research interests include: software architecture, model-driven engineering, domain-specific modelling, business processes, compliance engineering and service-oriented computing.

Uwe Zdun is a Full Professor for Software Architecture at the Faculty of Computer Science, University of Vienna. He received his PhD from the University of Essen in 2002. His research focuses on architectural decision, software patterns, modelling of complex software systems, service-oriented systems, domain-specific languages, and model-driven development. He has published more than 100 articles in journals, workshops, and conferences, and is a co-author of the books *Remoting Patterns – Foundations of Enterprise, Internet, and Real-Time Distributed Object Middleware* (J. Wiley and Sons) and *Software-Architektur: Grundlagen, Konzepte, Praxis* (Elsevier/Spektrum). He has participated in numerous R&D projects and industrial projects. He is the European Editor of the *Journal Transactions on Pattern Languages of Programming (TPLoP)* published by Springer, and Associate Editor-in-Chief for design and architecture for the *IEEE Software Magazine*.

Schahram Dustdar is a Full Professor of Computer Science (Informatics) with a focus on internet technologies heading the Distributed Systems Group, Institute of Information Systems, Vienna University of Technology (TU Wien) where he is the Director of the Vita Lab. From 2004–2010, he is the Honorary Professor of Information Systems at the Department of Computing Science at the University of Groningen (RuG), The Netherlands. From 1999–2007, he worked as the co-Founder and Chief Scientist of Caramba Labs Software AG (CarambaLabs.com) in Vienna (acquired by Engineering NetWorld AG), a venture capital co-funded software company focused on software for collaborative processes in teams. Caramba Labs was nominated for several (international and national) awards: World Technology Award in the category of software (2001); top-startup companies in Austria (Cap Gemini Ernst and Young) (2002), and MERCUR Innovationspreis der Wirtschaftskammer (2002). Since 2009, he is an ACM Distinguished Scientist.

This paper is a revised and expanded version of a paper entitled 'Name-based view integration for enhancing the reusability in process-driven SOAs' presented at the 1st International Workshop on Reuse in Business Process Management (rBPM'10) in conjunction with BPM 2010 (The 8th International Conference on Business Process Management), Hoboken, New Jersey, USA, 13–16 September 2010.

1 Introduction

Process-driven, service-oriented architectures (SOAs) advocate the notion of process in order to aggregates various business functionality to accomplish a certain business 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 modelling notation (BPMN) and UML activity diagram. 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). After that, process implementations can be deployed in a process engine for executing and monitoring.

The IEEE (1990) 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 programme or software system". The significant benefit of reuse is to improve software quality and productivity (Gaffney and Durek, 1989; Fichman and Kemerer, 2001). There are several types of reusable aspects in software projects such as architectures, source code, data, design, documentation, test cases, requirements, etc. (Krueger, 1992; Frakes and Terry, 1996). The state-of-the-art software reuse practice suffers from several technical and non-technical inhibitors (Frakes and Kang, 2005; Morisio et al., 2002). Reuse in business process development is not an exception. We identify the most important factors that hinder the reuse of artefacts during the process development life cycle as:

- Most of the languages widely used for modelling 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 artefacts.
- 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 analysing, understanding, and reusing any artefacts.
- The lack of adequate method support for flexibly integrating and composing reusable artefacts also contributes to the difficulty of reusing process artefacts.

In our previous work, we proposed a novel approach for addressing the complexity of business process development (Tran et al., 2007, 2008a, 2008b, 2009b; Holmes et al., 2008). Our approach explored the notion of views and the model-driven paradigm in order to separate process representations (e.g., process designs or implementations) into different (semi-)formalised view models. As such, stakeholders can be provided with tailored perspectives by view integration mechanisms (Tran et al., 2008a, 2007) with respect to their particular expertise and interests. View models are also organised into appropriate levels of abstraction: high-level, abstract views are suitable for business experts whilst 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 artefacts 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 artefacts (i.e., process views and view elements) reuse via industrial case studies.

This article is organised as follows. In Section 2, we introduce the view-based modelling framework (VbMF) (Tran et al., 2007, 2008a; Holmes et al., 2008) that realises the view-based, model-driven approach. Next, Section 3 presents a name-based view integration approach which is simple but efficient and flexible for enhancing the reusability of business processes. Processes extracted from four case studies are exemplified to illustrate our approach in Section 4 along with a preliminary quantitative study to evaluate this approach in industrial context. Then, Section 5 discusses the related work. Finally, Section 6 summarises our main contributions.

2 View-based modelling framework

In this section, we introduce the VbMF, which is an implementation of our view-based, model-driven approach (Tran et al., 2007). A typical business process embodies various tangled concerns, such as the control flow, data processing, service and process invocations, fault handling, event handling, human interactions, transactions, to name but a few. The entanglement of those concerns increases the complexity of process development and maintenance as the number of involved services and processes grow. In order to deal with this complexity, we use the notion of *architectural views* (or views for short) to describe the various process concerns. In particular, a view is a representation of one particular concern of a process. We devise different view models for formalising the concept of architectural view.

VbMF initially provides stakeholders with basic (semi-)formalisations, which are the FlowView, CollaborationView (CV) and InformationView (IV) models, for describing a business process. The FlowView model specifies the orchestration of process activities, the CV model represents the interactions with other processes or services, and the IV model elicits data representations and processing. Nonetheless, VbMF is not bound to these view models but can be extended for capturing many other concerns, i.e., human interaction (Holmes et al., 2008), data access and integration (Mayr et al., 2008), and traceability (Tran et al., 2009b). View models of VbMF are derived from fundamental concepts and elements of the core model (see Figure 2). As a result, the core model plays an important role in our approach because it provides the basis for extending and integrating view models, and establishing and maintaining the dependencies between view models (Tran et al., 2007). In other words, thus, the concepts of the core model are the extension points and integration points of VbMF (Tran et al., 2007).

There are many stakeholders involved in process development at different levels of abstraction. For instance, business experts require high-level abstractions that offer domain or business concepts concerning their distinct knowledge, skills, and needs, while IT experts merely work with low-level, technology-specific descriptions. The MDD paradigm provides a potential solution to this problem by separating the platform-independent and platform-specific models. A platform-independent model is a model of a software system that does not depend on the specific technologies or platforms used to implement it while a platform-specific model links to particular technologies or platforms (OMG, 2003; Frankel, 2002). Leveraging this advantage of the MDD paradigm, we devise a model-driven stack that has two basic layers: abstract and technology-specific. The abstract layer includes the views without the technical details such that the business experts can understand and manipulate. Then, the IT experts can refine or map these abstract concepts into platform- and technology-specific views. The technology-specific layer contains the views that embody concrete information of technologies or platforms. On the one hand, a technology-specific view model can be directly derived from the core model, such as the *TransactionView* model shown in Figure 1. On the other hand, a technology-specific view model can also be an extension of an abstract one, i.e., the *BpelCollaborationView* (BCV) model extends the CV model, the BPEL4PeopleView model extends the *HumanView* model (Holmes et al., 2008), etc., by using the model refinement mechanism (see Figure 1). By refining an abstract layer down to a technology-specific layer, our view-based approach helps bridging the abstraction levels along the vertical dimension, i.e., the dimension of abstraction, which is orthogonal to the *horizontal dimension* described in the previous paragraph (see Figure 1).

Based on the aforementioned view model specifications, stakeholders can create different types of views for describing specific business processes. These process views can be instances of the concerns' view models, extension view models, or integrated view models (see Figure 1). They can be manipulated by the stakeholders to achieve a certain business goal, or adapt to new requirements in business environment or changes in technology and platform. Finally, we provide *model-to-code transformations* (or so-called *code generations*) that take these views as inputs and generate process implementations and deployment configurations. The resulting code and configurations, which may be augmented with hand-written code, can be deployed in process engines for execution.

We implemented VbMF as eclipse plug-ins based on the eclipse modelling framework (EMF). To illustrate how VbMF works in reality, we exemplify parts of the billing and provisioning system of a domain registrar and hosting provider (Evenson and Schreder, 2007). The billing system comprises a wide variety of services including: credit bureau services (cash clearing, card validation and payment, etc.), domain services (who is, domain registration and transfer, etc.), hosting services (web and e-mail 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.

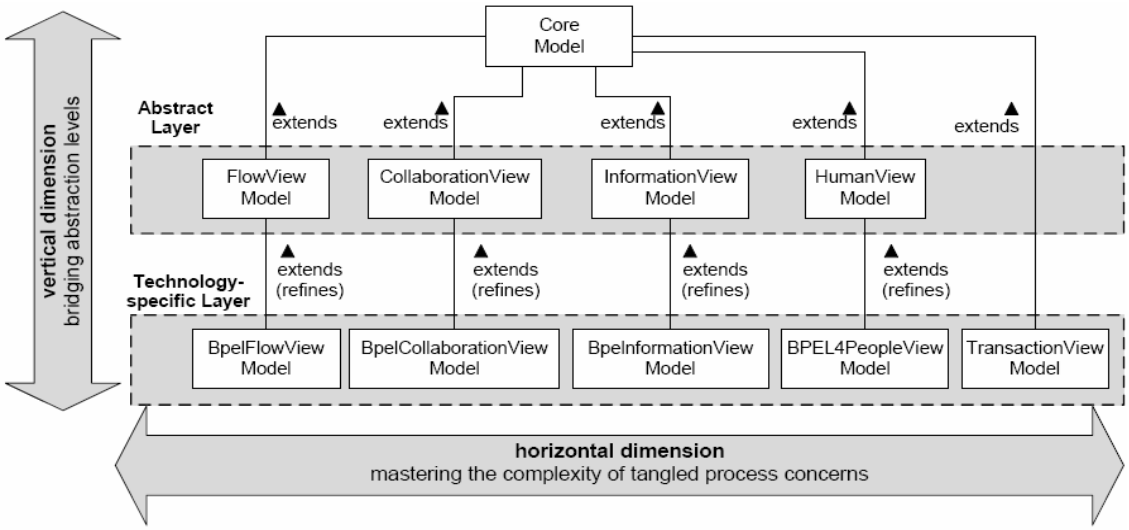
Figure 3 shows some VbMF views of the billing renewal process. The VbMF FlowView model is devised to capture essential control structures such as sequence and parallel execution, exclusive decision, loop, and so on (van der Aalst et al., 2003). The billing renewal FlowView

[see Figure 3(a)], which is an instance of the FlowView model, represents the execution order of the process's activities, i.e., the process's control flow, in form of the tree-based editors automatically generated by EMF. The blue circles are *atomic tasks* standing for the process's activities orchestrated by the aforementioned control structures to charge a customer's billing and renew his/her contract. Note that each *atomic task* will not contain any details, i.e., data handling, service invocations, transactions, etc. These details will be expressed in the corresponding views such as the CV or IV. As such, the process's FlowView is equivalent to a high-level BPMN, EPC, or UML Activity diagram abstracted away from the details of all process activities. To this extent, business and domain experts can better formulate business- and domain-oriented concepts using the FlowView.

Besides the FlowView, the other high-level views also provide high-level concepts of the corresponding process concerns complementing the process tasks defined in the FlowView. For instance, CV and IV [Figure 3(b)] provide abstract services and process interactions, business objects, respectively. Along with the FlowView, these high-level views target the business and domain experts who are rather familiar with business and domain concepts than the technical details (Tran et al., 2007).

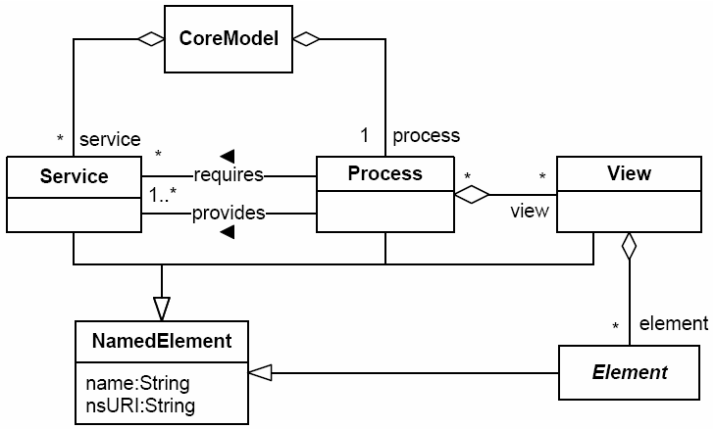
The abstract concepts provided by using the aforementioned high-level VbMF views can be elaborated with more technology-specific details using the low-level views (Tran et al., 2007). That is, the IT experts will refine or implement the abstract concepts of the high-level views by adding more technology-specific information, i.e., web service invocations, XML schema data types, etc. For instance, Figure 3(c) depicts the low-level CV and IV of the billing renewal process for the WS-BPEL and web service technologies. Note that the relationship between FlowView and other views as well as between the high-level and low-level views can be (semi-)-automatically established and maintained via the name-based view integration approach presented in the next section. For further details of VbMF, we would like to refer the readers to (Tran et al., 2007, 2008a, 2008b, 2009b; Holmes et al., 2008).

Figure 1 Overview of the VbMF



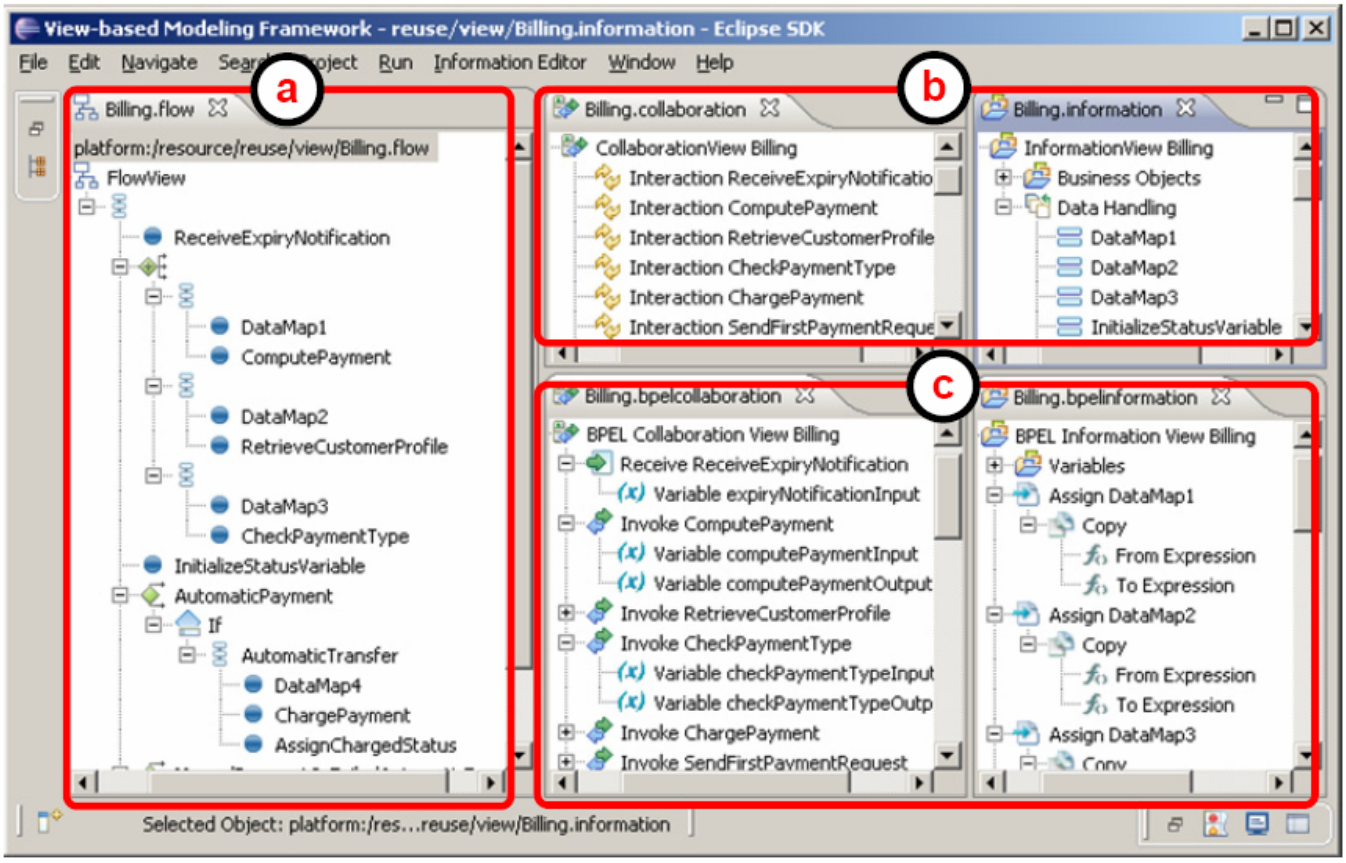
Source: Tran et al. (2007) and Holmes et al. (2008)

Figure 2 Core model – the foundation for VbMF's extension and integration



Source: Tran et al. (2007)

Figure 3 Billing renewal process development using VbMF (see online version for colours)



3 Name-based view integration approach

In the view-based, model-driven approach, the FlowView – as the most important concern in process-driven SOA – is often used as the central view. There are many stakeholders involved in process development at different levels of abstraction. For instance, business experts require high-level abstractions that offer domain or business concepts concerning their distinct knowledge, skills, and needs, while IT experts merely work with low-level, technology-specific descriptions. According to the specific needs and knowledge of the stakeholders, views can be combined to provide a richer view or a more thorough view of a certain process. For instance, IT experts may need to involve the process control flow along with service interactions which is only provided via an integration of the *FlowView* with either the CV or BCV.

We propose a name-based matching algorithm for realising the view integration mechanism (see Algorithm 1). This algorithm is simple but can be effectively used at the view level (or model level) because from a modeller's point of view in reality it makes sense, and is reasonable, to assign the same name to the modelling entities that pose the same functionality and semantics. Moreover, this can be achieved in a (semi-)automatic fashion as we illustrate in the following scenario to define the process activity 'ReceiveExpiryNotification' of the billing renewal process. First of all, the business analysts sketch out the design of the billing renewal process with various process *atomic tasks* orchestrated by control flow structures [see Figure 3(a)]. After that, the business analysts, maybe together with the domain experts or technical experts, can choose to define this task as an interaction. As such, either a new abstract interaction is created in the CV or a low-level, technology-specific interaction is created in the BCV. As such, the newly created element and the corresponding *atomic task* have the same name. We can automatically achieve the various VbMF views of the Billing Renewal process as shown in Figure 3 in the same manner. Human intervention might be necessary in some special situations, i.e., name conflicts when existing views are refactored or merging. Nonetheless, other existing view or model merging approaches such as those using database schema matching, class hierarchical structures, or ontology-based structures can better enhance the accuracy of our name-based matching approach with reasonable effort. In this article, we mainly focus on the name-based view integration and illustrate its promising advantages contributing to improve the reusability of process artefacts.

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.

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 3.1 is given as:

$$m_1 \uparrow m_2 \Leftrightarrow \left(\hat{m}_1 \cap \hat{m}_2 \neq \varnothing\right) \vee \left(m_1 \in \hat{m}_2\right) \vee \left(m_2 \in \hat{m}_1\right)$$

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 3.2 can be written as:

$$I\left(e_1, e_2\right) \Leftrightarrow \left(m_1 \uparrow m_2\right) \wedge \left(e_1.name = e_2.name\right)$$

where

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

Algorithm 1 View integration by name-based matching

```
Input: View V1 and view V2
Output: Integrated view V12
begin
    V12.initialise();
    E1 ← V1.getAllElements();
    E2 ← V2.getAllElements();
    V12:addElements(E1);
    V12:addElements(E2);
    foreach e1 ∈ E1 do
        found ← false;
        while not found do
            e2 ← E2:next();
            if (e1:name =
            e2:name) ∧ (e1:superType ← e2:superType) then
                found ← true;
                enew ← createNewElement();
                enew.attribute ← merge(e1.attribute,
                e2.attribute);
                enew.reference ← merge(e1.reference,
                e2.reference);
                V12:addElements(enew);
                V12:removeElements(e1, e2);
    return V12;
end
```

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 $I(e_1, e_2)$ can be achieved 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 using a configuration file that contains the integration points of a certain pair of views. As we mentioned above, the integration points can be (semi-)automatically derived from the relationships between two views. Later on, the view integration algorithm just retrieves 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, $P << k \times l$). Nonetheless, this approach requires additional support, especially tool support, for automatically deriving and maintaining the integration points as well as keeping the configuration files up-to-date. These tasks are among our ongoing endeavours to complete the framework.

4 Case study

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

4.1 Process artefacts reuse scenario

As shown in Section 2, the billing renewal process has been developed so far using VbMF. Now, the company starts develop an order handling process such that internet customers can order the company's products via the website. Figure 4 shows the core functionality of the order handling process in terms of a BPMN diagram. The company opts to reuse existing artefacts as much as possible to develop the order handling process rather than starting from scratch. After analysing 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 and its functionality should be reused in the order handling process rather than being re-developed.

Figure 4 Overview of the order handling process (see online version for colours)

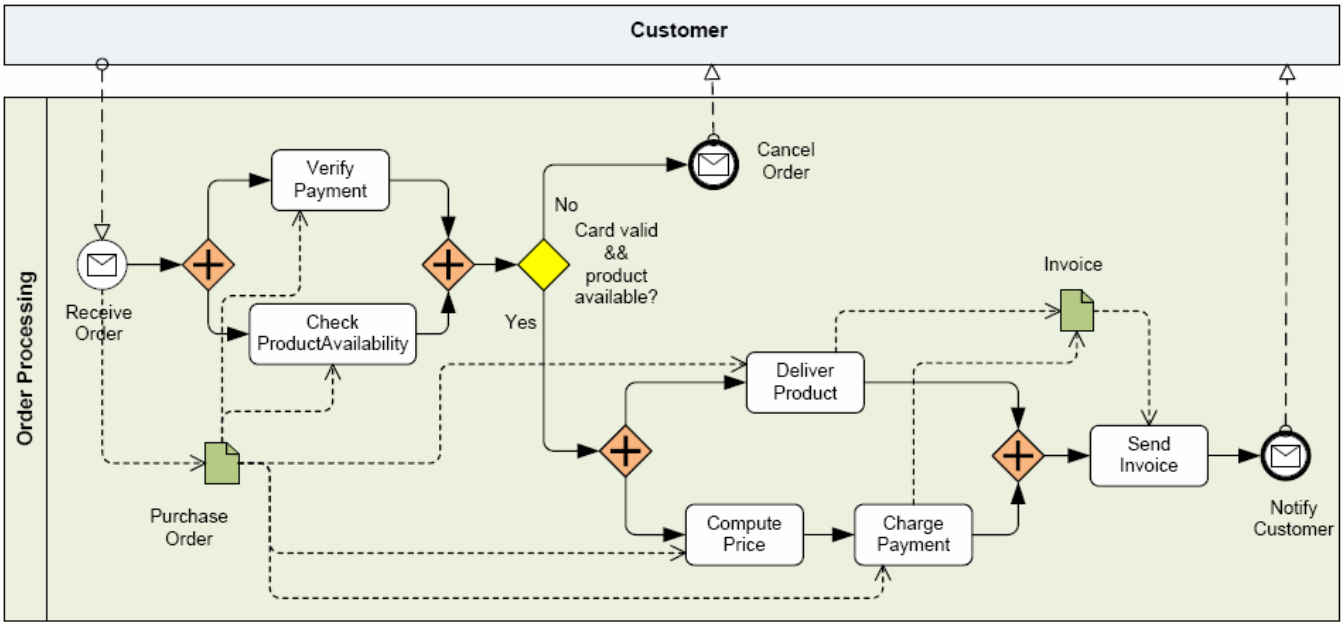


Figure 5 Name-based view integration approach for reusing by referencing the charge payment element of the billing renewal process in the order handling process (see online version for colours)

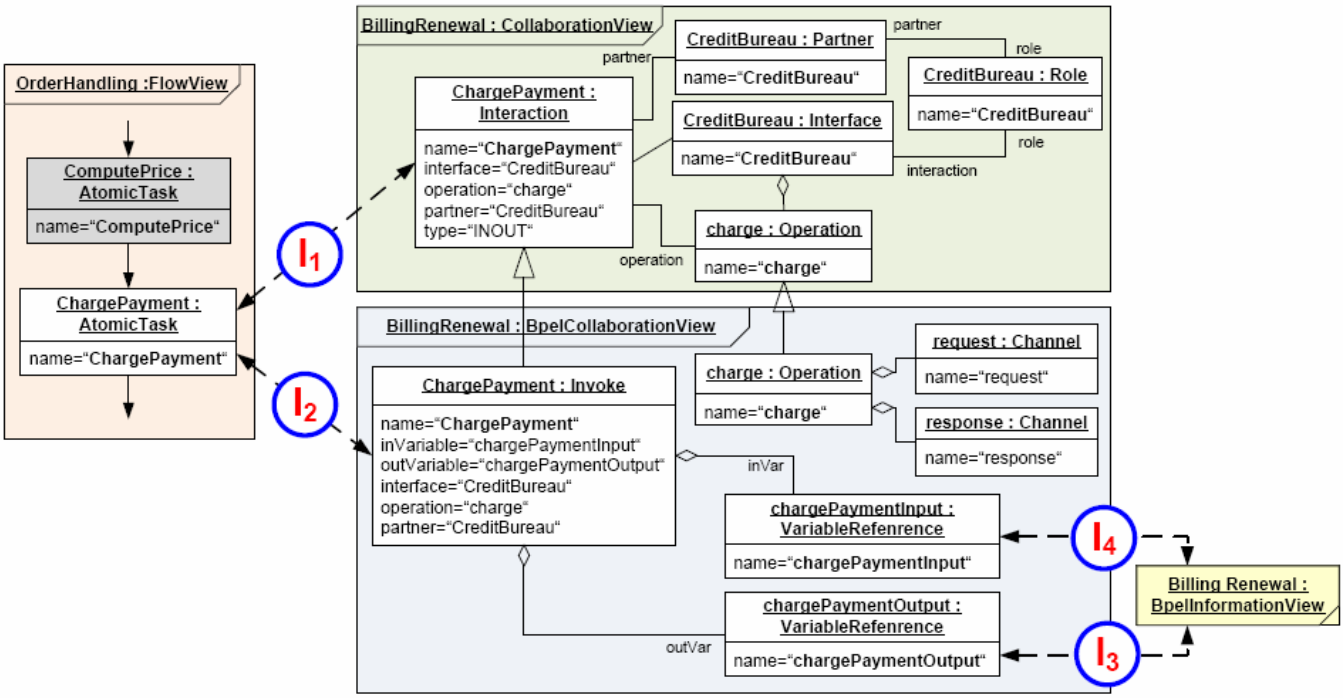


Figure 5 illustrates how the developers reuse the existing *ChargePayment* activity for modelling the order handling process. The scenario is presented in terms of UML object diagrams. On the right-hand side, we show the CV and BCV of the billing renewal process where the *ChargePayment* activity is defined at high-level and low-level of abstract, respectively. In the billing renewal CV, *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 BCV 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.

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 5.

2 Perform one of the following tasks (we note that these tasks can be supported by the framework in a (semi-)-automatic manner):

   a Explicitly define either an integration point $\mathbf{I}_1$ between the *ChargePayment:AtomicTask* and the *ChargePayment:Interaction* or $\mathbf{I}_2$ between the *ChargePayment:AtomicTask* and the *ChargePayment:Invoke*.

   b Explicitly specifies the CV and BCV of the billing renewal process are input views of the order handling process. As VbMF supports view integration by name-based matching (cf. Section 3), the integration points I_1 and I_2 can be implicitly resolved by VbMF tooling, i.e., the code generators.A question might be raised 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 (BIV). Therefore, these objects are part of the integration points $\mathbf{I}_3$ and $\mathbf{I}_4$, respectively, between the BCV and BIV 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 modelling framework:

1 Reuse the existing integration points between the BCV and BIV of the billing renewal process: The stakeholders can gain more benefit of reusability but they have to analyse the subsequent dependencies of the reused objects in the BIV. In addition, these subsequent dependencies also require extra effort to perform appropriate synchronisation when making any change in the reused views. This task can be (semi-)-automatically supported by our traceability approach in (Tran et al., 2009b).

2 Create new objects in the order handling BIV bearing the corresponding names, then $\mathbf{I}_3$ and $\mathbf{I}_4$ can be automatically derived. Although no benefit of reusability gained, there is also no binding to the billing renewal BIV. That is, no extra effort for understanding the subsequent dependencies or maintaining view synchronisation is required.

In summary, the separation of concerns principle realised in VbMF has isolated tangled process concerns by different domain of interests – (semi-)formalised views. The concept of integration point enables the flexibility of partially or totally reusing existing artefacts to develop new processes. As we explained during the development of the order handling process, each element of a certain process view might become a potential reusable artefact. This way, VbMF enables the stakeholders to gain more reuse of existing process development artefacts.

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 artefacts. To explore the application and pragmatic usage of our approach, appropriate experiments to quantitatively evaluating it in industrial business process development environment are definitely necessary. We note that 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 software reuse technique (Gaffney and Cruickshank, 1992; Frakes and Terry, 1996). We present in this section our quantitative evaluations of the reuse rate according to the model proposed by Gaffney and Cruickshank (1992) (which is called *the proportion of reuse*) as well as by Frakes and Terry (1996) (which is called *reuse percent*). 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 following 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 elements of the corresponding view/model (Gaffney and Cruickshank, 1992; Frakes and Terry, 1996).

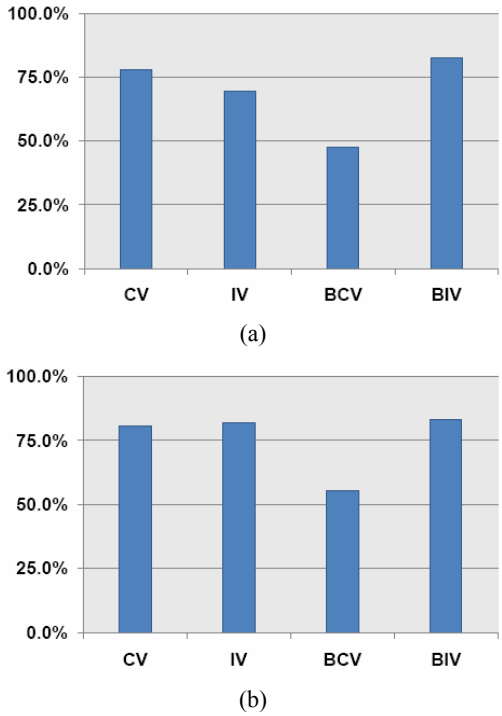
We have conducted the quantitative evaluation based on four processes extracted from industrial use cases. Two of them are the billing renewal and the order handling processes mentioned in the previous sections. Two other processes are the customer relationship management (CRM) fulfilment process (Evenson and Schreder, 2007) and the travel booking process (IBM, 2006). The CRM fulfilment process is part of the 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 modelling and implementation phases. In Table 1, we present the reuse rate R_R of VbMF views, such as CV, IV, BCV, and BIV, of each case study.

Table 1 The reuse rate of process view models in four use cases

Process	*CV*			*IV*			*BCV*			*BIV*		
	ER	*E*	*RR (%)*	*ER*	*E*	*RR (%)*	*ER*	*E*	*RR (%)*	*ER*	*E*	*RR (%)*
Billing renewal	49	63	*77.78*	59	85	*69.41*	63	132	*47.73*	407	494	*82.39*
CRM fulfilment	60	74	*81.08*	63	78	*80.77*	74	131	*56.49*	448	537	*83.43*
Order handling	29	36	*80.56*	36	44	*81.82*	36	65	*55.38*	238	286	*83.22*
Travel booking	27	33	*81.82*	33	43	*76.74*	33	56	*58.93*	219	260	*84.23*

As illustrated in the previous development and reuse scenario, each element of VbMF process views is potentially reusable artefact. 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, i.e., 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.

Figure 6 The reuse rate of view models in the billing renewal and order handling processes, (a) billing renewal process (b) order handling process (see online version for colours)



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 CV and InformationView such as *interaction* and *data handling*, or elements of the BCV and BIV, 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 while none of the (Bpel)InformationView's element depends on other views' elements. As a result, the ratio of reuse of the (Bpel)InformationView is 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: approximately 80% for the CV, 70% for the IV, 50% for the BCV, and 80% for the BIV. Because the reuse rates of view models of each use case is almost identical to those of the others, we only show the visualisations of the evaluation results of the billing renewal and order handling processes (see Figure 6).

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 (Morisio et al., 2002; Krueger, 1992; Frakes and Kang, 2005). Several works in this field have 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 (Frakes and Kang, 2005). Yet, there have been very few investigations of reuse in the area of business process management, in particular, business process development.

As we mentioned above, most popular languages used for modelling 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. There are a number of recent studies aiming at addressing the aforementioned challenges. Hallerbach et al. (2010) propose an approach for dealing with the variability of process models that configures and manages various variants along with corresponding master process models. Similarly, another approach, namely, 'configurable process models', presented

by Gottschalk et al. (2008) can help identifying configurable elements of a business process model and enable stakeholders to choose an appropriate configuration in order to come up with a working business process. Our approach is different from these works as the stakeholders are able to work with the process representation through various perspective in terms of (semi-)formalised view models rather than considering business processes in a whole.

To the best of our knowledge, most 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. (2003) proposed several so-called workflow patterns that 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, analysing, understanding the control flow of business processes. Similarly, Schumm et al. (2010) present an approach based on the notion of process fragment that enables a flexible method for describing and integrating existing artefacts into business processes. Weber and Reichert (2008) present an approach they call process model refactoring that allows stakeholders to prepare process representations in such a way that reuse become easier afterwards. 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 (Tran et al., 2007, 2009a), 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 (2008) present a preliminary approach based on π-calculus and ontologies to provide richer representations of business process aspects such as function, information, organisation, etc. 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 this knowledge in the business process life cycle.

6 Conclusions

In the domain of process-driven SOAs, reusing existing development artefacts is hindered by various factors. First, the languages used for modelling 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 analyse, understand, and reuse them. Last but not least, there is still a lack of methods for flexibly integrating reusable artefacts.

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 artefacts can be achieved using our approach. Nonetheless, further endeavours 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 to exploit the benefit of reuse and enhancing the automation in reusing process development artefacts.

Acknowledgements

We are grateful to the anonymous reviewers for their constructive and truly helpful comments. This work was partially supported by the European Union FP7 project COMPAS, grant no. 215175 and the European Union FP7 project INDENICA, grant no. 257483.

References

Evenson, M. and Schreder, B. (2007) 'SemBiz Project: D4.1 use case definition and functional requirements analysis', available at http://sembiz.org/attach/D4.1.pdf.

Fichman, R. and Kemerer, C. F. (2001) 'Incentive compatibility and systematic software reuse', *J. Systems and Software*, Vol. 57, No. 1, pp.45–60.

Frakes, W. and Kang, K. (2005) 'Software reuse research: status and future', *IEEE Trans. Software Eng*., Vol. 31, No. 7, pp.529–536.

Frakes, W. and Terry, C. (1996) 'Software reuse: metrics and models', *ACM Comp. Surv*., Vol. 28, No. 2, pp.415–435.

Frankel, D. (2002) *Model Driven Architecture: Applying MDA to Enterprise Computing*, John Wiley & Sons, Inc., New York, NY, USA.

Gaffney, J. and Durek, T.A. (1989) 'Software reuse: key to enhanced productivity: some quantitative models', *Information and Software Technology*, Vol. 31, No. 5, pp.258–267.

Gaffney, J.E. and Cruickshank, R.D. (1992) 'A general economics model of software reuse', in *14th Int'l Conf. Software Eng. (ICSE)*, ACM Press, pp.327–337.

Gottschalk, F., van der Aalst, W.M.P., Jansen-Vullers, M.H. and Rosa, M.L. (2008) 'Configurable workflow models', *Int. J. Cooperative Inf. Syst*., Vol. 17, No. 2, pp.177–221.

Hallerbach, A., Bauer, T. and Reichert, M. (2010) 'Capturing variability in business process models: the Provop approach', *Journal of Software Maintenance*, Vol. 22, Nos. 6–7, pp.519–546.

Holmes, T., Tran, H., Zdun, U. and Dustdar, S. (2008) '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, pp.246–261.

IBM (2006) 'Business process use cases', available at http://publib.boulder.ibm.com/bpcsamp (accessed on 2008/01/05).

IEEE (1990) *Standard Glossary of Software Eng. Terminology*.

Krueger, C.W. (1992) 'Software reuse', *ACM Comp. Surv*., Vol. 24, No. 2, pp.131–183.

Markovic, I. and Pereira, A.C. (2008) 'Towards a formal framework for reuse in business process modeling', in *BPM Workshops Advances in Semantics for Web Services 2007 (Semantics4ws'07)*, Springer, pp.484–495.

Mayr, C., Zdun, U. and Dustdar, S. (2008) 'Model-driven integration and management of data access objects in process-driven SOAs', in *ServiceWave*, pp.62–73.

Morisio, M., Ezran, M. and Tully, C. (2002) 'Success and failure factors in software reuse', *IEEE Trans. Software Eng*., Vol. 28, No. 4, pp.340–357.

OMG (2003) 'Model-driven architecture', available at http://www.omg.org/mda (accessed on 2006/03/02).

Schumm, D., Leymann, F., Ma, Z., Scheibler, T. and Strauch, S. (2010) 'Integrating compliance into business processes process fragments as reusable compliance controls, in *Multikonferenz Wirtschaftsinformatik (MKWI)*, Universitätsverlag Göttingen, pp.2125–2137.

Tran, H., Holmes, T., Zdun, U. and Dustdar, S. (2009a) 'Modeling process-driven SOAs – a view-based approach', *Handbook of Research on Business Process Modeling*, IGI Global, Chapter 2.

Tran, H., Zdun, U. and Dustdar, S. (2009b) 'VbTrace: using view-based and model-driven development to support traceability in process-driven SOAs', *J. Softw. Syst. Model*, available at http://dx.doi.org/10.1007/s10270-009-0137-0.

Tran, H., Zdun, U. and Dustdar, S. (2007) '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)*, LNI, GI, Vol. 116, pp.105–124.

Tran, H., Zdun, U. and Dustdar, S. (2008a) '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, pp.55–66.

Tran, H., Zdun, U. and Dustdar, S. (2008b) 'View-based reverse engineering approach for enhancing model interoperability and reusability in process-driven SOAs', in *10th Int'l. Conf. Software Reuse (ICSR)*, Springer, pp.233–244.

van der Aalst, W., ter Hofstede, A.H.M., Kiepuszewski, B. and Barros, A.P. (2003) 'Workflow patterns', *Distributed and Parallel Databases*, Vol. 14, No. 1, pp.5–51.

Weber, B. and Reichert, M. (2008) 'Refactoring process models in large process repositories', in *Proceedings of the 20th International Conference on Advanced Information Systems Engineering, CAiSE '08*, pp.124–139, Springer-Verlag, Berlin, Heidelberg.