Inheritance of Business Processes: A Journey Visiting Four Notorious Problems

W.M.P. van der Aalst

Eindhoven University of Technology, Faculty of Technology and Management, Department of Information and Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
w.m.p.v.d.aalst@tm.tue.nl

Abstract. The new millennium is characterized by speed and distribution in every aspect of most business and organization undertaking. Organizations are challenged to bring ideas and concepts to products and services in an ever-increasing pace. Companies distributed by space, time and capabilities come together to deliver products and solutions for which there is any need in the global marketplace. This creates new opportunities but also poses new problems. In this paper we will address four problems directly related to changing business processes in the networked economy. Two of the problems addressed surface when existing workflow specifications are changed: the *dynamic change problem* and the *management information* problem. The third problem refers to *coordination problems in inter-organizational workflows*. The fourth problem becomes relevant when predefined workflow specifications are *customized* for a particular business situation. This paper will show that these problems have common characteristics. Moreover, we will point out that an approach based on inheritance of dynamic behavior provides a partial solution for each of the problems.

1 Introduction

Inheritance is one of the cornerstones of object-oriented programming and object-oriented design. The basic idea of inheritance is to provide mechanisms which allow for constructing *subclasses* that inherit certain properties of a given *superclass*. This paper focuses on workflow processes [1, 24, 31, 43]. Therefore, a *class* corresponds to a *workflow process definition* (i.e., a routing diagram) and *objects* (i.e., instances of the class) correspond to *cases*. In most object-oriented methods a class is characterized by a set of *attributes* and a set of *methods*. Attributes are used to describe properties of an object (i.e., an instance of the class). Methods symbolize operations on objects (e.g., create, destroy, and change attribute). The structure of a class is specified by the attributes and methods of that class. Note that the structure only refers to the static aspects of the interface. The dynamic behavior of a class is either hidden inside the methods or modeled explicitly (in UML the life-cycle of a class is modeled in terms of statecharts [50]). Although the dynamic behavior is an intrinsic part of the class description (either explicit or implicit), inheritance of dynamic behavior is not well-understood. (See [13, 14] for an elaborate discussion on this topic and pointers to related work. Examples of alternative approaches are given in [38, 57, 58].) Given the widespread use of inheritance concepts/mechanisms for the static aspects, this is remarkable. Every object-oriented

programming language supports inheritance with respect to the static structure of a class (i.e., the interface consisting of attributes and methods). Since workflow management aims at supporting business processes, these results are not very useful in this context. However, the work presented in [4, 6, 13, 14] deals with inheritance of dynamic behavior in a comprehensive manner. This work is based on a particular class of Petri nets: the so-called sound workflow nets defined in [1]. This class of Petri nets corresponds to workflow processes without deadlocks, livelocks, and other anomalies. Other inheritance-based approaches abstract from the causal relations between tasks/methods, i.e., the control or routing structure is not taken into account. Some of the workflow management systems available claim to be object-oriented and thus provide some support for inheritance. For example, the workflow management system InConcert [30] allows for building workflow class hierarchies. Unfortunately, inheritance is restricted to the attributes and the structure of the process is not taken into account. Many workflow management systems have been implemented using object-oriented programming languages. However, these systems do not offer object-oriented mechanisms such as inheritance to the workflow designer or the designer has to program code to benefit from the object-oriented features provided by the host language. Nevertheless, we think that inheritance is a very useful concept for workflow management. Therefore, we advocate the use of the inheritance notions presented in [4, 6, 13, 14] for the four workflow related problems already mentioned in the abstract:

1. How to deal with the *dynamic change problem*? (Avoiding consistency problems when migrating cases from one process to another.)
2. How to deal with the *management information problem*? (Providing aggregate management information of work in progress in the presence of many versions/variants of the same workflow process.)
3. How to handle *coordination problems* in *inter-organizational* workflows by enforcing local consistency rules?
4. How measure the *difference* between two processes? (Delta analysis to estimate the effort required to customize a process.)

The goal of this paper is to demonstrate that several problems related to workflow design and analysis can be addressed using the principle of inheritance. This paper does not provide new scientific results but integrates existing results which have been published in several papers [3, 5–7, 10, 14]. This way we hope to reveal commonalities and show the relationships between four practical problems and our work on inheritance.

2 Problems

In this paper, we focus on business processes which can be characterized as workflow processes. Workflows are *case-based*, i.e., every piece of work is executed for a specific case: an order, an insurance claim, a tax declaration, etc. The objective of a workflow management system is to handle these cases (by executing *tasks*) as efficiently and effectively as possible. The *workflow process definition* specifies which tasks need to be executed and in what order. When a task is executed for a case, this is usually done by using one or more resources, e.g., a machine, an employee, etc. In this paper, we use

Petri nets [48] to represent workflow process definitions as indicated in[1]. For more papers on the application of Petri nets to workflow management we refer to [8, 11, 12, 17, 19, 22, 23, 29, 42, 49].

A Petri net represents a workflow if and only if it has exactly one starting place (*source*) and exactly one end place (*sink*) and the net obtained by adding a transition with the sink as the only input place and the source as the only output place is strongly connected. The latter is the case if for every two nodes x and y in the net, there is a path from x to y. Petri nets with this particular structure are called *workflow nets*. Each task is modeled by a transition. Tasks are connected by places (represented by circles) to specify the ordering of tasks. Places may contain tokens (represented by black dots). The *state*, often called *marking*, is the distribution of tokens over places. A transition, i.e., a task, is enabled if and only if each of the input places contains a token. Enabled transitions can fire while removing tokens from the input places and putting tokens on the output places. A detailed description of the class of workflow nets is beyond the scope of this paper and not needed for the remainder. However, some basic knowledge of Petri nets is needed to fully understand the concepts.

2.1 Problem 1: Dynamic change

At the moment, there are more than 200 workflow products commercially available [24] and many organisations are introducing workflow technology to support their business processes. It is widely recognised that workflow management systems should provide flexibility [7–9, 12, 16, 21, 28, 33, 36, 37, 47, 56]. However, today's workflow management systems have problems dealing with *changes*, e.g., new technology, new laws, and new market requirements may lead to (structural) modifications of the workflow process definition at hand. In addition, ad-hoc changes may be necessary, e.g., because of exceptions. The inability to deal with various changes limits the application of today's workflow management systems.

In this paper, we restrict ourselves to changes in the process perspective. In the process perspective there are basically two types of changes:

- *Individual (ad-hoc) changes*, i.e., ad-hoc adaptation of the workflow process: A single case (or a limited set of cases) is affected. A good example is that of a hospital: If someone enters the hospital with serious hart problems, you are not going to ask him for his ID, although the workflow process may prescribe this. For these ad-hoc changes one can distinguish between entry time changes (changes that occur when a case is not yet in the system) and on-the-fly changes (while in the system, the process definition for a case changes).
- *Structural (evolutionary) changes*, i.e., evolution of the workflow process: All new cases benefit from the adaptation. A structural change is typically the result of a BPR effort. An example of such a change is the change of a 4-year curriculum at a university to a 5-year one.

While executing a change there are typically running cases in the system. Figure 1 shows three ways to deal with these active workflow instances: (a) *restart*: running cases are rolled back and restarted at the beginning of the new process, (b) *proceed*: changes

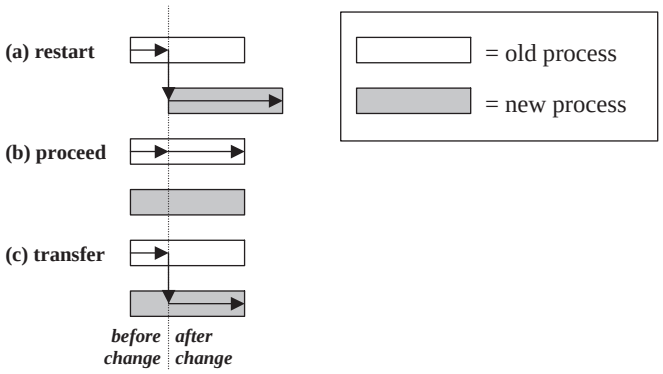

Fig. 1. How to handle running cases?

do not affect running cases by allowing for multiple versions of the process, and (c) *transfer*: a case is transferred to the new process. The term *dynamic change* [21] is used to refer to the latter policy. Restarting cases causes no real difficulties except that it is often difficult to rollback the tasks that have already been executed. The proceed policy also causes hardly any problems. In fact, it is the only policy truly supported by today's commercial workflow management systems. The only policy that causes serious theoretical and practical problems is the transfer of cases. The term dynamic change refers to the problem of handling old cases in a new process, e.g., how to transfer cases to a new, i.e., improved, version of the process.

Figure 2 illustrates the dynamic change problem. If the sequential workflow process (left) is changed into the workflow process where tasks B and C can be executed in parallel (right) there are no problems, i.e., it is always possible to transfer a case from the left to the right. The sequential process has five possible states and each of these states corresponds to a state in the parallel process. For example, the state with a token in $s3$ is mapped onto the state with a token in $p3$ and $p4$. In both cases, tasks A and B have been executed and C and D still need to be executed.

Now consider the situation where the parallel process is changed into the sequential one, i.e., a case is moved from the right-hand-side process to the left-hand-side process. For most of the states of the right-hand-side process this is no problem, e.g., a token in $p1$ is moved to $s1$, a token in $p3$ and a token in $p4$ are mapped onto a single token in $s3$, and a token in $p4$ and a token in $p5$ are mapped onto a single token in $s4$. However, the state with a token in $p2$ and $p5$ (A and C have been executed) causes problems because there is no corresponding state in the sequential process (it is not possible to execute C before B). This simple example shows that it is not straightforward to migrate old cases to the new process after a change. Some authors have proposed a solution for the dynamic change problem [12, 18, 20]. However, these solutions either require human intervention or are restricted to workflows with a particular structure.

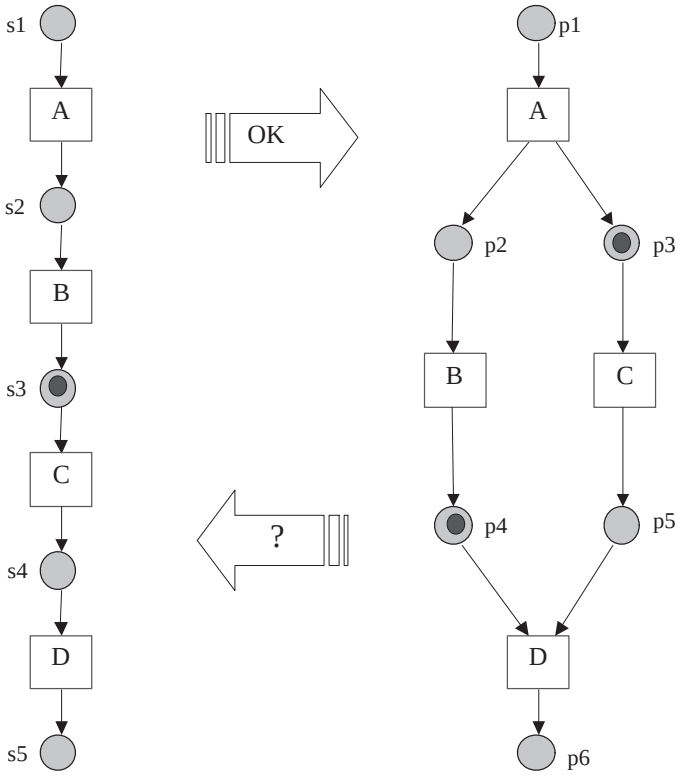

Fig. 2. The dynamic change problem.

2.2 Problem 2: Management information

Another problem of change is that it typically leads to multiple variants of the same process. For evolutionary (i.e., structural) change the number of variants is limited. Ad-hoc changes may lead to the situation where the number of variants may be of the same order of magnitude as the number of cases. To manage a workflow process with different variants it is desirable to have an aggregated view of the work in progress. Note that in a manufacturing process the manager can get a good impression of the work in progress by walking through the factory. For a workflow process handling digitized information, this is not possible. Therefore, it is of the utmost importance to supply the manager with tools to obtain a condensed but accurate view of the workflow processes. Figure 3 shows a workflow process with two variants: a sequential one (left) and a parallel one (middle). The numbers in the places indicate the number of cases in a specific state, e.g., in the sequential process there are 3 cases in-between task B and task C, and in the parallel process there are 2 cases in-between A and B. Since the manager requires an aggregated view rather than a view for every variant of the workflow process, the cases need to be mapped onto a generalized version of the different processes. A solution is to find the Greatest Common Divisor (GCD) or the Least Common Multiple (LCM) for the two processes shown. Finding the GCD or LCM of a set of processes is difficult and different definitions can be used (cf. [5]). Since all the states of the sequential process can be represented in the parallel process, we may choose the parallel process to present management information. Other choice are possible. However, let us assume that the GCD of the sequential process and the parallel process is indeed the parallel one. Figure 3 shows the aggregated view of the two workflow processes (right). For all places in the right-hand-side process except $m3$, it is quite straightforward to verify that the numbers are correct. The number of tokens in place $m3$ corresponds to the number of cases in-between A and C. In the sequential process, there are 1+3=4 cases in-between A and C. In the parallel process, there are also 4 cases in-between A and C, which brings the total to 8. For this small example, it may seem trivial to obtain this information. However, in general there are many variants of processes which may have up to 100 tasks and it is far from trivial to present aggregated information to the manager. The topic of generating management information was addressed in [5–7, 55, 56]. Despite its relevance for the next generation of workflow management systems only few researchers seem to be working on this topic.

2.3 Problem 3: Inter-organizational interface agreements

E-commerce refers to the enabling of purchasing and selling of goods and services through a communications network [32, 59]. The ability to conduct business activities involved in marketing, finance, manufacturing, selling and negotiation, electronically, is what E-commerce is all about. One major objective of adopting E-commerce strategies is to reduce costs and improve the efficiency of business processes, by replacing paper business with electronic alternatives. E-commerce, in its earliest incarnation known as *Electronic Data Interchange* (EDI), has been traditionally used by larger corporations to share and exchange information between business partners and suppliers using private networks. EDI enables the exchange of business data from one computer

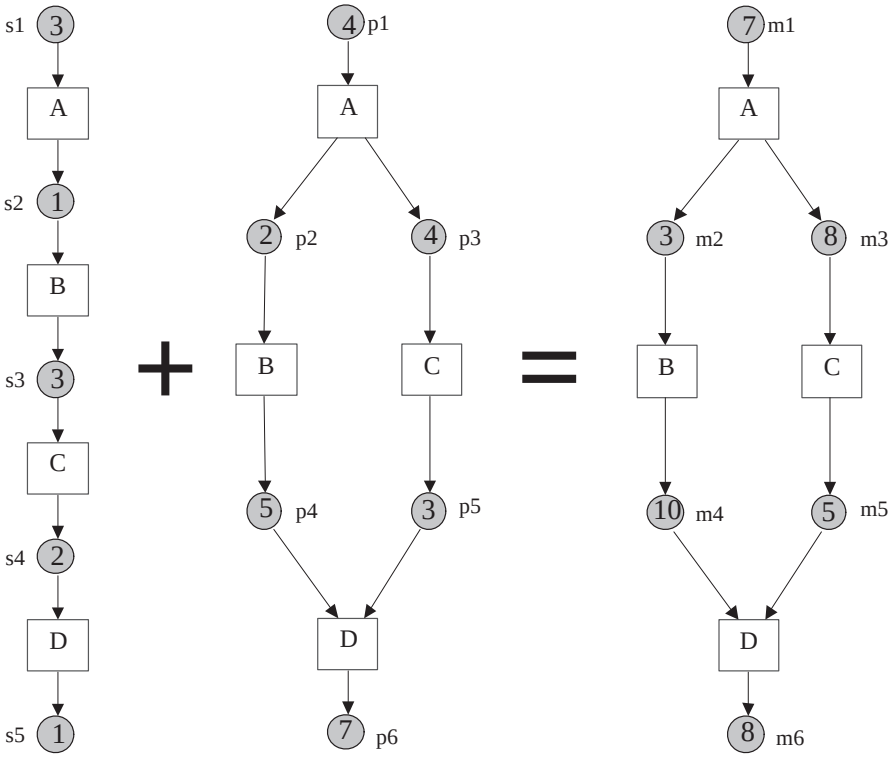

Fig. 3. Mapping cases in different processes onto one workflow process.

to another computer. It eliminates the need to re-key information from documents or messages by supporting the creation of electronic versions of documents or messages using public standard formats, which can then be transmitted, received and interpreted by other systems. Typical applications were supply-chain management processes like order placement and processing. However with the explosive growth of the Internet in the last couple of years, E-commerce is now able to offer solutions for a much broader range of business processes than EDI previously addressed. Also, the extensive availability of the Internet has enabled smaller companies, hindered previously by the large financial investment required for these private networks, to conduct business electronically. Technologies like bar coding, automatic teller machines, email, fax, video-conferencing, workflow and the world-wide-web have continued to impact the success of E-commerce. Although the term E-commerce frequently refers to on-line retailing involving businesses and consumers, experts predict that as E-commerce continues to grow, business-to-business E-commerce will continue to enjoy the lion share of the revenue. Business-to-business E-commerce has seen tremendous growth due to the globalization of the worldwide economy, which in turn is enabled in large part by the omnipresence of the Internet. Many corporations are extending their markets by mergers and strategic alliances with other companies throughout the world. Business processes of each of the business partners now become coupled in some way, creating inter-organizational workflow processes. Workflow systems enable the automated management and coordination of tasks, people, and resources involved in performing a business process, in a way that streamlines and improves the efficiency of the business process. They provide tools for modeling, enactment, administration, and monitoring of business processes. They, therefore, could be very useful in managing complex workflow processes such as those that involve multiple organizations, i.e., *inter-organizational workflows*. In particular the design of such workflow processes is often very complex and presents some challenges [15, 25, 26, 35, 39–42, 44, 45].

The third problem we introduce in this paper, is a problem caused by a malfunctioning of the coordination of two business partners participating in one common process. To describe this problem we us the following terms [10]:

- The *total workflow* is the whole inter-organizational workflow as it is actually executed.
- The *public workflow*, also referred to as *contract workflow*, is the business process the partners agreed on. This workflow only comprises tasks which are of interest to all business partners involved, i.e., it is a abstraction of the real workflow.
- A *private workflow* is a part of the total workflow executed by a specific business partner (also called *domain*).

One of the characteristics of inter-organizational workflow is that the companies involved do not know about the processes enacted inside the other domains. The only view they have of the others is the abstraction specified in the public workflow. Each of the business partners involved is responsible for a part of the public workflow. Tasks which are only of local interest are added without informing the other business partners. Tasks related to quality control, internal bookkeeping, and storage management are typical examples of tasks which are only of local interest. Local extensions of the workflow

may cause serious problems at the global level. Reordering tasks and adding or removing causal relations may cause deadlocks, livelocks, confusion, and other anomalies. Consider for example the public workflow shown in Figure 4. This workflow is partitioned over two domains: tasks A, B, C, and D are mapped onto domain L (left) and tasks E, F, G, and H are mapped onto domain R (right). The input place of the private workflow of domain R is $p8$ and $p11$ is the output place of this domain.

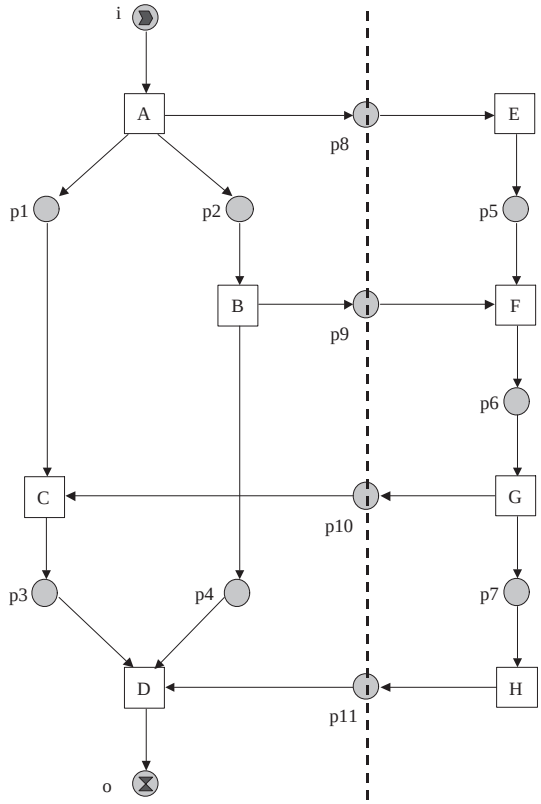

Fig. 4. The public workflow used to illustrate problems resulting from local change.

Both private workflows are modified to accommodate local needs. In domain L a causal relation is added between task C and task B to make sure that C is executed before B. In domain R the order in which F and G are executed is reversed. Figure 5 shows the two modified private workflows. Place $p12$ has been added to force the execution of C before B. From a local point of view, this change does not cause any problems. For domain L it is perfectly acceptable to reduce the degree of parallelism. However, this change leads to a global deadlock. On the one hand C should be executed before B (place $p12$) and on the other hand B should be executed before C (causal relation

via $p9$, $p6$, and $p10$). The circular dependency involving tasks C, B, F, and G causes the overall workflow process to deadlock in the state marking places $p1$, $p2$, and $p5$. Figure 5 also shows the modification of the private workflow of domain R: task F and task G are reversed. This change alleviates the problem caused by the addition of place $p12$. If both private workflows are modified as indicated in Figure 5 there would be no deadlock. However, the modification of the private workflow of domain R is also not acceptable. Based to the public workflow both partners agreed upon, the execution of task C may depend on the results of task F. By reversing the order of task F and task G there can be problems (e.g., missing data) because task C may be executed before task F. These examples show that local changes may cause global errors, i.e., modifications which are harmless from a local perspective may cause deadlocks, livelocks, missing data, etc. Note that even if both business partners know about each others fragment of the public workflow and take this knowledge into account, the anomalies such as described can occur (both make conflicting changes, e.g., serializing tasks differently).

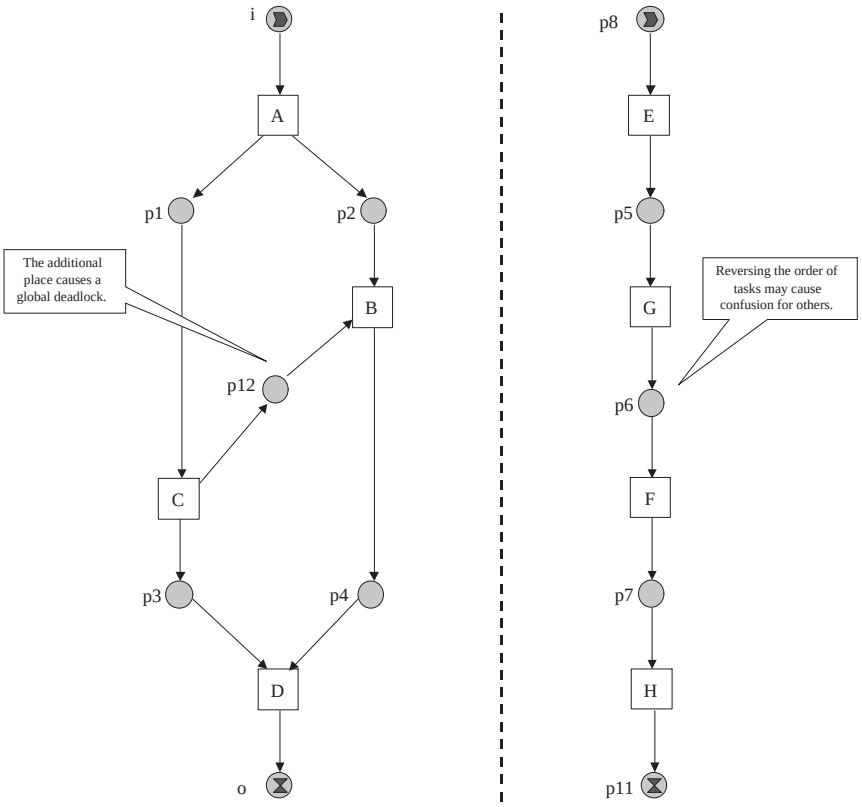

Fig. 5. Both private workflows have been modified.

2.4 Problem 4: Customizing business processes

Today's information systems support a variety of business processes. These information systems are based on general purpose software packages (e.g., workflow management systems), domain specific support systems (e.g., ERP - Enterprise Resource Planning - systems), software for a specific type of businesses (e.g., call centers, or hospitals), and company specific solutions. To avoid 're-inventing the wheel' companies are 'sharing' software by using standard solutions. As a result, the business process that is supported may differ from the business process actually desired. Consider for example today's generation of ERP systems, e.g., SAP R/3 and BaanERP. These systems are based on industry's 'best practices'. However, the 'best practices solution' may not apply to a specific company. Such a company, assuming that it already made the choice to use a particular ERP, has two choices: Either the best practices solution is used despite it shortcomings or the software is *customized* by reconfiguring or reprogramming parts of the functionality. To make this choice the costs of customization need to be assessed. Moreover, if the company is in the process of selecting a standard system (e.g., an ERP system), then the costs of customization amongst different systems need to be compared.

Suppose a company interested in evaluating the costs of customization has both a concise specification of the desired business processes (e.g., the current process) and a model of the business processes supported by the standard system. In this case, the challenge is to determine (automatically) the difference between both models. We use the term *delta analysis* for such an investigation [27]. However, delta analysis is not as simple as it may seem. Consider for example the two workflow processes shown in Figure 2. What is the difference between both processes? Just considering the tasks used/supported is not sufficient. If the routing differs, then the processes also differ. A sequential processes consisting of 50 tasks has little in common with a process where the same 50 tasks are executed in parallel or where only one of these tasks is selected. For delta analysis it is necessary to determine the commonalty of both processes. Only if it is clear where both processes "agree on", it is possible to point out differences. Consider the two workflow processes shown in Figure 6. What do they have in common? In a way we are again looking for the *Greatest Common Divisor* (GCD) of the two workflows (cf. Section 2.2).

The GCD of two workflow process is the part where both processes agree on [5]. Therefore, only tasks which are used in both processes can appear in the GCD. However, it is not clear whether all tasks which are used in both processes should appear in the GCD. The two workflow processes shown in Figure 6 agree on the role of task A and task D. Therefore, the GCD should at least comprise these two tasks. For the other two tasks this is less clear. Figure 7 shows four possibilities. If both B and C are present in the GCD, then it is reasonable to put them in parallel: All execution sequences possible in the sequential process are also possible in the parallel process (alternative (A) in Figure 7). However, one might argue that alternative (A) is not the GCD since both processes do not agree on the ordering of B and C. Since both workflow processes agree on the role of task B relative to task A and task D, we may just add task B to the GCD (alternative (B) in Figure 7). The same holds for task C (alternative (C) in Figure 7). Since the choice between alternative (B) and alternative (C) is rather arbitrary

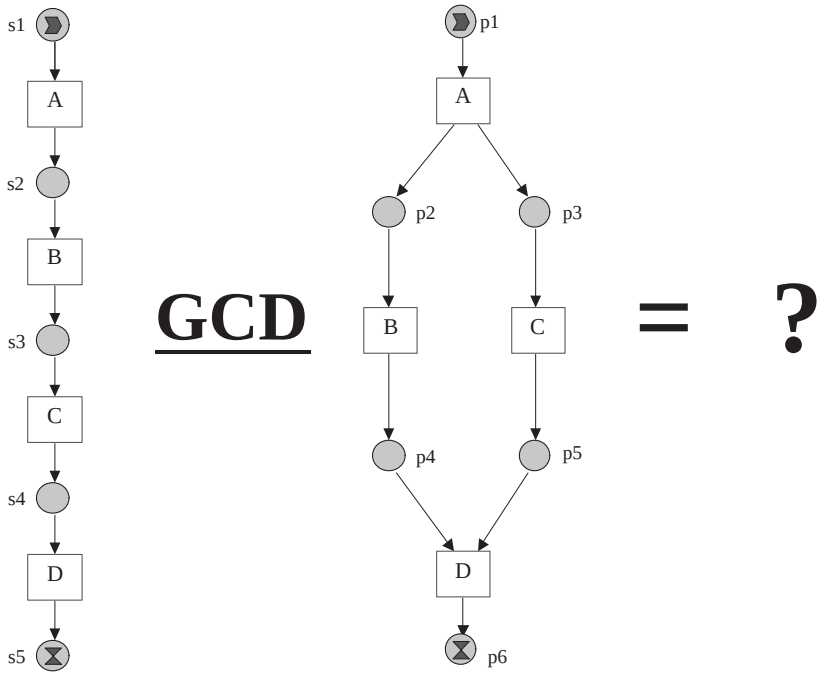

Fig. 6. What is the GCD of both processes?

one might argue that none of these tasks should be present (alternative (D) in Figure 7). This small example shows that the selection of a GCD is far from trivial. Note that the deliberations used when selecting a GCD are the same as when establishing the differences between two workflow processes. Therefore, a good definition of GCD is crucial for a structured delta analysis.

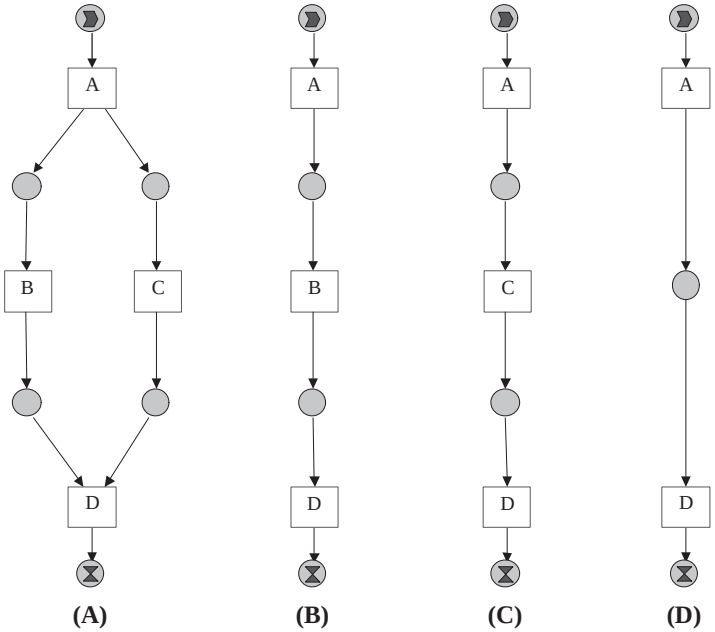

Fig. 7. Alternative GCD's?

Note that in order to do a delta analysis there have to be descriptions of both the desired (or current) process and the process as supported by the standard software. Given this observation it is interesting to point out the role of reference models in today's ERP systems. A good example of such a reference model is the ARIS (IDS Prof. Scheer) reference model of the SAP R/3 functionality [51, 34]. This reference model describes the business processes as they are (or could be) supported by SAP R/3. This model can be used to clarify and configure the SAP R/3 system. The Event-driven Process Chains (EPC's) used for specifying the business processes in ARIS and SAP R/3 are similar to Petri nets [2]. If a tool such as ARIS is also used to model the desired (or current) workflow process, then all the ingredients are there to do a delta analysis. Note that apart from issues such as determining the GCD, there are other problems such as the naming of tasks. Therefore, a good ontology is a prerequisite for any form of delta analysis.

3 Inheritance of dynamic behavior

To tackle the four problems identified in the previous section, we propose a solution based on inheritance. First we define four inheritance notions for workflow processes (i.e., workflows specified by workflow nets). Consider two workflow processes x and y. When is x a subclass of y? x is a subclass of superclass y if x inherits certain features of y. Intuitively, one could say that x is a subclass of y if and only if x can do what y can do. Clearly, all tasks present in y should also be present in x. Moreover, x will typically add new tasks. Therefore, it is reasonable to demand that x can do what y can do with respect to the tasks present in y. In fact, the behavior with respect to the existing tasks should be identical.

In [4, 14] we have identified four different notions of inheritance: *protocol inheritance*, *projection inheritance*, *protocol/projection inheritance*, and *life-cycle inheritance*. Protocol/projection inheritance is the most restrictive form of inheritance. If x is a subclass of y with respect to protocol/projection inheritance, then x is a subclass of y with respect to protocol inheritance *and* projection inheritance. Life-cycle inheritance is the most liberal form of inheritance, i.e., protocol and/or projection inheritance implies life-cycle inheritance.

The notion of projection inheritance is based on *abstraction*: *If it is not possible to distinguish x and y when arbitrary tasks of x are executed, but when only the effects of tasks that are also present in y are considered, then x is a subclass of y with respect to projection inheritance*.

For distinguishing x and y under projection inheritance we only consider the tasks present in both nets (i.e., in y). All other tasks in x are renamed to τ. One can think of these tasks as silent, internal, or not observable. Since branching bisimulation [13] is used as an equivalence notion, we abstract from transitions with a τ label, i.e., for deciding whether x is a subclass of y only the tasks with a label different from τ are considered. The behavior with respect to these tasks is called the *observable behavior*. Added tasks (i.e., tasks present in x but not in y) can be executed but are not observable by the outside world, i.e., projection inheritance conforms to hiding or abstracting from tasks new in x.

The notion of protocol inheritance is based on *encapsulation* (i.e., blocking of new tasks): *If it is not possible to distinguish x and y when only tasks of x that are also present in y are executed, then x is a subclass of y*.

For distinguishing x and y under protocol inheritance all tasks present in x but not in y are blocked. The new tasks are simply disallowed to be executed.

A formal definition of the four forms of inheritance is beyond the scope of this paper. (The definition builds on branching bisimulation equivalence and an abstraction operator which renames a given set of tasks to τ.) The interested reader is referred to [4, 6, 13, 14].

Figure 8 shows five workflow processes modeled in terms of workflow nets. Special symbols are used to indicate the source place (i) and the sink place (o). Workflow process (A) consists of three sequential tasks: *register*, *handle*, and *archive*. Each of the other workflow processes extends this process with one additional task: *check*. In workflow process (B) task *check* can be executed arbitrarily many times between *register* and *handle*. Workflow process (B) is a subclass of workflow process (A) with respect to

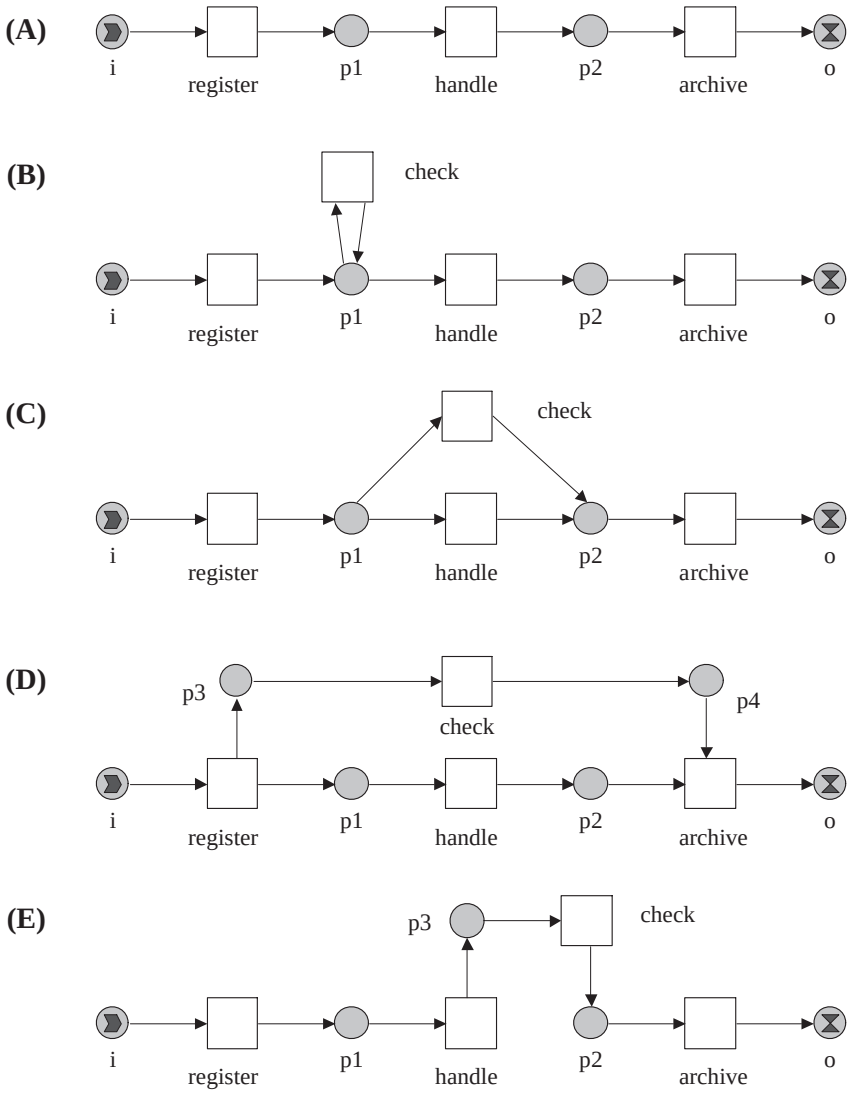

Fig. 8. Five routing diagrams describing variants of a simple workflow process.

projection inheritance: If task *check* is abstracted from, then the two processes behave equivalently (i.e., are branching bisimilar). Workflow process (B) is also a subclass of workflow process (A) with respect to protocol inheritance: Blocking task *check* yields two equivalent processes. Workflow process (C) is not a subclass with respect to projection inheritance: Hiding task *check* introduces the possibility to skip task *handle* and thus change the actual behavior. However, (C) is a subclass of (A) with respect to protocol inheritance. Workflow process (D) is a subclass of workflow process (A) with respect to projection inheritance: Hiding this task results in two equivalent processes. However, (D) is not a subclass of (A) with respect to protocol inheritance: Blocking task *check* results in a deadlock. Workflow process (E) is a subclass of workflow process (A) with respect to projection inheritance: The detour via task *check* can be hidden thus yielding an observable behavior identical to (A). Workflow process (E) is not a subclass of workflow process (A) with respect to protocol inheritance. All workflow processes are a subclass of (A) with respect to life-cycle inheritance. For life-cycle inheritance some of the new tasks are blocked and others are hidden to obtain two equivalent processes. Only workflow process (B) is a subclass with respect to protocol/projection inheritance.

In [4, 6, 13, 14] we proposed a number *inheritance-preserving transformation rules*. These rules correspond to frequently used design constructs and preserve one or more of the four inheritance notions. A detailed description of these rules is beyond the scope of this paper. Therefore, we give an informal description of four inheritance rules: PP, PT, PJ, and PJ3. Protocol/projection inheritance-preserving transformation rule PP is illustrated by Figure 9. New transitions (i.e., tasks) and places (i.e., conditions) are added to the original workflow net such that tokens are only temporarily removed from a place in the original net. The added subnet may have any structure as long as it is guaranteed that any token taken from place p will be returned eventually and no tokens are left in the subnet. Since the subnet only postpones behavior, the extended workflow net (right) is a subclass of the original workflow net (left) under both projection and protocol inheritance. If one abstracts from the newly added tasks or blocks the newly added tasks, one cannot find any differences between both nets.

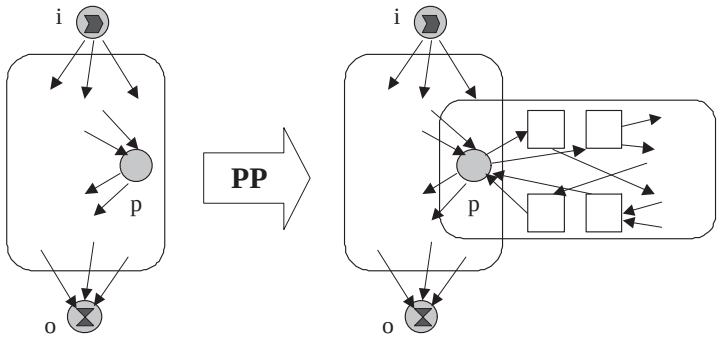

Fig. 9. Inheritance-preserving transformation rule PP.

Figure 10 illustrates transformation rule PT. PT preserves protocol inheritance and adds alternative behavior. The added subnet removes tokens from the original net to execute tasks not present in the original net. The added subnet may have any structure as long as any token taken from place p_i (input place) will be returned in place p_o (output place) eventually and no tokens are left in the subnet. Other requirements are that new tasks consuming tokens from p_i should not appear in the original net and that the routes via the subnet do not create new states in the original net. It is easy to see that PT preserves protocol inheritance: The added subnet is never activated if all new tasks are blocked. Note that PT does not preserve projection inheritance since the subnet can be used to bypass parts of the original net.

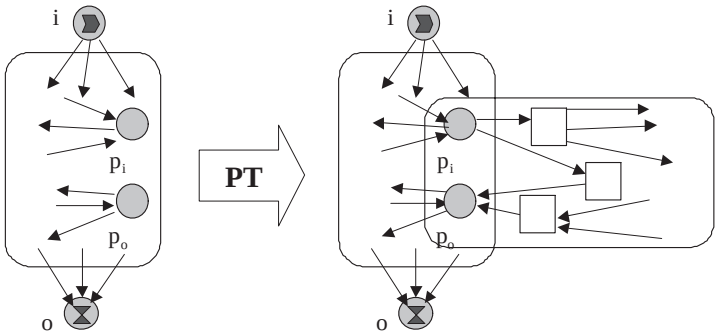

Fig. 10. Inheritance-preserving transformation rule PT.

Figure 11 shows inheritance-preserving transformation rule PJ. Rule PJ inserts new tasks in-between a task t_p and a place p in the original workflow net. In fact, rule PJ can be used to insert an arbitrary subflow in-between t_p and p. The added subnet may have any structure as long as it is guaranteed that once the subnet is activated by firing t_p eventually a token is put in place p and no tokens are left behind. It is easy to see that the extended workflow net (right) is a subclass of the original workflow net (left) under projection inheritance: by abstracting from the newly added tasks the observable behaviors coincide.

Projection inheritance-preserving transformation rule PJ3 can be used to add parallel behavior. (The rule is named PJ3 for historical reasons.) Figure 12 illustrates this rule. The execution of task t_i activates the subnet containing new tasks to be executed in parallel. Task t_o synchronizes the original workflow net and the added subnet. The added subnet may use arbitrary routing constructs as long as (1) the execution of t_i is always followed by the execution of t_o in the original net and t_o is always preceded by t_i, (2) activation of the subnet via firing t_i is always followed by a state which marks the input places of t_o in the subnet, and (3) no tokens are left behind in the subnet after firing t_o. If these three requirements are guaranteed, then the extended workflow net (right) is a subclass of the original workflow net (left) under projection inheritance.

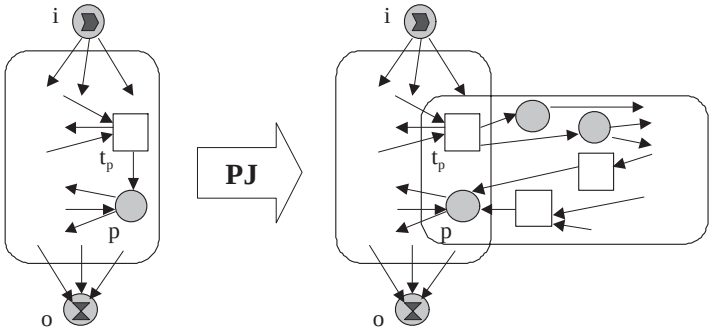

Fig. 11. Inheritance-preserving transformation rule PJ.

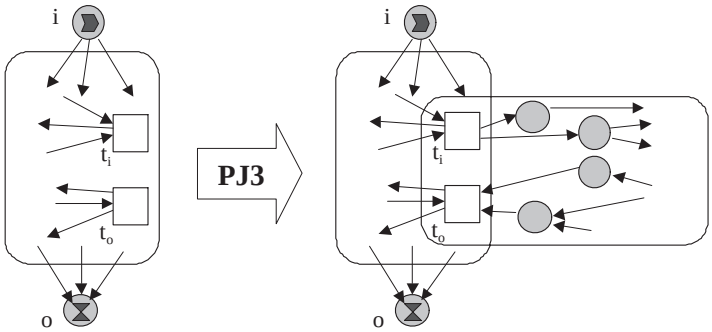

Fig. 12. Inheritance-preserving transformation rule PJ3.

For a formal definition of these rules we refer to [4, 6, 13, 14]. Details and subtle requirements are omitted to simplify the presentation of the main ideas. The workflow nets shown in Figure 8 illustrate the four rules. Rule PP introduces new tasks which only postpone behavior. Workflow process (B) shown in Figure 8 can be constructed from (A) by applying this rule; task *check* only postpones the execution of handle. Rule PT introduces alternative behavior. Workflow process (C) shown in Figure 8 can be constructed from (A) by applying PT. Rule PJ inserts new tasks in-between existing tasks. Workflow process (A) shown in Figure 8 can be extended to workflow process (E) using this rule. The extension can be a single task but also a complex subflow containing many tasks and all kinds of causality relations. Rule PJ3 adds parallel behavior. Workflow process (A) shown in Figure 8 can be extended to workflow process (D) using this rule. The four rules (PP, PT, PJ, and PJ3) correspond to design constructs that are often used in practice, namely iteration, choice, sequential composition, and parallel composition. If the designer sticks to these rules, inheritance is guaranteed!

4 Solution?

In the remainder we sketch in what way inheritance can assist in tackling the problems identified Section 2. We will not provide detailed solutions: For some of the problems we refer to other publications and for others we are exploring alternative solutions.

4.1 Problem 1: Dynamic change

The inheritance-preserving rules can be used to *avoid* the first problem indicated in this paper. Moreover, the four rules enable the designer to establish syntactic and semantic correctness in a compositional manner. For example, soundness can be verified by analyzing the original part and the extension separately. To tackle the dynamic change problem we use the *transfer rules* presented in [6]. Suppose that x is a subclass of y constructed using the rules PP, PT, PJ, and PJ3. For any state in workflow process y it is possible to transfer a case to x such that the transfer is instantaneous (i.e., no postponements needed) and does not introduce syntactic errors (e.g., deadlocks, livelocks, and improper termination) nor semantic errors (e.g., the double execution of tasks or unnecessary skipping of tasks). Moreover, it is also possible to transfer cases from subclass x to superclass y without any problems. Note that the transfer rules are derived from the transformation rules introduced earlier. The transfer rules to move a case to a subclass are: r_{PT}, r_{PP}, r_{PJ}, $r_{PJ3,C}$ and $r_{PJ3,P}$ (see [6]). Transfer rules r_{PT}, r_{PP}, and r_{PJ} are rather trivial because additional behavior (i.e., alternative branches or parts inserted in-between existing parts) is introduced without eliminating existing states. The transfer rule corresponding to transformation rule PJ3 is more complex because PJ3 adds parallel behavior rather than additional behavior. When adding parallel behavior, it may be necessary to mark places in the newly added parts. If this is the case, there is a choice to put the tokens in the beginning of the parallel part (conservative approach $r_{PJ3,C}$) or to put the tokens at the end of the parallel part (progressive approach $r_{PJ3,P}$). This choice depends of the desired policy. The transfer rules to move a case to a superclass

are: $r^{-1}_{PT,C}$, $r^{-1}_{PT,P}$, r^{-1}_{PP}, r^{-1}_{PJ}, and r^{-1}_{PJ3}. The transfer rule corresponding to transformation rule PJ3 is simple: Simply remove the parallel parts. Transfer rules r^{-1}_{PP} and r^{-1}_{PJ} move tokens from the extended part to the superclass part. For the transfer of cases to the superclass under transformation rule PT there are again two choices: a conservative approach ($r^{-1}_{PT,C}$) and a progressive approach ($r^{-1}_{PT,P}$). Again the choice depends of the desired transfer policy. Note that as long as the designer sticks to the inheritance preserving transformation rules, the transfer rules can be generated automatically. This means that there is no need to design complicated migration schemes.

To illustrate the transfer rules, we use the five workflow processes shown in Figure 8. Suppose a case is in variant (A) in the state corresponding to $p1$. If the case is migrated to variant (B), (C), or (E), then there is no need to modify the state (i.e., the state in the new process is just a token in $p1$). If the case is migrated to variant (D), the new state is either $p1 + p3$ (conservative mapping, $r_{PJ3,C}$) or $p1 + p4$ (progressive mapping, $r_{PJ3,P}$). A case in variant (E) in the state corresponding to $p3$, is mapped onto state $p2$ if it is transferred to (A). A case in variant (D) in the state corresponding to $p2 + p4$, is mapped onto state $p2$ if it is transferred to (A). These examples show that, as long as the designer sticks to the inheritance-preserving transformation rules, it is indeed possible to migrate cases from a superclass to a subclass and vice versa, i.e., the inheritance-preserving transformation rules can be used to avoid the dynamic change bug illustrated by Figure 2.

4.2 Problem 2: Management information

The transfer rules r_{PT}, r_{PP}, r_{PJ}, $r_{PJ3,C}$, $r_{PJ3,P}$, $r^{-1}_{PT,C}$, $r^{-1}_{PT,P}$, r^{-1}_{PP}, r^{-1}_{PJ}, and r^{-1}_{PJ3} can also be used to construct management information. Construct or select an appropriate workflow process such that each of the variants is a subclass or superclass of this processes via one or more inheritance-preserving transformation rules (applied in either direction). This workflow process is called the *management information net*. Any case residing in any of the variants can be mapped onto the management information net using the transfer rules. Therefore, the aggregate management information can be obtained automatically. Consider for example the following situation: There is one process template and every ad-hoc variant is a subclass of this variant constructed using the rules PP, PT, PJ, and PJ3. The template can also be used as a management information net and all cases can be projected onto this net using $r^{-1}_{PT,C}$, $r^{-1}_{PT,P}$, r^{-1}_{PP}, r^{-1}_{PJ}, and r^{-1}_{PJ3}. See [6] for technical details and more scenarios showing that the inheritance concepts can really help to provide aggregate management information.

4.3 Problem 3: Inter-organizational interface agreements

In general, modeling workflow processes that span multiple organizations can be complex. Many workflow-modeling tools require that the complete details of the entire process be fixed at the design stage. This approach is often the bottleneck in workflow design, because it requires each participating business partner to understand the nature of their partners' local processes which makes things even more complicated. This is neither necessary nor desirable. Partners should be able to agree on their business

process at a level that abstracts from irrelevant and confusing details. Furthermore, organizations should be able to participate towards the completion of a business process, while at the same time be at liberty to construct their private processes in any way that places them at best advantage.

The approach we propose here for designing interorganizational workflows, is a four-step process that involves creation of a public process, partitioning the public process amongst the partners and allowing for modification by the individual partners of their parts of the process to create private processes. This approach is based on the inheritance-preserving transformation rules and is described in more detail in [3, 10]. The steps are:

1. *Design public workflow process.*
   In the first step, the partners agree on the overall structure of the common business process. The key tasks are identified as well as the interfaces between them. The interfaces are specified as control and data dependencies. Process repositories may also be used at this stage to aid the design process. The workflow process model resulting from this stage will not contain organization specific information like role assignment, resources assignment, etc. We refer to the workflow process represented by this model as the *public process*.
2. *Partition the public workflow process definition amongst business partners.*
   In this step, the partners are assigned to be responsible for completing parts of the process. The workflow net is then partitioned along organizational lines. The partitioning creates a set of workflow net fragments for each business partner. Each set contains fragments of the public workflow net that the partner is responsible for.
3. *Create a private workflow for each business partner.*
   The partitioning stage creates sets that are not necessarily workflow nets, but rather fragments of a workflow net. A fragment may have multiple input or output places, or no input or output places at all. Moreover, a fragment may contain disjunct subsets (i.e. is unconnected). In other words, the fragment may not be a workflow net (i.e., the requirements mentioned earlier are violated). This step automatically creates a workflow net from each fragment set for each of the partners to manage locally in their organization. Note that in this step we abstract from places solely dedicated to message exchange between fragments, i.e., we remove all source places except the initial one and remove all sink places except the last one in the fragment. If after this abstraction we do not obtain a workflow net, additional modifications by connecting the fragments by introducing arcs and implicit places are needed. The result is a sound workflow net whose behavior preserves the interface of the original public process. We will refer to each of the local workflows as *private* processes
4. *Modify private processes using inheritance preserving transformation rules.*
   Here, each business partner will modify their private process created in the previous step to accurately reflect their own business process and incorporate details of how the tasks will be implemented at their organization. Some of the details that will be added at this stage are task implementation, assignment of roles to the tasks, data objects to be used and any mapping of global data objects to these local data objects, business rules, deadline specification etc. These extensions or modifications to the

minimal workflow must also preserve the behavior of the global business process. To guarantee that local extensions and modifications do not disturb the behavior of the public process, we use a notion of inheritance of dynamic behavior. For this purpose, we bring *projection inheritance* into play. Private workflow processes are modified while preserving projection inheritance, i.e., one is allowed to transform the local workflow into an arbitrary subclass under projection inheritance. For this purpose three of the four transformation rules can be used: PP, PJ, and PJ3.

The result of this approach is a set of private workflows which together form the actual total workflow. This total workflow is a sound workflow net, i.e., anomalies such as deadlocks and dangling references cannot occur. Moreover, the total workflow is a subclass of the private workflow under projection inheritance. This means that the workflow actually executed is consistent with the abstract workflow the business partners agreed on in the first place. These strong results show that the notion of inheritance can be used to tackle Problem 3. See [3, 10] for more details.

4.4 Problem 4: Customizing business processes

The fourth problem to be tackled by the inheritance notions defined earlier is the problem of delta analysis, i.e., given two workflow processes: What is the difference between those processes and how much does it cost to customize a process such that it coincides with the other? One of the core problems is to decide where both processes agree on, i.e., to determine the GCD (Greatest Common Divisor). In Figure 7, we showed four possible candidates for the GCD of the two workflow processes shown in Figure 6. Based on the inheritance relations we can experiment with various definitions of the GCD. The most straightforward definition of the GCD of two workflow processes x and y is the "smallest" superclass of both x and y under life-cycle inheritance, i.e., the x is a subclass of the GCD, y is a subclass of the GCD, and there is no workflow process z such that z is a real subclass of the GCD and a superclass of both x and y. Based on this definition it can be shown that for any set of workflow processes there exists a GCD. However, in some cases there may be multiple GCD's. Consider for example the two workflow processes shown in Figure 6. Workflow process (B) is a GCD of these two processes: Process (B) is a superclass of both processes shown in Figure 6 and there is no "smaller" workflow process satisfying the same conditions. However, for similar reasons, process (C) is also a GCD. If we closely observe the two processes shown in Figure 6, it may seem reasonable to have two GCD's. Both processes agree on the fact that tasks B and C are executed in-between tasks A and D. However, they do not agree on the ordering of B and C. Therefore, either B or C is added to the GCD.

The concept of GCD was introduced in [6] and a detailed analysis of this concept is given in [5]. Since none of the inheritance relations is a lattice, there is a trade-off between "uniqueness" and "existence". By using a weak notion of GCD existence is guaranteed but there may be multiple GCD's. By using a stronger notion, existence is no longer guaranteed but if the GCD exists, it is unique. Given this tradeoff, we define the notion of Maximal Common Divisor (MCD). An MCD is a "smallest" superclass of both x and y under life-cycle inheritance. Both (B) and (C) (Figure 7) are MCD's of the two workflow processes shown in Figure 6. Given any set there is at least one MCD.

In [5] we reserve the term GCD for a stronger notion: z is the GCD of x and y if and only if x is a subclass of z, y is a subclass of z, and for any v such that x is a subclass of v and y is a subclass of v: z is a subclass of v. If there is a GCD using this stronger notion, it is unique.

In [5, 6] conditions are given such that the MCD of a set of workflow processes is a GCD, and therefore, unique. Further research is needed to evaluate the usefulness of this particular notion of GCD for delta analysis. Given the fact that inheritance defines a relation on processes, it is useful for comparing workflows. Note that each of the four inheritance relations defines a partial order, i.e., each relation is reflexive, anti-symmetric, and transitive [14]. Therefore, inheritance is a sound basis for defining a notion of GCD. If a set of workflow processes is related under inheritance via subclass-superclass relationships, it is generally quite easy to find the GCD. If this is not the case (such as in Figure 6), the computation of a GCD is more involved and there are typically multiple candidates (i.e., MCD's). As pointed out when defining the problem of delta analysis, computing the GCD is just one of the ingredients needed to calculate the costs of customization. For example, the naming of tasks is important (i.e., a good ontology is a necessity) and it may be difficult to establish a suitable cost structure (What is the cost of adding/deleting a causal relation compared to adding/deleting/changing a task?). See [27] for a more elaborate discussion on these other aspects.

5 Conclusion and tool support

In this paper we visited four notorious problems related to the design and analysis of workflow processes:

- The problem of *dynamic change*: How to migrate workflow instances from the old to the new process?
- The problem of *management information*: How to provide aggregate information in the context of multiple versions/variants of a given process?
- The problem of *inter-organizational interface agreements*: How to align workflow processes of different organizations while keeping local autonomy?
- The problem of *customizing*: How to measure the differences and commonalities of different processes (i.e., delta analysis)?

The goal of this journey has been the integration of previous work in this area and to demonstrate that a diverse set of problems can be addressed using the principle of inheritance.

Our tool Woflan [54] supports the four notions of inheritance used in this paper. Given two workflow processes, Woflan can decide whether one process is a subclass of another process. These processes may be created using the BPR tool Protos [46] or workflow mangement systems such as Staffware [53] and COSA [52]. Clearly, this does not provide a complete solution for the four problems presented in this paper. However, the fact that Woflan supports our inheritance notions demonstrates that tool support is possible. In the future, we hope to integrate our ideas into commercial run-time systems (i.e., workflow management systems, enterprise resource planning systems, and e-business systems).

References

1. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
2. W.M.P. van der Aalst. Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*, 41(10):639–650, 1999.
3. W.M.P. van der Aalst and K. Anyanwu. Inheritance of Interorganizational Workflows to Enable Business-to-Business E-commerce. In A. Dognac, E. van Heck, T. Saarinnen, and et. al., editors, *Proceedings of the Second International Conference on Telecommunications and Electronic Commerce (ICTEC'99)*, pages 141–157, Nashville, Tennessee, October 1999.
4. W.M.P. van der Aalst and T. Basten. Life-cycle Inheritance: A Petri-net-based Approach. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 62–81. Springer-Verlag, Berlin, 1997.
5. W.M.P. van der Aalst and T. Basten. Identifying Commonalities and Differences in Object Life Cycles using Behavioral Inheritance. In J.M. Colom and M. Koutny, editors, *Application and Theory of Petri Nets 2001*, volume 2075 of *Lecture Notes in Computer Science*, pages 32–52. Springer-Verlag, Berlin, 2001.
6. W.M.P. van der Aalst and T. Basten. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoretical Computer Science*, 270(1-2):125–203, 2002.
7. W.M.P. van der Aalst, T. Basten, H.M.W. Verbeek, P.A.C. Verkoulen, and M. Voorhoeve. Adaptive Workflow: On the Interplay between Flexibility and Support. In J. Filipe, editor, *Enterprise Information Systems*, pages 63–70. Kluwer Academic Publishers, Norwell, 2000.
8. W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors. *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2000.
9. W.M.P. van der Aalst, G. De Michelis, and C.A. Ellis, editors. *Proceedings of Workflow Management: Net-based Concepts, Models, Techniques and Tools (WFM'98)*, Lisbon, Portugal, June 1998. UNINOVA, Lisbon.
10. W.M.P. van der Aalst and M. Weske. The P2P approach to Interorganizational Workflows. In K.R. Dittrich, A. Geppert, and M.C. Norrie, editors, *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01)*, volume 2068 of *Lecture Notes in Computer Science*, pages 140–156. Springer-Verlag, Berlin, 2001.
11. N.R. Adam, V. Atluri, and W. Huang. Modeling and Analysis of Workflows using Petri Nets. *Journal of Intelligent Information Systems*, 10(2):131–158, 1998.
12. A. Agostini and G. De Michelis. Improving Flexibility of Workflow Management Systems. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 218–234. Springer-Verlag, Berlin, 2000.
13. T. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, December 1998.
14. T. Basten and W.M.P. van der Aalst. Inheritance of Behavior. *Journal of Logic and Algebraic Programming*, 47(2):47–145, 2001.
15. R.W.H. Bons, R.M. Lee, and R.W. Wagenaar. Designing trustworthy interorganizational trade procedures for open electronic commerce. *International Journal of Electronic Commerce*, 2(3):61–83, 1998.
16. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. *Data and Knowledge Engineering*, 24(3):211–238, 1998.
17. J. Dehnert. Four Steps Towards Sound Process Models. (See this volume).
18. C. Ellis and K. Keddara. ML-DEWS: Modeling Language to Support Dynamic Evolution within Workflow Systems. *Computer Supported Cooperative Work*, 9(3/4):293–333, 2000.

19. C.A. Ellis. Information Control Nets: A Mathematical Model of Office Information Flow. In *Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems*, pages 225–240, Boulder, Colorado, 1979. ACM Press.
20. C.A. Ellis and K. Keddara. A Workflow Change Is a Workflow. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 201–217. Springer-Verlag, Berlin, 2000.
21. C.A. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In N. Comstock, C. Ellis, R. Kling, J. Mylopoulos, and S. Kaplan, editors, *Proceedings of the Conference on Organizational Computing Systems*, pages 10 – 21, Milpitas, California, August 1995. ACM SIGOIS, ACM Press, New York.
22. C.A. Ellis and G.J. Nutt. Modelling and Enactment of Workflow Systems. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, Berlin, 1993.
23. R. Eshuis and R. Wieringa. Comparing Petri Net and Activity Diagram Variants for Workflow Modeling: A Quest for Reactive Petri Nets. (See this volume).
24. L. Fischer, editor. *Workflow Handbook 2001, Workflow Management Coalition*. Future Strategies, Lighthouse Point, Florida, 2001.
25. P. Grefen, K. Aberer, Y. Hoffner, and H. Ludwig. CrossFlow: Cross-organizational Workflow Management in Dynamic Virtual Enterprises. *International Journal of Computer Systems, Science, and Engineering*, 15(5):277–290, 2001.
26. P. Grefen and S. Angelov. Three-Level Process Specification for Dynamic Service Outsourcing: From Petri Nets to ebXML and WFPDL. (See this volume).
27. V. Guth and A. Oberweis. Delta-Analysis of Petri Net based Models for Business Processes. In E. Kovacs, Z. Kovacs, B. Cserto, and L. Pepei, editors, *Proceedings of the 3rd International Conference on Applied Informatics*, pages 23–32, 1997.
28. Y. Han and A. Sheth. On Adaptive Workflow Modeling. In *Proceedings of the 4th International Conference on Information Systems Analysis and Synthesis*, pages 108–116, Orlando, Florida, July 1998.
29. R. Heckel. Open Petri Nets as Semantic Model for Workflow Integration. (See this volume).
30. InConcert. *InConcert Process Designer's Guide*. InConcert Inc., Cambridge, Massachusetts, 1997.
31. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.
32. R. Kalakota and A.B. Whinston. *Frontiers of Electronic Commerce*. Addison-Wesley, Reading, Massachusetts, 1996.
33. K. Keddara. *Dynamic Evolution of Workflow Systems*. PhD thesis, University of Colorado, Boulder, Colorado, USA, 1999.
34. G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation*. Addison-Wesley, Reading MA, 1998.
35. E. Kindler, A. Martens, and W. Reisig. Inter-Operability of Workflow Applications: Local Criteria for Global Soundness. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 235–253. Springer-Verlag, Berlin, 2000.
36. M. Klein, C. Dellarocas, and A. Bernstein, editors. *Proceedings of the CSCW-98 Workshop Towards Adaptive Workflow Systems*, Seattle, Washington, November 1998.
37. M. Klein, C. Dellarocas, and A. Bernstein, editors. *Adaptive Workflow Systems*, Special Issue of Computer Supported Cooperative Work, 2000.
38. C. Lakos. Composing Abstractions of Coloured Petri Nets. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 323–345. Springer-Verlag, Berlin, 2000.

39. A. Lazcano, G. Alonso, H. Schuldt, and C. Schuler. The WISE Approach to Electronic Commerce. *International Journal of Computer Systems, Science, and Engineering*, 15(5):345–357, 2001.
40. R.M. Lee. Distributed Electronic Trade Scenarios: Representation, Design, Prototyping. *International Journal of Electronic Commerce*, 3(2):105–120, 1999.
41. R.M. Lee and R.W.H. Bons. Soft-Coded Trade Procedures for Open-edi. *International Journal of Electronic Commerce*, 1(1):27–49, 1996.
42. K. Lenz and A. Oberweis. Inter-Organizational Business Process Management With XML Nets. (See this volume).
43. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.
44. M. Merz, B. Liberman, and W. Lamersdorf. Using Mobile Agents to Support Interorganizational Workflow-Management. *International Journal on Applied Artificial Intelligence*, 11(6):551–572, 1997.
45. M. Merz, B. Liberman, and W. Lamersdorf. Crossing Organisational Boundaries with Mobile Agents in Electronic Service Markets. *Integrated Computer-Aided Engineering*, 6(2):91–104, 1999.
46. Pallas Athena. *Protos User Manual*. Pallas Athena BV, Plasmolen, The Netherlands, 1999.
47. M. Reichert and P. Dadam. ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
48. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
49. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets II: Applications*, volume 1492 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
50. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison Wesley, Reading, MA, USA, 1998.
51. A.W. Scheer. *Business Process Engineering, Reference Models for Industrial Enterprises*. Springer-Verlag, Berlin, 1994.
52. Software-Ley. *COSA User Manual*. Software-Ley GmbH, Pullheim, Germany, 1998.
53. Staffware. *Staffware 2000 / GWD User Manual*. Staffware plc, Berkshire, United Kingdom, 1999.
54. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
55. M. Voorhoeve and W.M.P. van der Aalst. Conservative Adaption of Workflow. In M. Wolf and U. Reimer, editors, *Proceedings of the International Conference on Practical Aspects of Knowledge Management (PAKM'96), Workshop on Adaptive Workflow*, pages 1–15, Basel, Switzerland, October 1996.
56. M. Voorhoeve and W.M.P. van der Aalst. Ad-hoc Workflow: Problems and Solutions. In R. Wagner, editor, *Proceedings of the 8th DEXA International Workshop on Database and Expert Systems Applications*, pages 36–40, Toulouse, France, September 1997. IEEE Computer Society Press, Los Alamitos, California, 1997.
57. H. Wehrheim. Behavioural Subtyping and Property Preservation. In S. Smith and C. Talcott, editors, *Formal Methods for Open Object-Based Distributed Systems (FMOODS 2000)*, pages 213–232. Kluwer, 2000.
58. H. Wehrheim. Subtyping Patterns for Active Objects. In H. Giese and S. Philippi, editors, *Proceedings 8ter Workshop des GI Arbeitskreises GROOM (Grundlagen objekt-orientierter Modellierung)*, volume 24/00, Münster, Germany, 2000. University of Münster.
59. V. Zwass. Electronic commerce: structures and issues . *International Journal of Electronic Commerce*, 1(1):3–23, 1996.