Specialisation in business process modelling

Frank, Ulrich

In: ICB Research Reports - Forschungsberichte des ICB / 2012

DOI: https://doi.org/10.17185/duepublico/47062

URN: urn:nbn:de:hbz:464-20180917-153329-9

Link: https://duepublico.uni-duisburg-essen.de/servlets/DocumentServlet?id=47062

Source: ICB-Research Report No. 51, May 2012

ICB
Institut für Informatik und Wirtschaftsinformatik

Ulrich Frank

Specialisation in Business Process Modelling: Motivation, Approaches and Limitations

ICB-RESEARCH REPORT

UNIVERSITÄT DUISBURG ESSEN

ICB-Research Report No. 51
May 2012

Die Forschungsberichte des Instituts für Informatik und Wirtschaftsinformatik dienen der Darstellung vorläufiger Ergebnisse, die i. d. R. noch für spätere Veröffentlichungen überarbeitet werden. Die Autoren sind deshalb für kritische Hinweise dankbar.

The ICB Research Reports comprise preliminary results which will usually be revised for subsequent publications. Critical comments would be appreciated by the authors.

Author's Address:

Ulrich Frank

Institut für Informatik und
Wirtschaftsinformatik (ICB)
Universität Duisburg-Essen
Universitätsstr. 9
D-45141 Essen

ulrich.frank@uni-due.de

Contact:

Institut für Informatik und
Wirtschaftsinformatik (ICB)
Universität Duisburg-Essen
Universitätsstr. 9
45141 Essen

Tel.: 0201-183-4041
Fax: 0201-183-4011
Email: icb@uni-duisburg-essen.de

ISSN 1860-2770 (Print)
ISSN 1866-5101 (Online)

Abstract

To facilitate designing and maintaining collections of business process models and corresponding implementation documents, it is of pivotal relevance to identify commonalities that a set of process types share. They allow for reusing parts of process models and are a prerequisite of efficient and consistent modifications. Discovering or constructing commonalities requires abstraction. With respect to model integrity, there is need for precisely specified abstraction concepts that enable convenient and secure adaptations to particular requirements. Against the background of various alternative approaches to foster abstraction in process modelling, this report is mainly aimed at investigating chances and specific challenges that relate to the conception of generalisation/specialisation for process types. For this purpose, a preliminary conception of process specialisation is proposed that is based on specialisation of static artefacts. A subsequent overview of existing approaches to specify a concept of process specialisation shows that none of these is satisfactory. Moreover, it will be shown that a conception of process specialisation that corresponds to specialisation of static artefacts is not possible. Finally, an outline of a relaxed conception of specialisation and the use of local meta process types are proposed as a possible loophole.

Table of Contents

Figures

1 Introduction

Abstraction is of outstanding importance for dealing with complexity – in everyday life and especially in conceptual modelling. It allows us to fade out those aspects of a subject that are not relevant for a certain purpose. It also enables us to take advantage of commonalities shared by a set of concepts: Common concepts are a prerequisite for reuse and integration. At the same time, abstraction fosters system maintenance. On the one hand abstracting on commonalities enables modifications that are applied only once – to common representations – and update consistently a range of corresponding components. On the other hand, modifications of those system parts that are not included in the common representations – that are abstracted from – can be modified without causing harmful side effects or in other words: without challenging the validity of the common concepts. The use of abstraction in conceptual modelling requires precisely defined abstraction concepts, which includes mechanisms that allow for convenient and secure adaptations. The most important abstraction concepts in conceptual modelling and systems design comprise classes, encapsulation, polymorphism and – last but not least – generalisation/specialisation. A class allows abstracting from particular instances. As a consequence, the features defined for a class, especially its methods, can be reused by all its instances. On a conceptual level, changes that are applied to a class are immediately effective in all corresponding instances. Of course, the effect of changing classes in a system depends on the type of change and the particular programming language. Encapsulation, also referred to as information hiding, is an abstraction over all possible implementation that satisfy a certain interface. Hence, it allows for abstracting from the implementation of a method. As a consequence, the implementation can be changed without affecting the interface, thereby reducing the range of side effects substantially. Generalisation results in a general concept that represents the common features of a set of more specific concepts. Specialisation allows for adding further features to a general concept to define more specific concepts. Hence, generalisation allows abstracting from the specific features of specialized concepts. As a consequence, changes that are restricted to common features need to be applied only once, while they are effective in all specialized concepts. Also, adding further specialized concepts or modifying existing ones does not compromise the semantics of the general concept. Since generalisation implies specialisation et vice versa, we will from now on use the term “specialisation” only to represent both sides of the same coin. Polymorphism allows for abstracting from the class of the object a message is sent to. Hence, it is an abstraction over all methods of a certain kind *and* all corresponding classes of objects a corresponding message might be sent to. This allows for adding new classes to a system that offer a specific implementation of a method without the need to change the code where the method is called. If, for instance, a graphical editor is to be extended to include triangles, a corresponding class would be specialized from an existing class. Hence, polymorphism takes advantage of both ideas, specialisation and information hiding.

Abstraction concepts are widely used in conceptual modelling. Also, their benefits are, provided they are appropriately used, undisputed. However, so far, the use of abstraction concepts is widely restricted to static or functional concepts, especially in the area of object-oriented modelling and system design. Current languages for business process modelling show a remarkable lack of abstraction. This is a serious shortcoming with respect to modelling productivity and model maintenance. This report is aimed at analysing the background for this surprising insufficiency. First, we will show that there is urgent need for abstraction in process modelling. Against this background, we will discuss possible approaches to gain a higher level of abstraction. Since a concept of process specialisation that satisfies certain requirements, would be especially useful, we will then review existing approaches to process specialisation or process inheritance respectively. Based on the results of this review and principal concerns regarding the feasibility of a powerful specialisation concept, we will finally present alternative approaches to increase the level of abstraction in process modelling.

2 The Need for Abstraction in Process Modelling

The lack of abstraction concepts in business process modelling languages such as EPC (Scheer 1999) or BPMN (OMG 2011) could be contributed to the fact that users of these languages did not complain about this insufficiency. While the latter may be true, there is nevertheless a very good reason to demand for a higher level of abstraction in process modelling. To back this claim, we will look at scenarios that describe the prototypical use of process models in larger organisations.

2.1 Motivating Example

Larger organisations may comprise a few tens of business process types. Designing and maintaining a corresponding number of business process models face substantial challenges. The example in Figure 1 shows a collection of process models in an imaginary company.

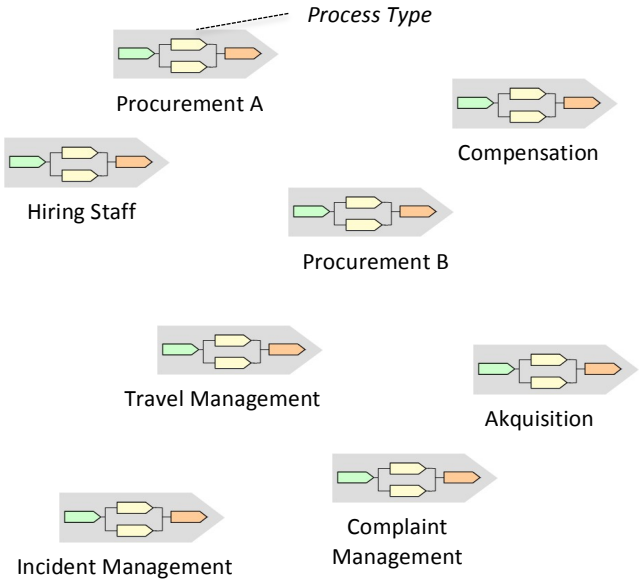

Figure 1: Illustration of Collection of Business Process Models

Scenario 1: A company is confronted with a new legal regulation that may have to be accounted for in several business process types. This scenario recommends an abstraction that represents those process features that are affected by the legal regulation. This would, at best, allow for restricting the required modifications to the corresponding commonalities. Otherwise, every process model would have to be checked and possibly modified – with respective consequences on integrity and costs.

Scenario 2: A further more specific order management process needs to be implemented. An abstraction that allows for identifying the commonalities of existing order management processes and additional ones would be beneficial, since it would enable reusing existing models and would facilitate model maintenance later on.

The following section gives an overview of possible approaches to address the challenges illustrated by the above scenarios. Assessing these approaches will contribute to develop requirements a sophisticated process abstraction concept should satisfy.

2.2 Possible Approaches to Process Abstraction

Representing relevant commonalities of a set of business process models can be accomplished on different levels of granularity, i.e. one could focus on common features of business process types, or of common features of sub-processes or events. We start with focusing on abstractions that concern entire business process models. For this purpose, we distinguish four different approaches: an informal conception of similarity, an informal conception of specialisation, aspects, and a formal conception of specialisation.

If two business process types share commonalities, they are similar. A weak form of representing similarities between two process models would be to introduce a special kind of association. It would allow for expressing that two process models are similar (see Figure 2).

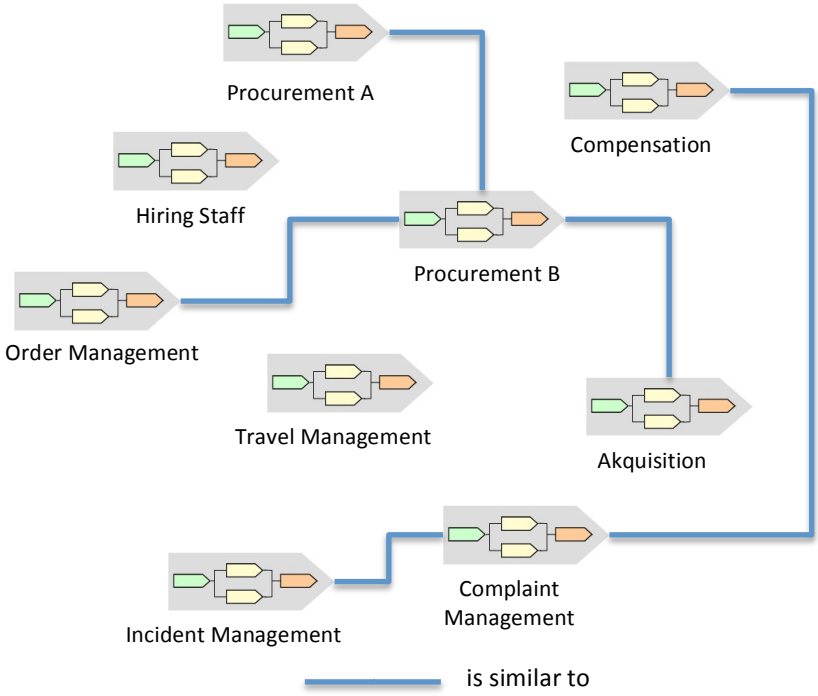

Figure 2: Illustration of informal Concept of Process Similarity

While this approach is easy to implement, it would be of limited help only. Since it does not include a formal specification of similarity, it would remain unclear, what the commonalities are, the idea of similarity is based on. However, it would be possible to add an informal description of the intended similarity. Since there may be various kinds of similarities between two process models, one could refine the simple concept of process similarity by allowing for expressing different kinds of – again informal – similarity. Figure 3 illustrates the use of an informal conception of different kinds of similarity.

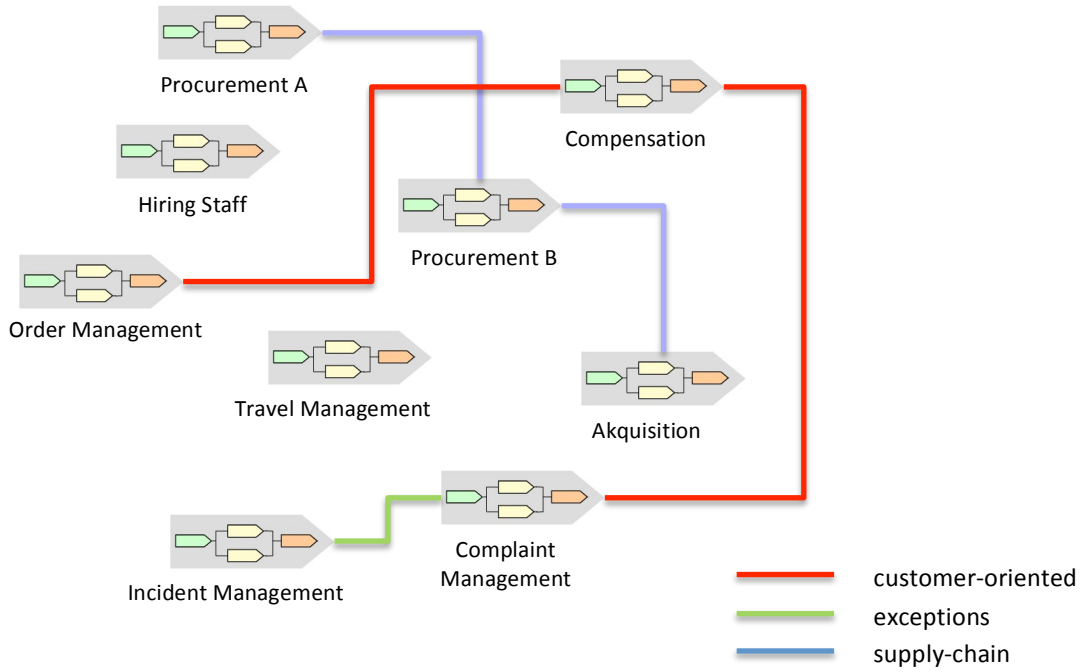

Figure 3: Different kinds of Similarity

Allowing for differentiating kinds of similarity increases the chance to identify business process models that share common characteristics of a certain kind. While it is fairly easy to implement, its contribution to fostering reuse and maintenance is still rather limited, since it lacks a formal specification of the commonalities the different kinds of similarity are based on. Nevertheless it may help with identifying process models that *may* share common parts or that *may* be affected by a certain new requirement.

In Software Engineering, there are various approaches to reduce effort and risk of maintaining large systems. With respect to the above scenarios, the concept of an "aspect" (Kiczales, Lamping et al. 1997; Kiczales, Hugunin et al. 2003) is of particular relevance. It serves as an abstraction in those cases where abstraction concepts provided by object-oriented programming language are not suited to express certain commonalities and/or as an instrument to reengineering code by adding additional abstractions to existing code in order to facilitate maintenance. Parts of a software system that address the same or similar tasks are represented as "cross-cutting concerns". Examples of cross-cutting concerns are printing, security, persistence, user-interface etc. A cross-cutting concern is assigned to methods within the system that serve to handle the respective tasks. An aspect serves as an abstraction over these methods, which are usually part of different classes. If new requirements have an effect on e.g. security issues, the corresponding aspect would allow for writing code that is applied to the distributed methods in a predefined way. Figure 4 gives an example of how aspects can be conceptualized. In this case, all methods of respective classes that serve to update a graphical representation at the display are subsumed into a corresponding aspect "DisplayUpdating".

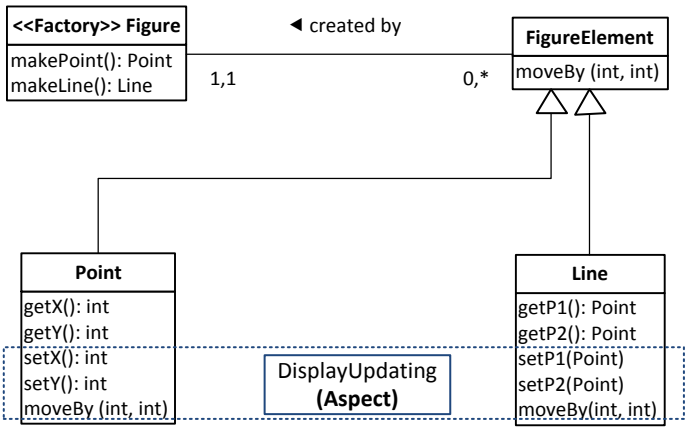

Figure 4: Conceptual Representation of Example Aspect, inspired by a corresponding code example in (Kiczales, Hugunin et al. 2003)

As a consequence, an aspect allows for addressing all particular methods it comprises in one statement, hence contributing to code reuse and especially to efficient and secure maintenance. The example in Figure 5 illustrates the use of aspects on the code level, based on an extension of Java. The language concept "pointcut" serves to define the abstraction by subsuming all respective methods. Then the pointcut can be used to write code that applies to all methods it comprises.

```
pointcut moves ():
        receptions (void FigureElement.moveBy (int, int)) ||
        receptions (void Point.setX (int))                 ||
        receptions (void Point.setY (int))                 ||
        receptions (void Line.setP1 (Point))               ||
        receptions (void Line.setP2 (Point));
```

specification of aspect → reference to corresponding methods

```
aspect DisplayUpdating {
        pointcut moves ():
        after (): moves () {
            Display.needsRepaint ();
        }
}
```

supplementing all respectively marked methods with additional code

Figure 5: Implementation of Aspects as an Extension to Java, adapted from (Kiczales, Hugunin et al. 2003)

While the abstraction that aspects allow for can be of remarkable value, it is important to note that there is no clear semantics of aspect. Instead, it is left to the software engineer to define the semantics through the specification of concerns and the association ("weaving") with corresponding code. With respect to business process models, the idea of aspects can be used to define concerns that are associated with business process types. Figure 6 illustrates the application of concerns to business process modelling.

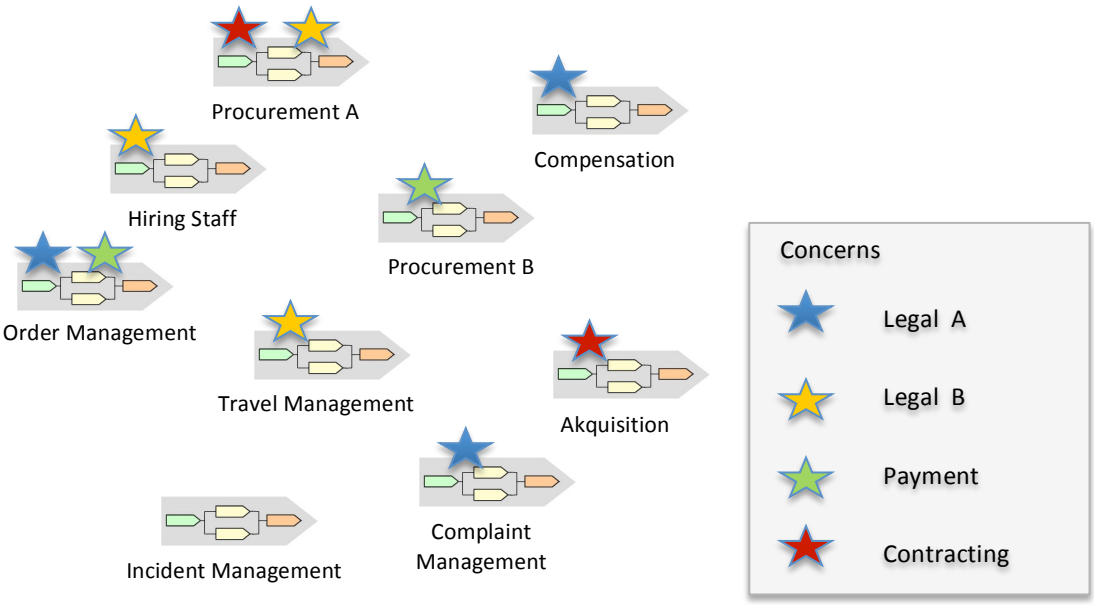

Figure 6: Assigning Concerns to Business Process Types

Defining concerns may help with identifying process models that may reuse common parts or that may be affected by new requirements. However, as long as there is no precise specification of concerns and how corresponding modifications should affect the respective process models, the concept remains dissatisfactory.

Compared to the previous approaches specialisation is promising the advantage that it seems to come with a clear meaning: As a default, specializing a concept means that all propositions that hold true about the general concept hold true for the specialized concept, too. Nevertheless, as we shall see, the specification of a formal concept of process specialisation is far from trivial. As a modest alternative, one could introduce an informal concept of specialisation. This would allow for expressing that a process type is regarded as a specialisation of a more general process type – hence indicating that change applied to the more general type would somehow affect the specialized types. However, updating the specialized process models could not be automated. Therefore, it would be most desirable to have a formal concept of process specialisation. Figure 7 illustrates that a change in a general process type – indicated by the blue box – would be propagated to the respective subtypes in a precisely defined way.

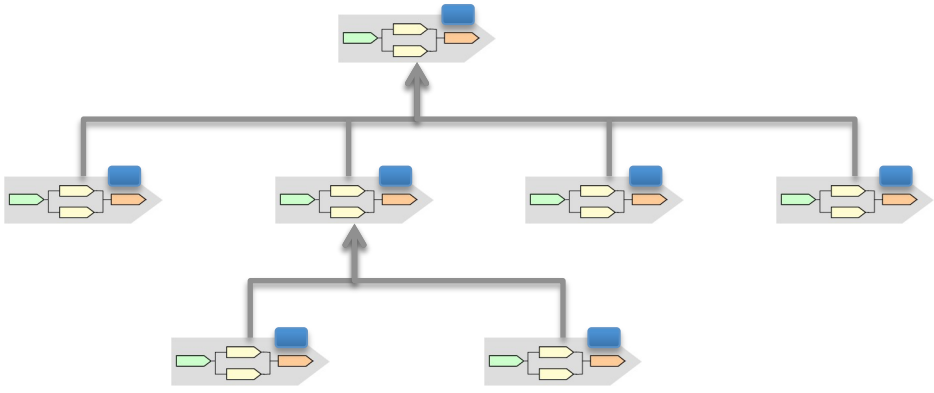

Figure 7: Illustration of Process Specialisation

A formal concept of process specialisation would be very beneficial for both use scenarios. It would allow for conveniently updating all business process models that are specialized from a general process models that captures the essential the new legal regulations relate to. Also, it would allow for specializing a new order management process from an existing one.

3 Process Specialisation: Peculiarities and Requirements

Generalisation/specialisation is an established concept for creating static abstractions, such as object models[1]. Its potential benefits for process modelling are obvious, too.

3.1 A Brief General Consideration of Specialisation

To prepare for analysing the peculiarities related to defining a concept of process specialisation, we shall first look at specialisation in general. While it is a pivotal concept in human communication, its semantics varies to a remarkable degree, which is illustrated by the following examples where specialisation is indicated by the predicate "is a".

"Fear is a sensation."
"A racing bike is a bike."
"A research report is a document."
"A student is a person."
"Electronic order processing is an order processing."

The examples show that the meaning of specialisation depends on the meaning of the corresponding concepts. A concept guides structuring and interpreting the things that we perceive. Specializing a concept can be regarded as restricting the interpretation, i.e. it does not apply to as many objects as the superordinate concept. Hence, if the interpretations – and the sets of objects they apply to – are not precisely defined, the semantics of specialisation remains vague, too. That is especially the case for intentional concepts, i.e. concepts which are defined by referring to emotional states or frames of mind. The underlying common understanding can be sufficient for agreeing that a specialisation is appropriate. Most people would probably agree that fear is a special kind of sensation. However, it is certainly more difficult to tell the exact difference between the two concepts, i.e. to specify the specialisation. This is easier with concepts that allow for an extensional definition. In this case the semantics of a concept is defined by the set (extension) of its features. The concept "person" could be defined by features such as last name, first name, gender, date of birth etc. The specialized concept "Student" would include further features. As a consequence, it would be much easier to define the semantics of specialisation. But even with concepts that can be defined extensionally, a sound conception of specialisation can be challenging. Probably nobody would object that a racing bike is a special kind of bike. At first sight, both have common features: two wheels, a saddle, brakes etc. However, they have clearly different types of wheels, saddles etc. Apparently, it is hard to conceptualize the specialisation of an ordinary brake to a

[1] That does not mean, however, that using generalisation/specialisation in object-oriented modelling is without any problems. The semantics of generalization/specialisation in object-oriented programming languages is different from that of natural language or predicate logics – which is caused by different concepts of "class". This semantic mismatch is often not accounted for, which can result in inconsistent models (for a comprehensive discussion of this problem, see Frank 2003).

racing bike brake. As a consequence, it is hard to define a precise meaning for specialisation in such a case.

In order to further analyse the question how specialisation could be specified for process types, we will first define a general conception of specialisation that is motivated by the benefits intended for reuse and maintenance. With respect to extensional concepts, a conception of generalisation/specialisation can be defined by a few essential characteristics and additional constraints.

Essential Characteristics:

E1: A set of concepts that share common features can be generalized into a superordinate concept. This happens through abstraction: By abstracting to the common features which constitute the superordinate concept – and by abstracting from those features that are specific to each subordinate concept.

E2: A concept can be specialized from a superordinate concept by adding further features.

Constraints:

C1: *Subordination*: Every assertion that holds for a general concept is true for its subordinate concepts, too.[2] This request is an implication of subordination in logics (e.g. in predicate logics) – and a prerequisite for efficient maintainability: Every modification of a general concept is immediately effective in its subordinated concepts.

C2: *Substitutability*: Substitutability is an implication of subordination. However, it includes a further level of abstraction and it is directly related to software systems. Whenever an instance of a super concept (e.g. a superclass) is required, it can be replaced by an instance of one of its sub concepts (e.g. subclasses) without causing any harm. For this purpose, the instance of a sub concept has to behave in a way that makes it indistinguishable from the behaviour expected from instances of the respective super concept. Substitutability is a common (though not undisputed) request for the construction of type systems of object-oriented programming languages (see e.g. Wegner and Zdonik 1988 or Liskov and Wing 1994).

C3: *Monotonic extension*: For E2 not to hurt C1 or C2, adding new features to sub concepts must be monotonic, i.e. it must not affect the semantics of the corresponding super concept.

In the realm of software development, the term "inheritance" is often used instead of specialisation. In its less restrictive interpretation, it simply means to copy features from a subordinate concept, which then may be arbitrarily modified. Hence, the emphasis is on reuse. However, the redefinition of inherited features has severe downsides with respect to integrity. On the one hand, it compromises maintenance: It is not possible anymore to modify a set of subordinate concepts simply be changing the corresponding feature within the superordinate concept. On the other hand, it is a threat to substitutability. Therefore, inheritance is

[2] Note that this does not include propositions on a meta level, e.g. "The concept 'Human Being' is the superordinate concept of 'Student'".

mostly supplemented by rules that restrict the permissible modifications. In object-oriented programming languages, it is usually possible to override the implementation of a method. While this may cause a different behaviour and thereby violate constraint C2, it does not have an effect on the formal definition of the interface. Some programming languages as well as the UML 2.0 (OMG 2007, p. 40) allow for specific changes to the interface of a subordinate class, too. A covariant redefinition means that the classes of the parameters that are passed with a method call are replaced by their subclasses. Take, for instance, the two classes "Laser_Printer" and "Colour_Laser_Printer", with the latter specified as a subclass of the first. "Laser_Printer" includes a method "getLevel (c: Cartridge)". "Colour Laser Printer" redefines the interfaces covariantly into "getLevel (c: Colour_Cartridge)". While covariant redefinition is suggested by some as an approach that corresponds to a common pattern in natural language – and hence contributes to more "natural" models (see, e.g. Ducournau 2002), it comes with a severe problem: It violates the characteristic feature b) It does not allow for replacing an instance of a class by an instance of a corresponding subclass without avoiding problems. From a logical perspective, it produces a contradiction: "Laser_Printer requires Cartridge" implies "Colour_Laser_Printer requires Cartridge". However, this proposition is redefined to "Colour_Laser_Printer requires Colour_Cartridge". Since there are cartridges that are not colour cartridges, the redefinition contradicts the logical implication. This is the same effect that is caused by so called non-monotonic extensions of a knowledge base: It may seem natural, but it violates traditional logics. With respect to programming covariant redefinitions – if they are accepted by the compiler – can cause run-time errors with write accesses: If, e.g., a parameter of the class "Cartridge" is passed to an object of the class "Colour_Laser_Printer" that acts as a substitute for an object of superclass "Laser_Printer", the interface of the subclass would be hurt (for a comprehensive discussion see Meyer 1997). A contravariant redefinition – which is usually not supported, because it seems contra-intuitive – would replace the parameter classes by corresponding superclasses.

Inheritance in conjunction with redefining inherited features adds flexibility to reuse. Nevertheless, we will at first restrict our analysis to specialisation that corresponds to the above conceptualisation in order to avoid the threats to integrity created by redefinitions.

3.2 Peculiarties of Process Specialisation

Figure 8 shows a further example of a specialisation hierarchy of business process types. It is a high level model that defines specialisation relationships between business process types without representing the respective control flows. At first sight, it may seem irritating that "credit application business firm" is defined as subordinate to "credit application certified firm", since a "certified firm" is a subclass of "business firm". However, while a more specialized subject may correspond to a more specialized process type, it does not have to. The model shown in Figure 8 may have resulted from requirements analysis, where domain ex-

perts declared the specialized process types to be "special cases" of the respective super types.

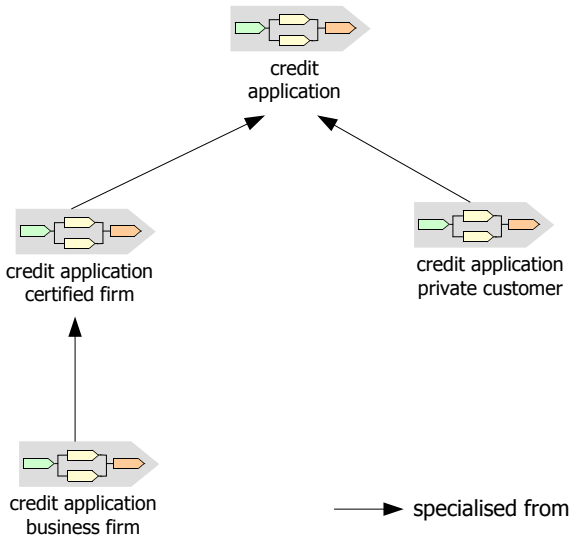

Figure 8: Specialisation relationships between business process types

To develop an idea of process specialisation that promotes reuse and efficient maintenance, we need to apply the conception of specialisation defined above. This requires defining a process type by its features. According to specialisation for static abstractions, one could try a conception of business process specialisation that is based on adding further features. With respect to a dynamic abstraction, the features of a business process type can be regarded – among other concepts – as the event and process types its control flow is composed of. Figure 9 shows a corresponding conception applied to the decomposed business process models in Figure 8 – except for "credit application private customer".

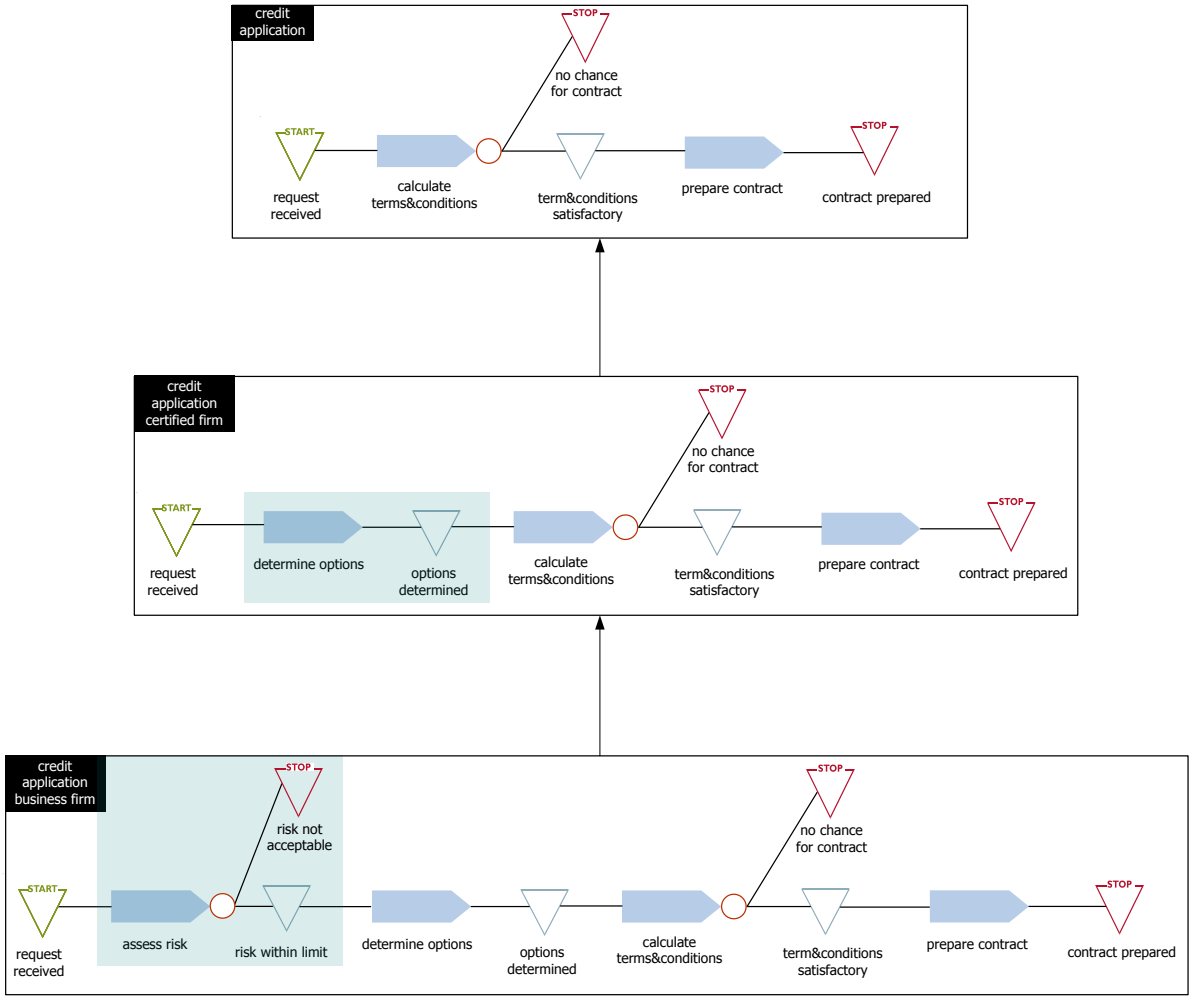

Figure 9: Illustration of Process Specialisation by Adding Parts

The generic business process model is specialized into "credit application certified firm" by adding further elements to the control flow. This applies to the specialisation of "credit application certified firm", too. At first sight, the example in Figure 9 may suggest that the shown extension of a process type can be used as a foundation for a corresponding specialisation concept. However, this would be a deceptive impression. On the one hand, such a conception might be regarded as inappropriate with respect to existing ideas of process specialisation. Sometimes, a further process type where process steps have been eliminated may be regarded as the "special case" (see example in Figure 10). While this may be regarded as a result of the ambiguous use of the term "specialisation" in natural language that has to be overcome with the specification of a modelling language, it cannot entirely be ignored since usability demands for modelling concepts that corresponds to the technical terminology prospective users are familiar with (Frank 2011).

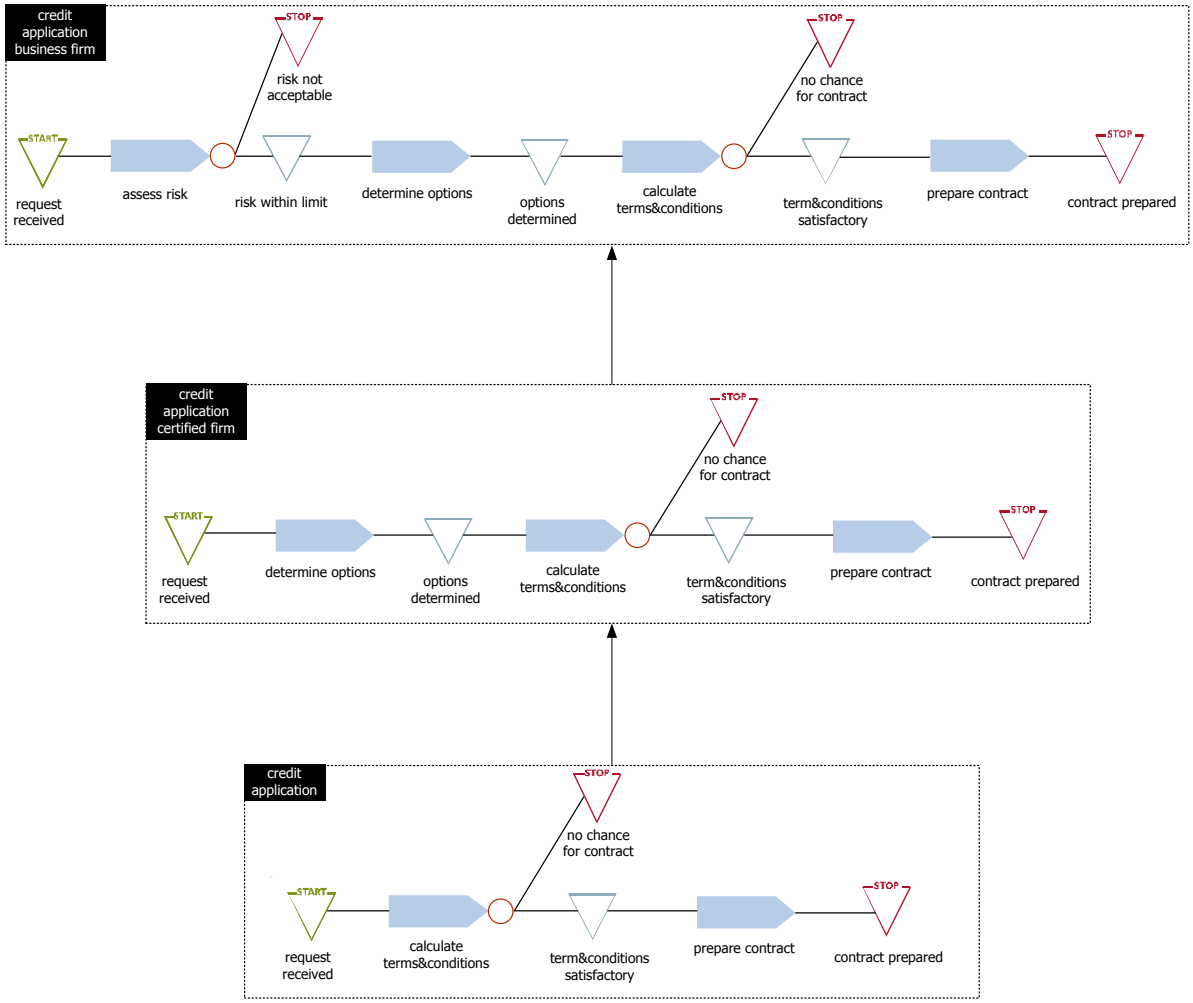

Figure 10: Illustration of Process Specialisation by Eliminating Parts

On the other hand – and more important – adding features, i.e. subprocesses and events, in a way that the constraints defined for specialisation are not violated, is apparently not as easy as with static abstractions. At first sight, it is obvious that the extensions shown in Figure 9 do not satisfy the substitutability constraint: Substituting an instance of a subprocess for an instance of the corresponding super process will certainly not remain unnoticed. Our brief discussion shows that developing a conception generalizing/specializing business process types faces serious obstacles.

4 Existing Approaches to Process Specialisation

Compared to the plethora of literature on business process modelling, relatively little work has been published on generalisation/specialisation.

4.1 Specialisation of Behaviour in Object-Oriented Software Systems

The majority of related work is focussing on specialisation or inheritance respectively in object-oriented programming languages – with special emphasis on object behaviour. Hence, these publications are aimed at dynamic abstractions of classes or objects respectively. Their main focus is on the redefinition of inherited operations: What are acceptable rules for the possible scope of redefining – or specializing – the behaviour of these operations. For this reason, these publications are related to the specialisation of business process types. Most authors agree that the demand for substitutability needs to be fulfilled: "The objects of the subtype ought to behave the same as those of the supertype as far as anyone or any program using supertype objects can tell." (Liskov and Wing 1994), p. 1811; for a similar request see Wegner and Zdonik 1988) At the same time, it is widely agreed that prohibiting any redefinition of inherited operations is too restrictive. This constitutes the challenge to define rules for modifying inherited operations in a way that the resulting behaviour of a corresponding object does not deviate from the behaviour that is expected from an object of the superclass. While some of these works do without graphical representations of object behaviour, Schrefl and Stumptner use a kind of state charts ("behaviour diagram") to visualize their approach. They draw upon Petri Nets: "States correspond to places of Petri nets, activities to transitions." (Schrefl and Stumptner 2002, p. 95) Behaviour diagrams are used to represent object life cycles. The notion of specialisation they suggest is based on the conception of a subnet. A behaviour diagram B' is a subnet of another behaviour diagram B, if B' is "embedded" in B. For a subnet to represent an acceptable specialisation of object behaviour, it has to be a consistent extension. Schrefl and Stumptner differentiate "observation consistency" and "invocation consistency"[3]. A specialisation is an observation consistent extension of object behaviour, if the resulting behaviour is the same as the behaviour defined for the superclass. Imagine, one would view a process instance through a special lens that faded out states and activities which are not present in the lifecycle of objects of superclasses. Then it would not be possible to distinguish the behaviour of an object of the subclass from the one of an object of the superclass. Invocation consistency is not restricted to observable behaviour. It include the invocation of operations: "Thus, any operation invocable on instances of a supertype must under the same precondition also be invocable on instances of any subtype and executing the operation on instances of a subtype meets the postcondition for the operation specified at the supertype." (Schrefl and Stumptner 2002, p. 102) The authors provide a formal definition for

[3] Note that we do not follow the distinction between weak and strong "invocation consistency" suggested by the authors, because we do not need it for our purpose.

invocation consistency, which can be nicely illustrated through examples. Figure 11 shows an example of a specialisation that is invocation consistent.

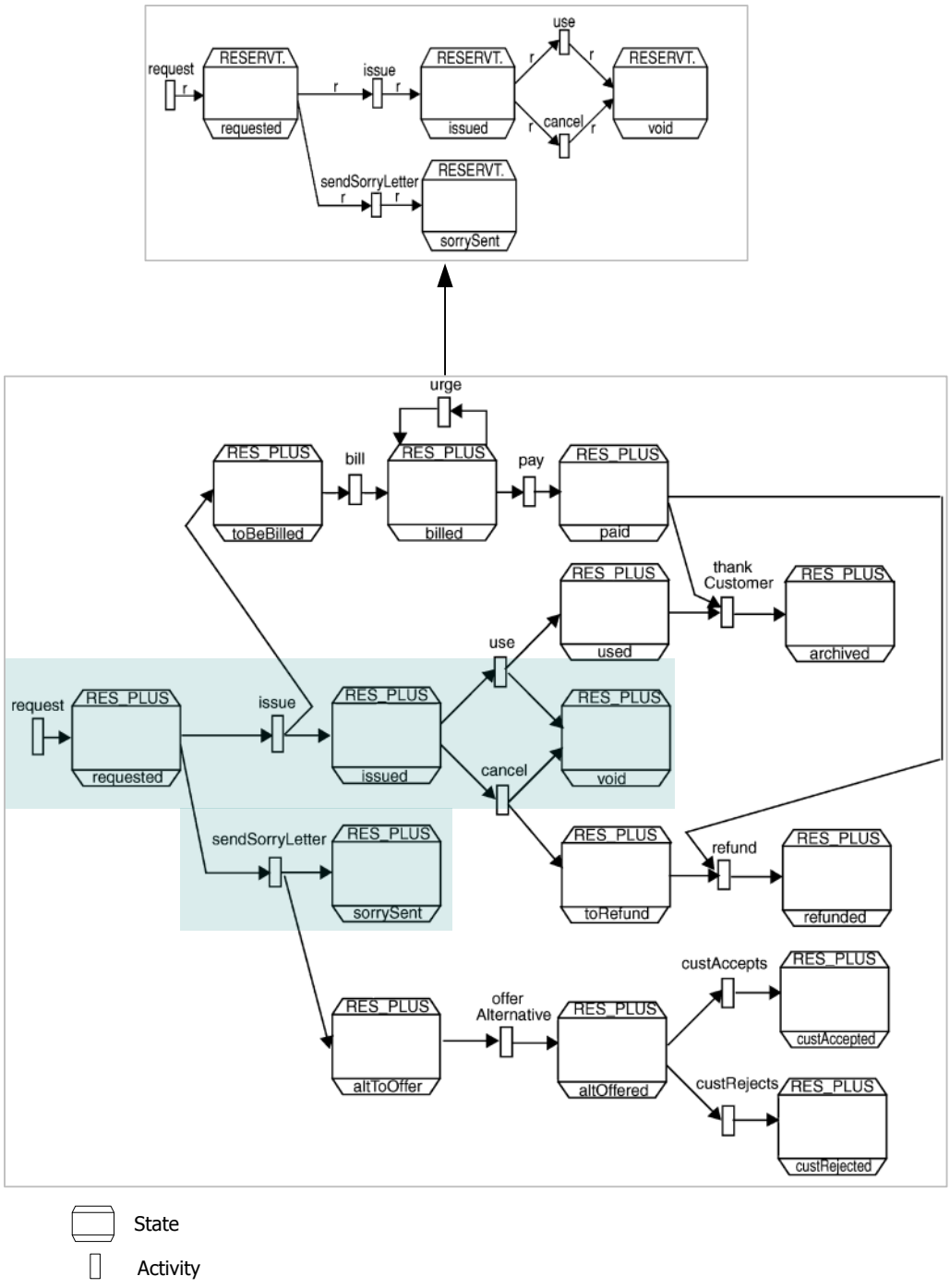

Figure 11: Example of invocation consistent specialisation, constructed from examples in (Schrefl and Stumptner 2002)

The net that represents the object lifecycle of the superclass is contained in the net representing the object lifecycle of a corresponding subclass. A state can trigger one or a set of mutually exclusive activities (exclusive choice, XOR). An activity can start only after all states that directly precede it, have occurred. An activity (also referred to as "event") can result in one or more states (AND). An object of the specialized class depicted in Figure 11 has the same

behaviour as objects of the superclass. Apparently, the additional activities and states do not affect the inherited lifecycle – which indicates observation consistency. In addition to that, every operation that can be invoked on objects of the superclass, e.g. "issue", "use" etc., can be invoked on instances of the subclass, too – provided the same conditions are satisfied. The Figure 12 shows a case where specialisation does not satisfy invocation consistency:

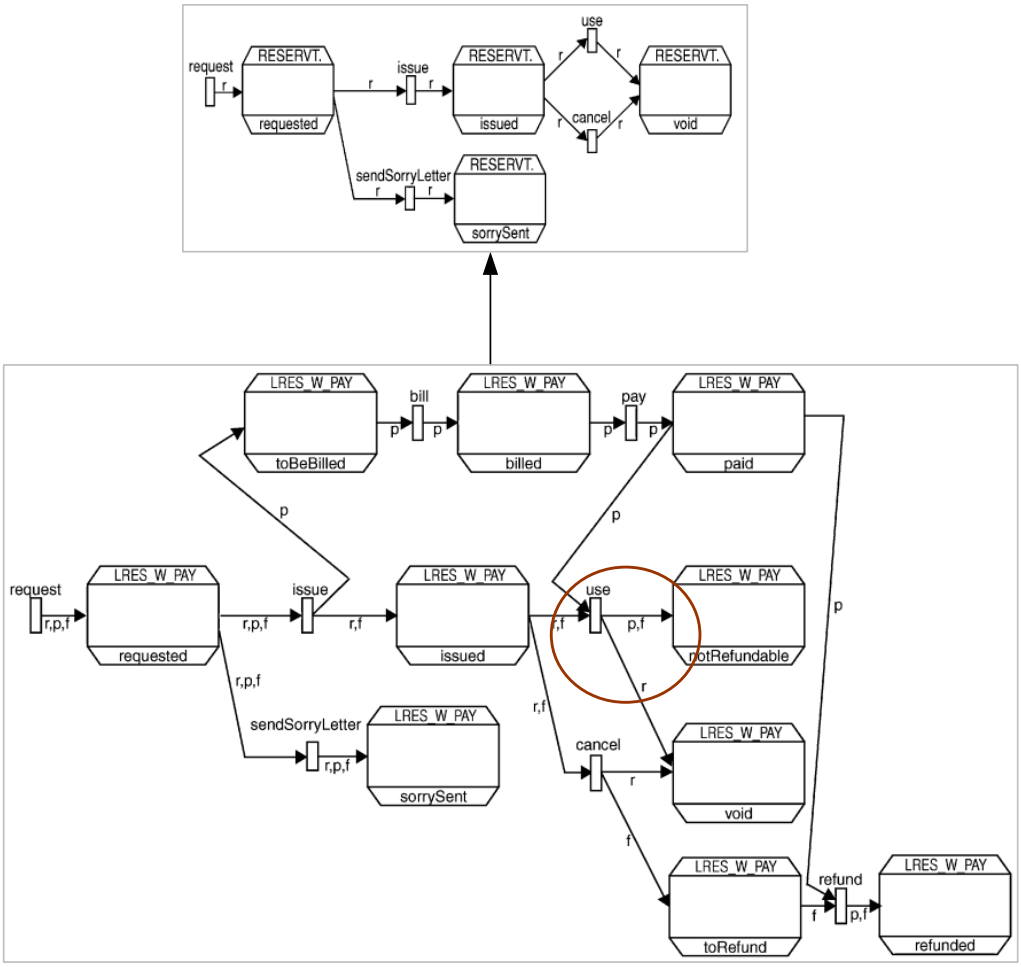

Figure 12: Example specialisation that is not invocation consistent – constructed from examples in (Schrefl and Stumptner 2002)

For objects of the superclass, the activity "use" will be performed immediately after "reservation issued" has occurred. This is different for objects of the subclass: There this change of state can be performed only after the reservation has been paid for.

Could such a conception of specializing object behaviour serve as a model for a specialisation concept of business process types? Unfortunately, the answer is no – even though the authors indicate that it could be used for workflow types. This is for a simple reason: The conception of substitutability applied by Schrefl and Stumptner (and others in the area of object-oriented software engineering) is not appropriate for business process models. Their perspective on object behaviour is characterized by the idea of a contract that defines possible states and state changes. An object satisfies such a contract, if it allows all paths of possible state changes to be executed. Hence, a specialisation of object behaviour is regarded as con-

sistent, if it fulfils the contract defined for the superclass. It is essential for assessing this conception that a subclass is assumed to fulfil the contract, too, if it allows for additional paths of state changes that do not interfere with those of the superclass. Hence, it is assumed that additional behaviour – to whatever extent it may occur – does not jeopardize substitutability. While this conception may be satisfactory with respect to building reliable software systems – which still has to be shown, it is not acceptable for modelling business processes. A business process is characterized by the consumption of scarce resources. If a business process does not only include those processes and consumes those resources that are required for doing the job, but includes further processes, consumes further resources, it would certainly not pass as an acceptable substitute. Instead, it would be regarded as a strange imposition. Furthermore, but not as important, behaviour is restricted to the lifecycle of objects of one class. A business process model is not conceptualized as the behaviour of objects of one particular class. This is the case for most workflow models, too.

What are the lessons to be learned from the work on specialisation of object behaviour for the definition of specialisation for business process models? The idea that business process should fulfil a contract deserves attention. Contracts are an important instrument to reduce complexity – certainly not only for constructing software systems. In the area of business process models, contracts seem to be relevant for two specific reasons. Firstly, the concept of a contract is well known in business. Secondly, it has gained additional weight through the popularity of service-orientation. A further aspect of contracts that is accounted for in software engineering is related to specializing the parameters required for or delivered by an operation. This may be a useful approach with respect to resources or organizational units required to execute a business process. If, e.g., a process requires a programmer, a computer scientist could be used as a substitute. This could be expressed explicitly by defining computer scientist as a specialisation of programmer. Beyond providing inspiration, the work on specializing object behaviour suggests that the idea of specializing process types by extending them is not applicable to business process types.

4.2 Workflow Inheritance

One further approach is aimed at specialisation in the realm of workflow types, which are similar to business process types. In a series of publications, van der Aalst and various coauthors analyse the subject of workflow inheritance and propose a few conceptions of specialisation (e.g. Aalst and Basten 1999a, Aalst and Basten 1999b, Aalst and Basten 2002a, Aalst and Basten 2003, Aalst and Basten 2002b). Van der Aalst et al. motivate their approach with pragmatic reasons. They discuss various kinds of change (e.g. “dynamic change”, “customzing business processes”, see, e.g. Aalst and Basten 1999b) that suggest the use of appropriate specialisation concepts. They model workflow types with Petri nets. In general, workflow inheritance is characterized by extending an inherited net through further transitions (tasks) and places (conditions). However, not any extension qualifies for specialisation. Van der Aalst and Basten claim that an instance x of a process type that is of a specialized from

the process type y "… can do what y can do with respect to the tasks present in y" (Aalst and Basten 2003, p. 96). Against this background, they propose four concepts of inheritance with varying degrees of semantic restriction (Aalst and Basten 2003, p. 96 ff.). The term "inheritance" is misleading because they actually mean rules for different kinds of specialisation. For this purpose, they distinguish "blocking" and "hiding" of tasks. A task that was added on the level of a specialized process type can be hidden, i.e. it can be abstracted from that task in a way as if it would not exist. Blocking a task on the other hand means that its execution is blocked. "Protocol inheritance" implies that an instance of a specialized process type x shows the same behaviour as an instance of its superordinate process type y, if additional tasks are blocked. In other words: in this case, they cannot be distinguished. If x seems to be identical with y, if certain tasks in x are hidden, they speak of "projection inheritance". "Protocol/projection inheritance" defines the most restrictive form of specialisation. It requires that x cannot be distinguished from y both if the additional tasks were hidden and if they were blocked. "Life-cycle inheritance" on the other side is least restrictive. It is the case, if a specialized process type conforms either to protocol or to projection inheritance (Aalst and Basten 1997, p. 70). For each of these four types of specialisation, they define transformation rules that preserve the characteristics required for an instance of a subtype. Figure 13 illustrates the concepts with a few examples. All of the examples show a specialized workflow type that is extended by the task "check". A transition (task) that is filled with a red rectangle displaying an "X" indicates that it is blocked in order to satisfy a specialisation constraint. A transparent grey rectangle indicates that the part of the workflow it covers is assumed not to be executed. Note that the letters used to identify the workflow types are those defined by the authors. In Figure 13 they are presented in a different order. The instances of workflow type C execute as if they were instances of A, if the transition (task) "check" is blocked. If it was hidden only, it would be possible that the transition "handle" does not fire (place p1 can result in firing "check" OR "handle"). This would violate the specialisation constraint. If the shaded part of workflow type D is not executed, a corresponding instance would produce the same behaviour as an instance of A. On the other hand, blocking "check" in this case would result in a deadlock because the synchronisation could not be performed. In workflow type B, the transition "check" can either be hidden or blocked. In both cases a corresponding instance would execute as if it was an instance of A.

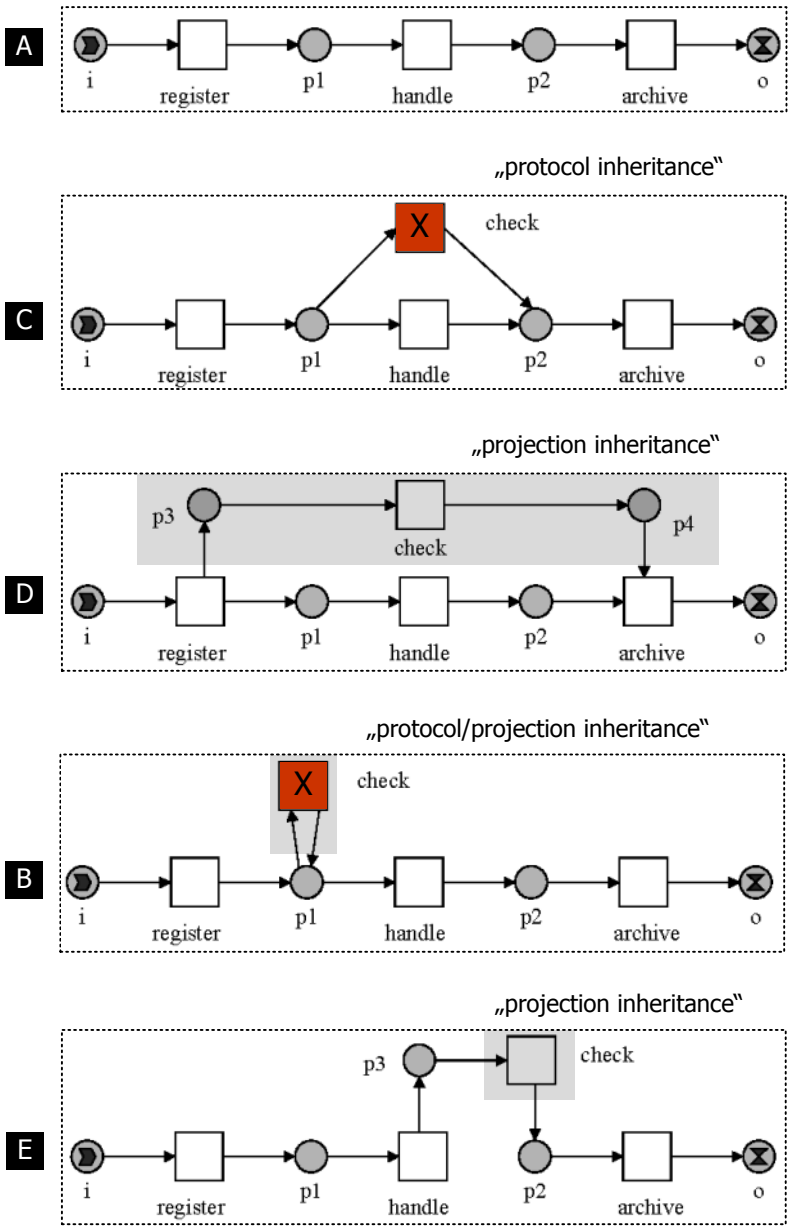

Figure 13: Illustrations of specialisation concepts suggested by van der Aalst and Basten (adapted from Aalst and Basten 2003, p. 97)

Do the concepts of specialisation proposed by van der Aalst et al. provide a suitable foundation for specializing business process types? While the various kinds of specialisation are defined with respect to suitability, they do not satisfy this demand with respect to business processes: Adding a further task (or in the terminology we use for MEMO business process models: a further process) will result in a process type that requires additional effort and/or the use of further resources compared to a generic process type. As we already asserted above, this would not be acceptable for business processes. When the authors discuss the benefits of their contribution, they do not focus on substitutability. Instead, they show how their approach fosters the solution of certain abstraction problems. For instance: Dynamic change of a workflow type is supported by the inheritance rules, since they make sure that the extended workflow still satisfies the contracts to be fulfilled by a superordinate workflow

type. However, if somebody applies a modification to realize a so called "ad hoc" workflow, it will often not be required to preserve the execution of the original workflow type. A further benefit they mention is the support of "management information". They argue that managers are often not interested in the details of a workflow. The level of abstraction, they are interested in, could be captured by a generic workflow type that is reduced to the essential tasks and conditions. The transformation rules defined by van der Aalst and Basten would allow for showing an instance of a subordinate workflow type as if it was an instance of the superordinate type. However, this is rather an example for defining a projection that satisfies certain information needs than a convincing example for generalisation/specialisation. It seems that the authors' primary focus is on defining a formal conception that allows for building tools which can, e.g. check whether a workflow type is a formally valid specialisation of another type. While there is no doubt that formalisation can be of great value, it does not help much formalising a conception that is not satisfactory. The authors themselves realize that the solution they suggest is of limited use only: "Clearly, this does not provide a complete solution for the four problems presented in this paper." (Aalst and Basten 2003, p. 405) They do, however, indicate how their conception of specialisation could also be interpreted: as the construction of a new *variant* of a workflow type.

4.3 An alternative Approach to Process Specialisation

Wyner and Lees suggest an approach to defining a specialisation concept for process models that is worthwhile mentioning (Wyner and Lee 1999). While the authors compare their work to research on specializing behaviour in object-oriented software systems, they relate it clearly to business processes – although they do not use this term explicitly. They speak of "process" instead. They emphasize a formal approach. Later, they published a formal specification of a conception of specialisation they suggest (Wyner and Lee 2002). This conception is remarkable because it seems to reverse common concepts of specialisation: Instead of adding further features, they suggest to delete parts of a process type in order to specialize it. Nevertheless they emphasize substitutability: "a process p_1 is a specialisation of a process p_0 if every instance of p_1 is also an instance of p_0, but not necessarily vice versa." (Wyner and Lee 1999, p. 11) They suggest an abstraction that regards a process type as a "set of possible behaviors", which they refer to as the "extension" of a process type. Specializing a process type happens through reducing its extension. They demonstrate the proposed conception by applying it to state charts and to data flow diagrams. We focus on state charts here, because functional abstractions – as we have seen – do not provide a reasonable foundation for defining specialisation semantics for business process types. Wyner and Lee model processes in various types of restaurants ("fast food", "full service", "buffet" etc.). The corresponding state charts are depicted in Figure 14. Subsequently, they generalize these special process types to a generic process type. The generic process type includes all special state charts as possible paths of execution (see Figure 15). Specialisation can be specified as a function that results in a projection of the state chart which characterized the subordinate process type. This projection needs to describe (at least) one possible path of execution within the set of

paths defined for the superordinate process type. Hence, an instance of a specialized process would represent a potential path of execution of an instance of the superordinate process. Hence, one could speak of "observation consistency", even though the conception of specialisation is clearly different from that proposed by Schrefl and Stumptner. However, the specialized process types are not invocation consistent: It is, for example, not possible to set an instance of the specialized process "full service restaurant" to the initial state "COOK" – although this should be possible according to the state chart defined for the superordinate process type. Also, it would not be possible to change the state "COOK" to "SERVE" – which is an option for instance of the generic process type. Hence, substitutability, although demanded for by the authors, is clearly restricted. At the same time it is obvious that the generic process type is abstract in the sense that it must not have instances. Even more important is the fact that the concept of generalisation they apply is neither compliant with generalisation in natural language nor in formal systems: A generic concept represents the common features of a set of more special concepts – in other words: the intersection of the corresponding sets of features. Wyner and Lee use the union of these sets instead.

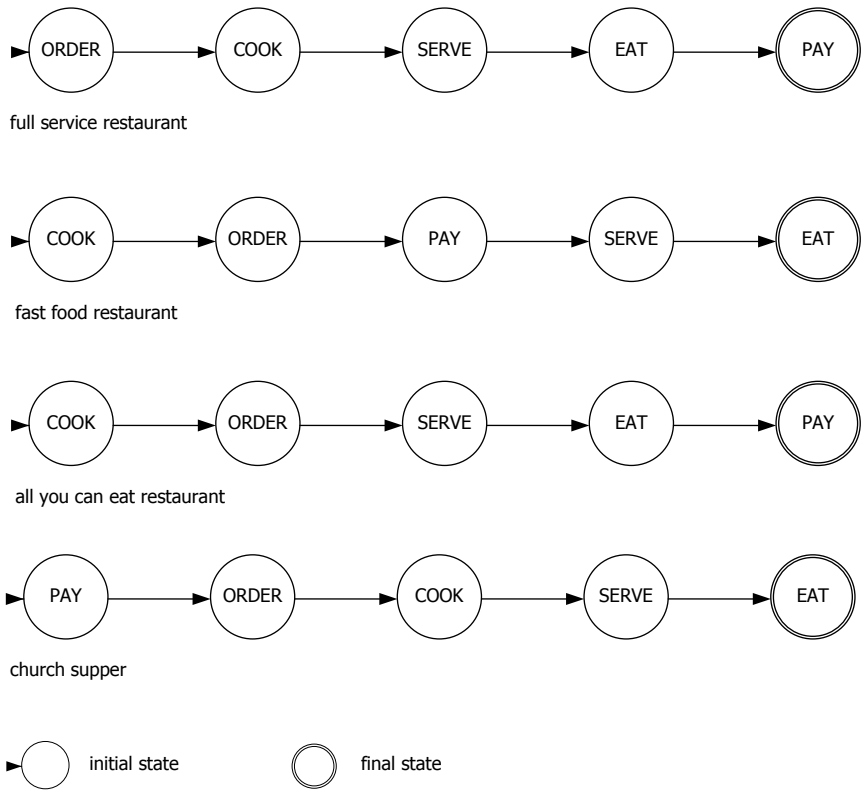

Figure 14: State charts of processes in various types of restaurants – (Wyner and Lee 1994)

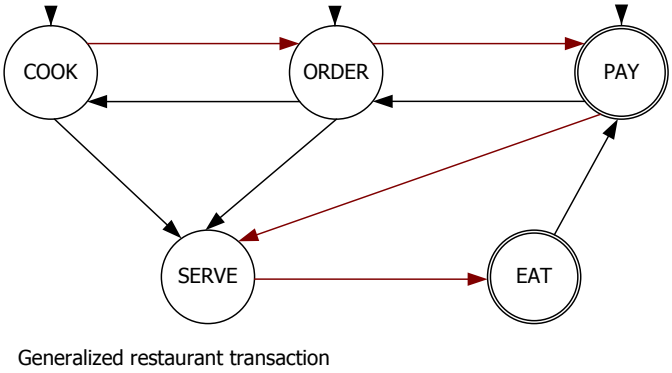

Figure 15: Generalized state chart and included "specialized" process – adapted from (Wyner and Lee 1994)

The authors do not realize that their conception of specialisation does not satisfy the demand for suitability. However, it seems that they do not feel too comfortable with the fact that the conception they propose represents a clear contrast to other conceptions of specialisation. In a somewhat cumbersome discussion they ask whether there are two kinds of specialisation – comparing specialisation by adding features (attributes) and by deleting features (states and activities). While the discussion does not result in a convincing answer to the question, it seems that two aspects contributed to the confusion they caused. Firstly, it appears that Wyner and Lee confuse two perspectives on classes and specialisation respectively. An intentional view stresses the definition of a class by its features. Specialisation in this view suggests adding further features to a class in order to specialize it. Applying this conception of specialisation to dynamic abstractions would result in the extension of behaviour as it is suggested by many. An extensional view regards classes as sets. In this perspective, a subclass is a restriction in the sense that it demands for features that apply only to a subset of the set that represents the superclass. Secondly, and more important for our analysis, they emphasize that specialisation does not have to imply adding further features, but may also be realized by deleting inherited features. They do not provide a convincing justification for this argument. Instead they give a misleading example. Nevertheless, they touch an important aspect. It seems reasonable to assume that a concept that we would regard as a specialisation of another process type may restrict the inherited features somehow. With respect to static abstractions, this can be illustrated with a classical example: square and rectangle. While it is reasonable to regard a square as a special kind of rectangle, it seems appropriate to delete one attribute of rectangle – that stores the length of one side – to make it a square. However, that would compromise substitutability. A possible solution to this problem is to add a constraint. In order not to violate the demand for substitutability, it should not change the essential features of the concept, but only restrict their use somehow. The use of constraints for defining a specialized type is conceivable for business process types, too. Take, for instance, two business process types – sale and sale for members – that are identical except for one thing: For regular customers, the process "financial transaction" allows the use of cash, vari-

ous credit cards or the specific corporate credit card. Those customers that qualify as members have to use the corporate credit card. Unfortunately, this kind of extension would not satisfy the demand for suitability either. If, in the above example, a regular customer was forced to use the corporate credit card, the business process would probably not be accepted anymore. It seems that Wyner and Lee confused specialisation with instantiation. The "general" process in Figure 14 is rather a (meta) model of all possible process models. To summarize: While it is based on a coherent formal foundation, the proposal made by Wyner and Lee is not a satisfactory conception for the specialisation of business process types.

4.4 Process Specialisation in Knowledge Representation

A further approach to process specialisation was inspired by work on the "process handbook" (Malone et al. 1999). It originates in the knowledge representation or semantic web community addresses this challenge (Bernstein et al. 2005, Ferndriger et al. 2008). Compared to the previous approaches it is less ambitious and more ambitious at the same time. On the one hand, it does not account for the control flow. Hence, it is restricted to functional abstractions. On the other hand, it is aimed at a more powerful conception of specialisation that allows for both adding and deleting subprocesses. Deleting a process is clearly a non-monotonic extension, since it affects the inherited concepts. Therefore, the approach is based on a conception of non-monotonic inheritance specified in formal ontology languages. In (Bernstein et al. 2005), the Semantic Web Services Language (SWSL) is used, in (Ferndriger et al. 2008), the approach is specified through an extension of OWL-S (Ontology Web Language – Services). The proposed conception allows for changing and deleting inherited services without producing contradictions – which would have to be expected in monotonic systems. The main focus of this approach is to take advantage of pragmatic specialisation relationships – that do not guarantee substitutability – and still allow for consistent reasonsing. Figure 16 illustrates the flexibility the approach allows for. The "specialized" business process type "Sell in retail store" has only one process in common with its superordinate business process type. The remaining processes are either replaced or deleted.

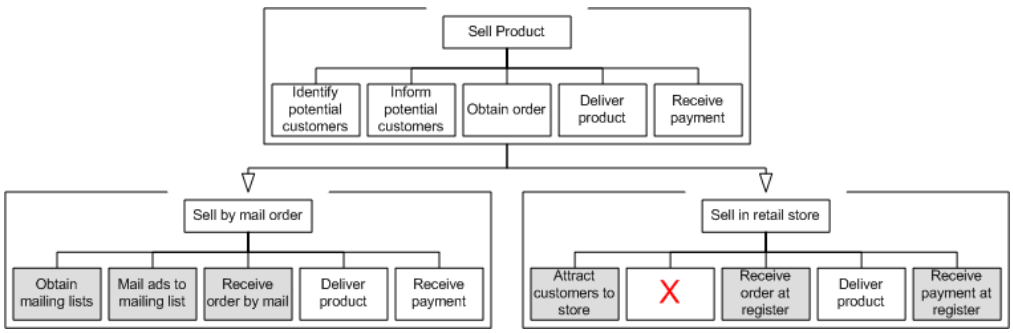

Figure 16: Illustration of non-monotonic inheritance (Bernstein et al. 2005)

Non-monotonic reasoning is intended to offer more tolerant and flexible reasoning capabilities. This claim is usually justified by referring to the power of human reasoning, which can be successful regardless of logical inconsistencies caused e.g. by exceptions. The example in

Figure 17 illustrates the effect of non-monotonic reasoning for deductive retrieval procedures by contrasting it to traditional logics. It is based on a declarative representation of the model shown in Figure 16.

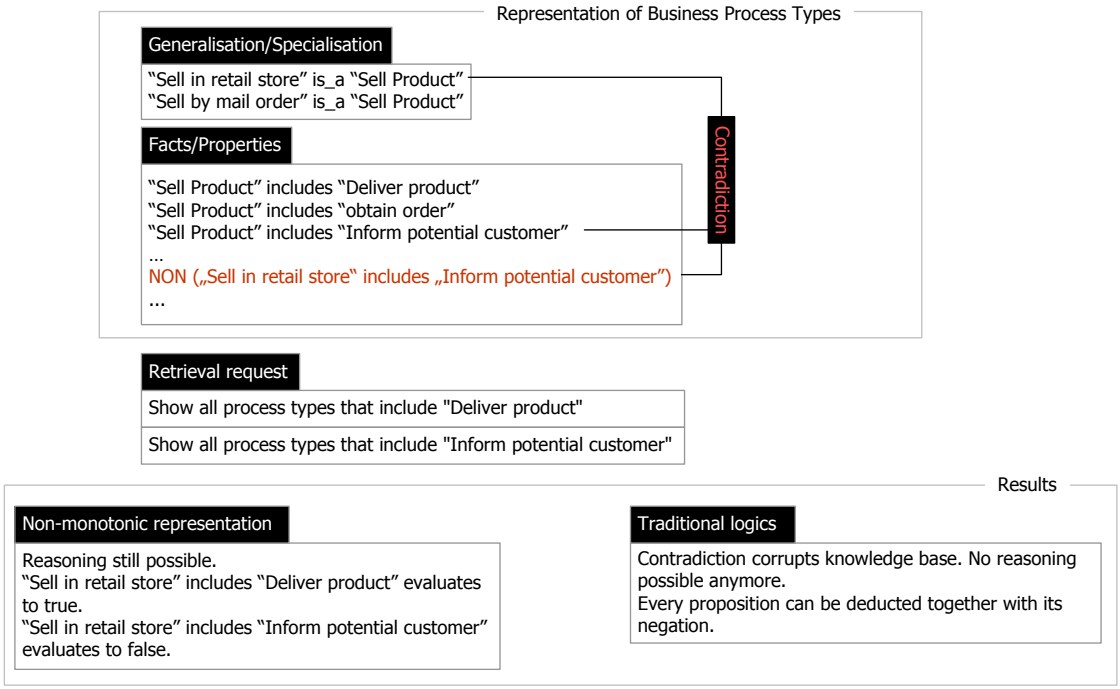

Figure 17: Monotonic vs. non-monotonic reasoning

What is the potential use of non-monotonic specialisation for business process modelling? First of all: While the authors claim to provide a more versatile and natural conception of specialisation, it also allows for creating strange and misleading abstractions. From a formal point of view, any specialisation relationship could be reversed. For the reconstruction of an existing technical language, where generalized terms are already defined as such, this may be acceptable. However, with respect to the motivation of our investigation, this approach is not satisfactory: It reduces specialisation to an arbitrary concept, since it allows every process type to be defined as a specialisation of any other process – et vice versa. In addition to that there are two reasons why this approach is not suited for our purpose. On the one hand, the intended applications are different. The proposal of a non-monotonic conception of specialisation targets business processes in electronic commerce that may be represented in large, international repositories with hundreds or thousands of entries. For such a scenario, retrieval and reasoning-based retrieval can become a major requirement. In our case, retrieving business process models is certainly not irrelevant, but it is not a major issue. On the other hand, languages for enterprise modelling are usually specified with meta models. Reconstructing the language specification using a formal language that supports non-monotonic reasoning would imply to give up all advantages that come with a meta modelling approach. Nevertheless, it would not require a tremendous effort to transform a business process model into a declarative representation using, e.g., an extension of OWL-S. Therefore, it could be an option for supporting retrieval in large repositories of business process models. Despite

its ambitious formal foundation, the proposed approach could serve as a model for a pragmatic concept of inheritance, since it allows for either adding, modifying or deleting inherited processes.

5 Assessment and Consequences

Our analysis of generalisation/specialisation of business process types suggests the following hypothesis:

> No matter whether a business process type is "specialized" by adding further events, processes, or constraints, its instances cannot be substituted for the instances of the superordinate type in an acceptable way.

Firstly, this hypothesis is supported by the very nature of a business process: Changing it by adding further parts will always result in additional effort and/or in the consumption of resources. A "specialized" process of this kind will not be acceptable as a substitute. Secondly, the hypothesis is indirectly supported by the failure of previous attempts: None of those succeeded in producing a satisfactory solution. Hence, it may seem adequate to not further pursue a specialisation concept for business process types. However, the benefits to be expected from a corresponding abstraction are appealing. Therefore, it may be an option to make do with a conception of specialisation that is restricted to functional abstractions of business process types.

5.1 Focus on Functional Abstractions as an Alternative?

A business process type can be modelled without accounting for the flow of control by only representing the processes (or function) it includes. A conception of specialisation for functional abstractions could be defined according to that of static abstractions: A business process type would be specialized by supplementing its processes with further processes. Different from a dynamic abstraction, the order to processes would not matter. Hence, adding further features would be monotonic. Every specialized process type would maintain the features of its superordinate process type. Figure 18 shows how this kind of specialisation could be applied to a functional abstraction of the above example.

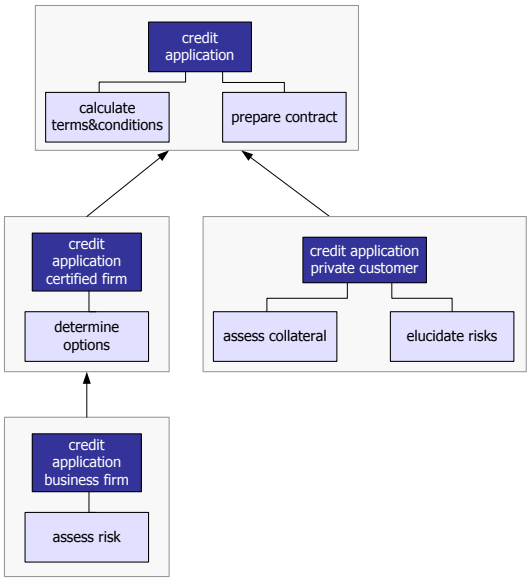

Figure 18: Generalisation/specialisation for functional abstractions of business processes

It seems hopeless to define a conception of specialisation/inheritance for dynamic abstractions of business process types, because the consequences of adding or deleting processes in a superordinate business process type cannot be specified on a generic level. As a consequence, it seems that the demand for substitutability cannot be satisfied. If one is content with functional abstraction and a pragmatic conception of inheritance, two questions need to be addressed. First: How could one utilize a functional conception of inheritance to support the convenient and safe construction of business process models, i.e. of dynamic abstractions? Second: To exploit the potential of inheritance, tool support is mandatory. Therefore we cannot neglect implementation issues here – even though our main focus is on semantics. The following example addresses both questions. Figure 19 shows a hierarchy of functional representations that is based on a pragmatic, functional conception of process inheritance. Figure 20 illustrates how these inheritance relationships could be represented in a corresponding modelling tool and how they could be used to support the creation of dynamic abstractions. However, there is still a remarkable challenge to overcome: Reusing an inherited subprocess in the context of a specialized business process type will usually imply a different context, i.e. different triggering and/or resulting events. Therefore, it is necessary to define a concept of subprocess that allows for abstracting from the context, i.e. to reduce a subprocess to its invariant core. In Figure 20 this intended abstraction is represented by a box inside the process symbols. Specializing the functional abstraction of a business process type means at first that all processes of the superordinate business process type are inherited. Subsequently, new processes have to be added and – if necessary – inherited processes are either modified or deleted. Based on such a definition, a user who wants to specify a business process model could be provided with the corresponding functional abstraction, i.e. with the collection of the included processes. For creating a specialized business process model, he would access the collection of the specialized functional abstraction, i.e. the inherited ones, the newly added and the modified (see Figure 20). Studying the business process model of the superordinate business process type may help with defining the control structure.

To allow for efficient reuse – which includes the support of consistent maintenance – it is important that inherited processes are represented as references. Every modification to the process definition in the superordinate business process model would then be immediately effective in the corresponding subordinate processes. However, it may be that the modification should not be effective in the subordinate business process type. In that case, the reference has to be replaced by a copy of the process specification, which could be modified subsequently.

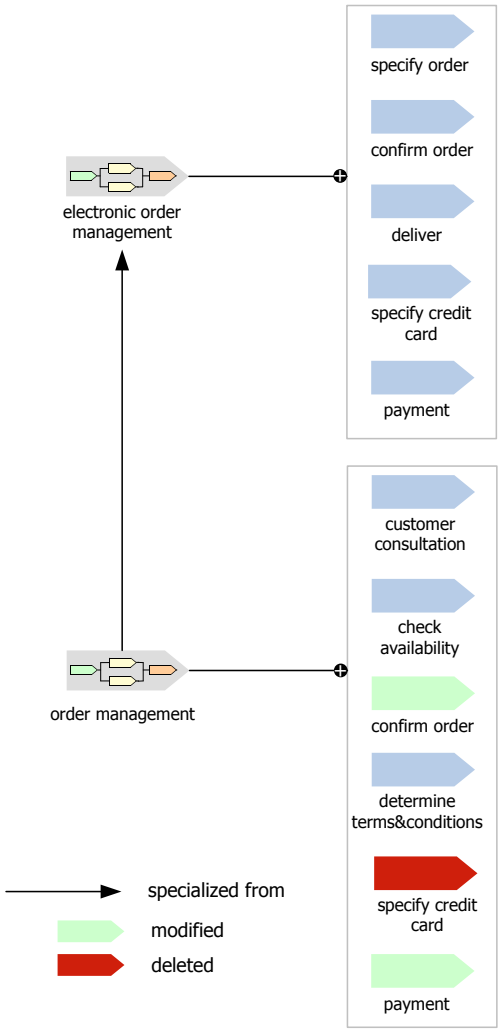

Figure 19: Example of pragmatic specialisation of functional abstraction

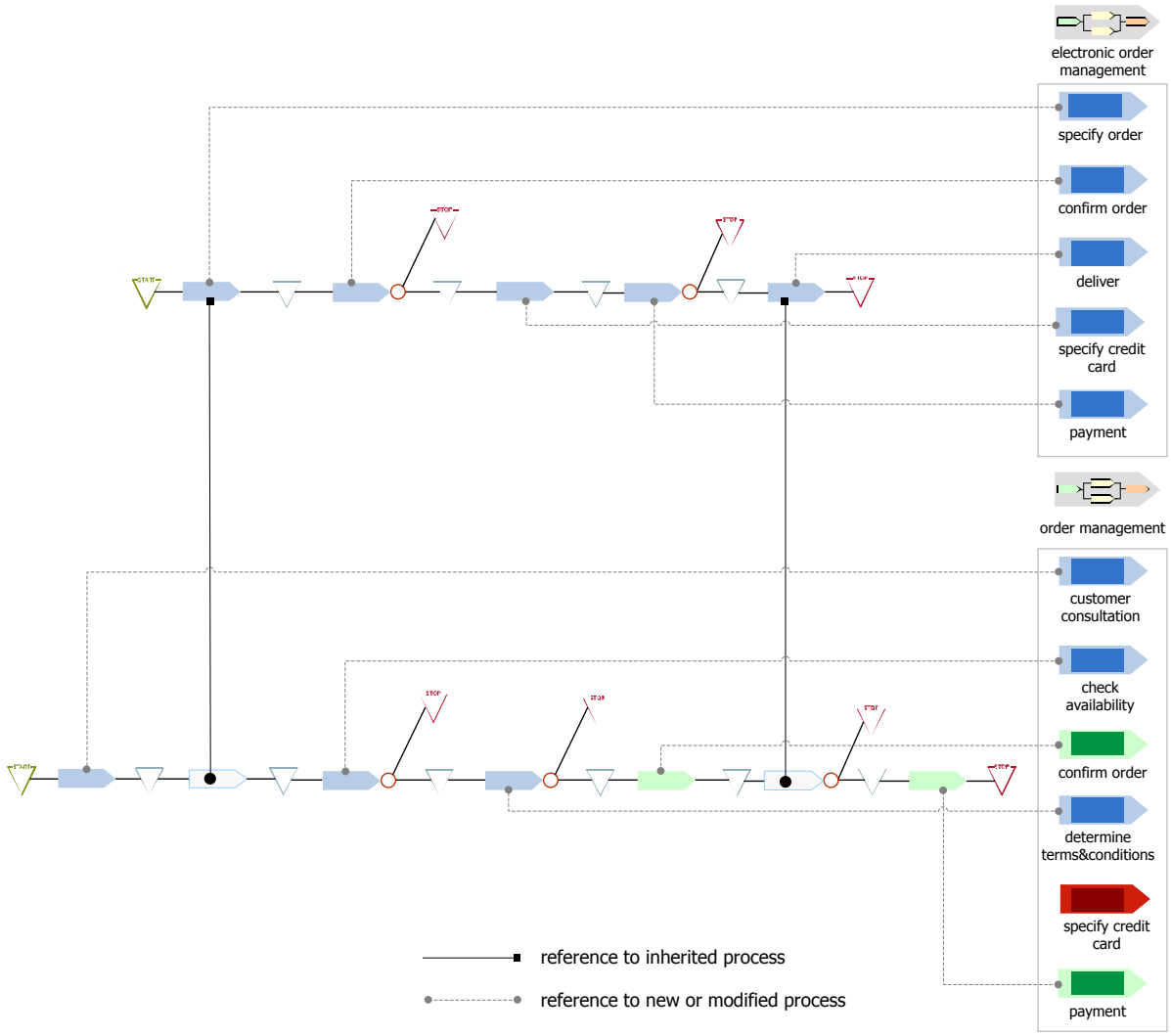

Figure 20: Constructing dynamic abstractions through references to functional abstractions

Figure 21 illustrates the difficulties related to abstracting a subprocess to its invariant core. At first sight, the example seems to represent a fortunate situation: "check credibility of new customer" appears to be a special case of "check credibility of existing customer". Hence, the common core could be represented in an (abstract) super process type that can be specialized into the two concrete types by specializing the corresponding abstract event types.

However, such a conceptualisation is not convincing for two reasons. First, it is based on a simplification since it does not account for internal differences in processing credibility checks. Second, it does not completely satisfy the substitutability constraint, since it may cause the problem of covariant redefinitions: If "check credibility of existing customer" together with the triggering and resulting specialized event types is substituted for the corresponding abstract concepts, no problem would occur. However, if "check credibility of existing customer" is used in the general context of "check credibility", and assigned the resulting event type "credibility of new customer is ok" – which should be possible, since it is defined in the super type, a type error would occur.

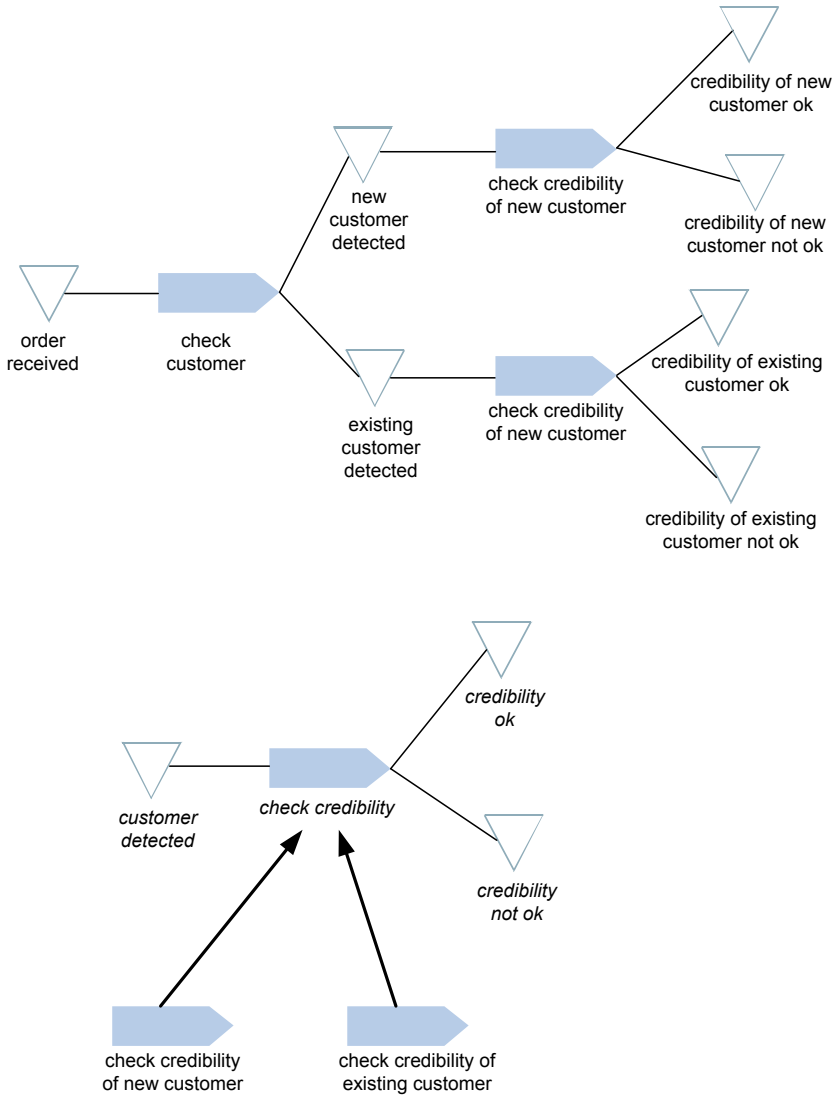

Figure 21: Using abstract Super Process Types and Corresponding abstract Event Types to represent Common Core

Hence, there is need for a relaxed concept of specialisation that fosters reuse but excludes the problem of covariant redefinitions.

5.2 Instantiation instead of Specialisation?

Apart from aiming at a relaxed conception of process specialisation one could also pursue different kinds of abstraction. Using process meta models as abstractions over a set of similar process models could be such an approach. Concrete process types would then be instantiated from the meta model instead. As a consequence, this approach would require a further level of abstraction: Usually, the modelling language is specified in a meta model on the M_2 level. If the M_2 level is used for specifying domain-specific meta concepts of process and events, an additional M_3 layer would be required for specifying the generic language concepts, such as “Event”, “Process” etc. The example in Figure 22 illustrates the idea. The concepts on the M_3 level serve to specify a process modelling language that can be used across

various domains. The M_2 level serves to represent concepts that are restricted to a narrower domain, e.g. to one specific organisation only. Hence, the range of reuse is much smaller compared to the M_2 level. In the example in Figure 22 the meta process type "check credibility" defines the set of possible concrete process types in the targeted domain. A particular process type on the M1 level would be instantiated from the meta type by specifying a selection of predefined features, i.e. event types that would be instantiated from meta event types and procedures (either for manual or machine execution) that would be instantiated by instantiating the meta concepts they include. For instance: If a procedure includes a statement "check account" where "account" is marked as a meta class, it would have to be instantiated in a valid instance, e.g. "account receivable".

While it needs further investigations to adequately assess the potential of this approach, it comes with a major challenge: Existing language architectures where modelling languages are specified on M_2 and models are typically located on M_1 would not be appropriate anymore. On the one hand, this would imply to redesign modelling languages. On the other hand, even more challenging, this would require building modelling tools that allow for multi-level modelling: Users could define their own, local language as an instantiation of a given language specification. Subsequently they would use their customized concepts in a corresponding customized modelling tool that would allow for editing instantiations of these concepts.

In their approach to define process variants Puhlmann et al. use abstract subprocess types that can be instantiated to concrete subprocess types (Puhlmann et al. 2005). However, they do not introduce a specification of meta subprocess types. Note that the approach proposed by Wyner and Lee is aimed at a similar approach. However, they speak erroneously of specialisation, thereby ignoring the problems related to deal with two different levels of abstraction.

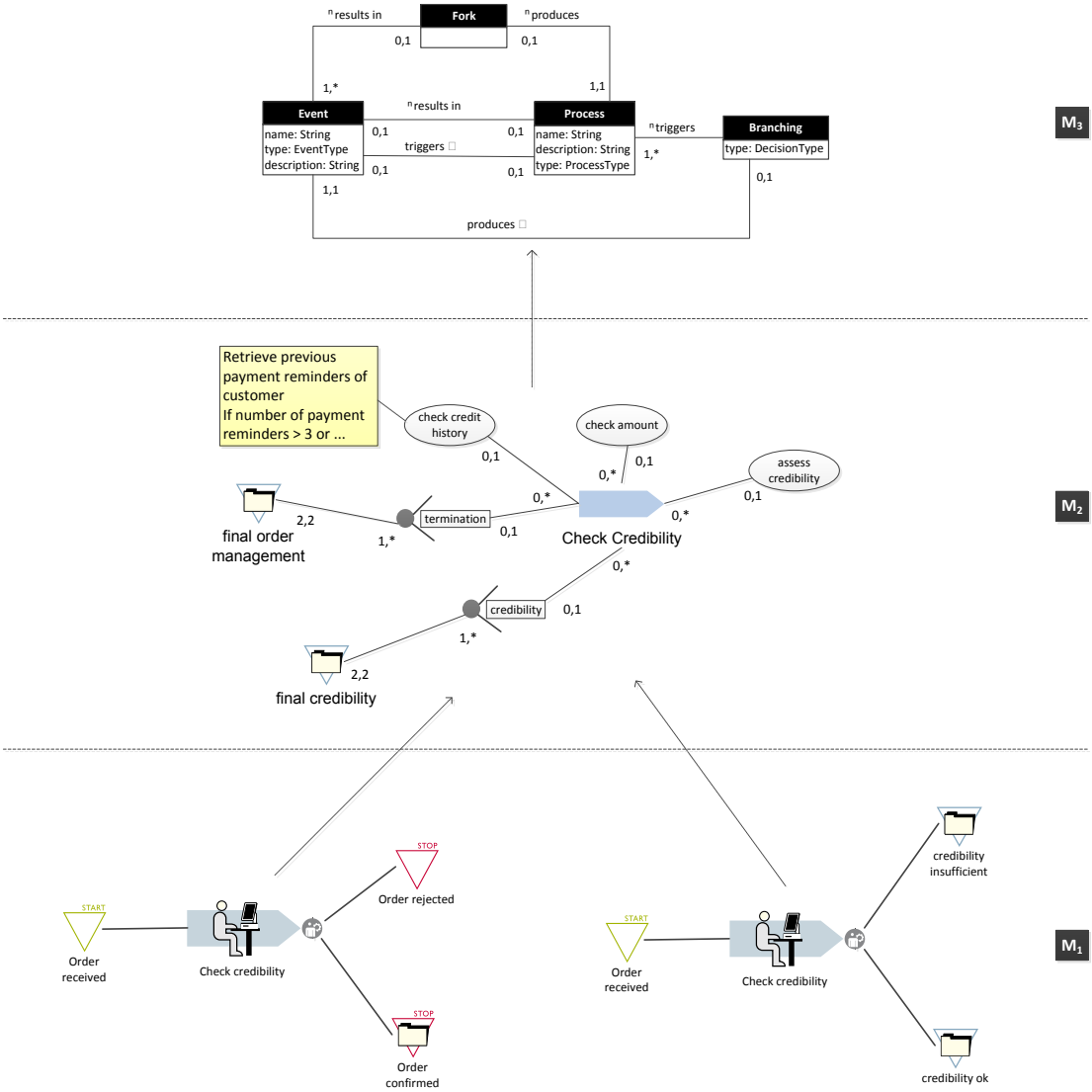

Figure 22: Reuse through Instantiation

6 Conclusions and Future Research Directions

Our investigation of abstraction concepts in process modelling in general, of process specialisation in particular resulted in a sobering insight. Compared to static models, there is a tremendous lack of abstraction in process modelling. As a consequence, there is hardly any support for reuse and secure maintenance. With respect to the relevance of business process modelling and the growing number of business process models this lack of abstraction creates a severe problem. Various approaches to define concepts of process variants or process configurations (Hallerbach 2009, Hallerbach et al. 2009, Schnieders 2008, Gottschalk 2008, La Rosa et al. 2011, Rosemann and van der Aalst 2007) or rules for constructing valid particular process models from reference models (Becker at al. 2007, Delfmann 2006) have addressed this problem in part. However, they remain unsatisfactory to a wide extent. Approaches to managing process variants lack a general conceptualisation of a common process core, all versions share – and that could be modified resulting in consistent updates of all versions. As a consequence, they require sophisticated approaches to managing process variants consistently. Rules for constructing process models from reference models lack the formal semantics that is required for automating the update of process models after a corresponding reference model had been modified. Apart from that they resemble the instantiation of process types presented in 5.2 as they are based on a definition of all possible particular process types in a reference model – that is, however, on the same level of abstraction.

There seem to be three major directions for future research. Due to the fact that a concept of process specialisation that would satisfy the substitutability constraint is not feasible, the only option for further research in this area is to aim at a relaxed conception of process specialisation that is still beneficial with respect to reuse and maintenance. Introducing a further level of abstraction to allow for reuse through instantiation seems to be a promising approach that deserves further research. It is, however, very demanding since it requires new language architectures and corresponding modelling tools. It also requires advanced users that are aware of the different levels of abstraction. Parameterized process and event types offer a further option (Puhlmann et al. 2005) that provides for flexibility without requiring an additional level of abstraction. Future work on process variants could be aimed at integrating specialisation, instantiation and parameterization to develop a sophisticated concept of process variant.

References

van der Aalst, W.M.P.; Basten, T. (1997): Life-cycle Inheritance: A Petri-net-based Approach. In: Azema, P.; Balbo, G. (Eds.): Application and Theory of Petri Nets 1997. LNCS, vol. 1248, Springer: Berlin et al. 1997, pp. 62–81

van der Aalst, W.M.P. (1999a): Inheritance of Workflows: An approach to tackling problems related to change. Computing Science Reports 99/06, Eindhoven University of Technology, Eindhoven 1999

van der Aalst, W.M.P. (1999b): Inheritance of Workflow Processes: Four Problems-One Solution? In: Cummins, F. (Ed.): Proceedings of the Second OOPSLA Workshop on the Implementation and Application of Object-Oriented Workflow Management Systems, pp. 1-22, Denver, Colorado, 1999. (Electronic proceedings, http://st.cs.uiuc.edu/OOPSLA99/, download 09-08-01)

van der Aalst, W.M.P.; Basten, T. (2002a): Inheritance of Workflows: An Approach to Tackling Problems Related to Change. In: Theoretical Computer Science, 270, 1-2, 2002, pp. 125-203

van der Aalst, W.M.P.; Basten, T. (2002b): Inheritance of Interorganizational Workflows to Enable Business-to-Business E-commerce. Electronic Commerce Research, Vol.2, No. 3, 2002, pp. 195-231

van der Aalst, W.M.P. (2003): Inheritance of Business Processes: A Journey Visiting Four Notorious Problems. In: H. Ehrig, H.; W. Reisig, W.; G. Rozenberg, G.; and H. Weber, H. (Eds.): Petri Net Technology for Communication Based Systems. LNCS, vol. 2472, Springer: Berlin et al. 2003, pp. 383-408

Becker, J.; Delfmann, P.; Knackstedt, R. (2007): Adaptive Reference Modeling: Integrating Configurative and Generic Adaptation Techniques for Information Models. In: Becker, J.; Delfmann, P. (Eds.): Reference Modeling. Efficient Information Systems Design Through Reuse of Information Models. Heidelberg: Physica-Verlag: Heidelberg 2007, p. 27–58

Bernstein, A.; Grosof, B.; Kifer, M. (2005): Beyond Monotonic Inheritance: Towards Non-Monotonic Semantic Web Process Ontologies. In: Proceedings of W3C Workshop On Frameworks for Semantics in Web Service. (http://www.w3.org/2005/04/FSWS/Submissions/45/w3c-ws-submission.html), download on 2012-05-09

Delfmann, P.(2006): Adaptive Referenzmodellierung: Methodische Konzepte zur Konstruktion und Anwendung wiederverwendungsorientierter Informationsmodelle, Logos: Berlin 2006

Ducournau, R. (2002): "Real World" as an argument for covariant specialisation in programming and modeling. In: Bruel, J.-M.; and Z. Bellahsène, Z. (Eds.): Advances in Object-Oriented Information Systems, OOIS'02 Workshops Proceedings, Lecture Notes in Computer Science, vol. 2426, p. 3–12, Springer: Berlin, Heidelberg, New York 2002

Ferndriger, S.; Bernstein, A. et al. (2008): Enhancing Semantic Web Services with Inheritance. International Semantic Web Conference, Karlsruhe 2008, p. 162-177

Frank, U. (2003): Ebenen der Abstraktion und ihre Abbildung auf konzeptionelle Modelle - oder: Anmerkungen zur Semantik von Spezialisierungs- und Instanzierungsbeziehungen. In: EMISA FORUM, 23. Jg., Heft 2, 2003, p. 14-18

Frank, U. (2011). MEMO Organisation Modelling Language (OrgML) - Requirements and Core Diagram Types. ICB Research Report No. 47, University Duisburg-Essen 2011

Gottschalk, F.; van der Aalst, W.M.P.; Jansen-Vullers, M., H. et al. (2008): Configurable Workflow Models. In: International Journal of Cooperative Information Systems 17 (02), 2008, p. 177–221

Hallerbach, A. (2009): Management von Prozessvarianten. Dissertation, Universität Ulm 2009

Hallerbach, A.; Bauer, T.; Reichert, M. (2009): Issues in Modeling Process Variants with Provop. In: 4th International Workshop on Business Process Design (BPD'08), Milan 2009, p. 56-67

Kiczales, G., J. Hugunin, et al. (2003). Aspect-Oriented Programming. Technical Report, Palo Alto Research Center 2003

Kiczales, G., J. Lamping, et al. (1997). Aspect-Oriented Programming. European Conference on Object-Oriented Programming. 1997, p. 220–242

La Rosa, M.; Dumas, M.; ter Hofstede, A. H. M. et al. (2011): Configurable multi-perspective business process models. In: Information Systems 36 (2), 2011, p. 313-340

Liskov, B. H.; Wing, J. M. (1994): A behavioural notion of subtyping. In: ACM Transactions on Programming Languages and Systems. Vol. 16, No. 6, 1994, p. 1811–1841

Malone, T.W.; Crowston, K. et al. (1999): Tools for inventing organizations: Toward a handbook of organizational processes. In: Management Science, vol. 45, no. 3, 1999, p. 425-443

Meyer, B. (1997): Object-Oriented Software Construction. 2nd Ed., Prentice Hall 1997

OMG (2007): OMG Unified Modeling Language (OMG UML), Superstructure. Version 2.1.2, 2007

OMG (2011): Business Process Model and Notation (BPMN). Version 2.0, 2012, (http://www.omg.org/spec/BPMN/2.0) download on 2012-05-09

Puhlmann, F.; Schnieders, A.; Weiland, J. et al. (2005): Variability Mechanisms for Process Models. PESOA-Report Nr. 17/2005, Daimler-Chrysler Research and Technology, Hasso-Plattner-Institut 2005

Rosemann, M.; van der Aalst, W.,M.,P. (2007): A configurable reference modelling language. In: Information Systems (32), 2007, p. 1–23

Scheer, A.W. (1999): ARIS – Business Process Modeling. 3rd Edition. Springer: Berlin, Heidelberg, New York 1999

Schnieders, A. (2008): Modellierung und Instantiierung variantenreicher Prozesse in der Softwareproduktfamilienentwicklung. Dissertation, Universität Potsdam 2008

Schrefl, M.; Stumptner, M. (2002): Behaviour-Consistent Specialisation of Object Life Cycles. In: ACM Transactions on Software Engineering and Methodology, Vol. 11, No. 1, 2002, pp. 92–148

Wegner, P.; Zdonik, S.B. (1988): Inheritance as an Incremental Modification Mechanism or What Like Is and Isn't Like. In: Gjessing, S.; Nygaard, K. (Eds.): Proceedings ECOOP '88, LNCS 322, Springer: Berlin, Heidelberg, New York 1988, pp. 55-77

Wyner, G.M.; Lee, J. (1995): Applying Specialisation to Process Models. In: Proceedings of Conference on Organizational Computing Systems. Milipitas, Ca., 1995, pp. 290-301

Wyner, G.M.; Lee, J. (2002): Process Specialisation: Defining Specialisation for State Diagrams. In: Computational & Mathematical Organization Theory, Vol. 8, No. 2, 2002, pp. 133-155

Previously published ICB - Research Reports

2012

No 50 (March)

Adelsberger, Heimo; Drechsler, Andreas; Herzig, Eric; Michaelis, Alexander; Schulz, Philipp; Schütz, Stefan; Ulrich, Udo: "Qualitative und quantitative Analyse von SOA-Studien – Eine Metastudie zu serviceorientierten Architekturen"

2011

No 49 (December 2011)

Frank, Ulrich: "MEMO Organisation Modelling Language (OrgML): Focus on Business Processes"

No 48 (December 2011)

Frank, Ulrich: "MEMO Organisation Modelling Language (OrgML): Focus on Organizational Structure"

No 47 (December 2011)

Frank, Ulrich: "MEMO Organisation Modelling Language (OrgML): Requirements and Core Diagram Types"

No 46 (December 2011)

Frank, Ulrich: "Multi-Perspective Enterprise Modelling: Background and Terminological Foundation"

No 45 (November 2011)

Frank, Ulrich; Strecker, Stefan; Heise, David; Kattenstroth, Heiko; Schauer, Carola: "Leitfaden zur Erstellung wissenschaftlicher Arbeiten in der Wirtschaftsinformatik"

No 44 (September 2010)

Berenbach, Brian; Daneva, Maya; Dörr, Jörg; Frickler, Samuel; Gervasi, Vincenzo; Glinz, Martin; Herrmann, Andrea; Krams, Benedikt; Madhavji, Nazim H.; Paech, Barbara; Schockert, Sixten; Seyff, Norbert (Eds): "17th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2011). Proceedings of the REFSQ 2011 Workshops REEW, EPICAL and RePriCo, the REFSQ 2011 Empirical Track (Empirical Live Experiment and Empirical Research Fair), and the REFSQ 2011 Doctoral Symposium"

No 43 (February 2011)

Frank, Ulrich: "The MEMO Meta Modelling Language (MML) and Lnguage Architecture – 2nd Edition"

2010

No 42 (December)

Frank, Ulrich: "Outline of a Method for Designing Domain-Specific Modelling Languages"

No 41 (December)

Adelsberger,Heimo; Drechsler, Andreas (Eds): "Ausgewählte Aspekte des Cloud-Computing aus einer IT-Management-Perspektive – Cloud Governance, Cloud Security und Einsatz von Cloud Computing in jungen Unternehmen"

No 40 (October 2010)

Bürsner, Simone; Dörr, Jörg; Gehlert, Andreas; Herrmann, Andrea; Herzwurm, Georg; Janzen, Dirk; Merten, Thorsten; Pietsch, Wolfram; Schmid, Klaus; Schneider, Kurt; Thurimella, Anil Kumar (Eds): "16th International Working Conference on Requirements Engineering: Foundation for Software Quality. Proceedings oft he Workshops CreaRE, PLREQ, RePriCo and RESC"

No 39 (May 2010)

Strecker, Stefan; Heise, David; Frank, Ulrich: "Entwurf einer Mentoring-Konzeption für den Studiengang M.Sc. Wirtschaftsinformatik an der Fakultät für Wirtschaftswissenschaften der Universität Duisburg-Essen"

No 38 (February 2010)

Schauer, Carola: "Wie praxisorientiert ist die Wirtschaftsinformatik? Einschätzungen von CIOs und WI-Professoren"

No 37 (January 2010)

Benavides, David; Batory, Don; Grunbacher, Paul (Eds.): "Fourth International Workshop on Variability Modelling of Software-intensive Systems"

2009

No 36 (December 2009)

Strecker, Stefan: "Ein Kommentar zur Diskussion um Begriff und Verständnis der IT-Governance - Anregungen zu einer kritischen Reflexion"

No 35 (August 2009)

Rüngeler, Irene; Tüxen, Michael; Rathgeb, Erwin P.:"Considerations on Handling Link Errors in STCP"

No 34 (June 2009)

Karastoyanova, Dimka; Kazhamiakan, Raman; Metzger, Andreas; Pistore, Marco (Eds.): "Workshop on Service Monitoring, Adaption and Beyond"

No 33 (May 2009)

Adelsberger,Heimo; Drechsler , Andreas; Bruckmann, Tobias; Kalvelage, Peter; Kinne, Sophia; Pellinger, Jan; Rosenberger, Marcel; Trepper, Tobias: „Einsatz von Social Software in Unternehmen – Studie über Umfang und Zweck der Nutzung"

No 32 (April 2009)

Barth, Manfred; Gadatsch, Andreas; Kütz, Martin; Rüding, Otto; Schauer, Hanno; Strecker, Stefan: „Leitbild IT-Controller/-in – Beitrag der Fachgruppe IT-Controlling der Gesellschaft für Informatik e. V."

No 31 (April 2009)

Frank, Ulrich; Strecker, Stefan: "Beyond ERP Systems: An Outline of Self-Referential Enterprise Systems – Requirements, Conceptual Foundation and Design Options"

No 30 (February 2009)
Schauer, Hanno; Wolff, Frank: „Kriterien guter Wissensarbeit – Ein Vorschlag aus dem Blickwinkel der Wissenschaftstheorie (Langfassung)“

No 29 (January 2009)
Benavides, David; Metzger, Andreas; Eisenecker, Ulrich (Eds.): “Third International Workshop on Variability Modelling of Software-intensive Systems”

2008

No 28 (December 2008)
Goedicke, Michael; Striewe, Michael; Balz, Moritz: „Computer Aided Assessments and Programming Exercises with JACK“

No 27 (December 2008)
Schauer, Carola: “Größe und Ausrichtung der Disziplin Wirtschaftsinformatik an Universitäten im deutschsprachigen Raum - Aktueller Status und Entwicklung seit 1992”

No 26 (September 2008)
Milen, Tilev; Bruno Müller-Clostermann: “ CapSys: A Tool for Macroscopic Capacity Planning”

No 25 (August 2008)
Eicker, Stefan; Spies, Thorsten; Tschersich, Markus: “Einsatz von Multi-Touch beim Softwaredesign am Beispiel der CRC Card-Methode”

No 24 (August 2008)
Frank, Ulrich: “The MEMO Meta Modelling Language (MML) and Language Architecture – Revised Version”

No 23 (January 2008)
Sprenger, Jonas; Jung, Jürgen: “Enterprise Modelling in the Context of Manufacturing – Outline of an Approach Supporting Production Planning”

No 22 (January 2008)
Heymans, Patrick; Kang, Kyo-Chul; Metzger, Andreas, Pohl, Klaus (Eds.): “Second International Workshop on Variability Modelling of Software-intensive Systems”

2007

No 21 (September 2007)
Eicker, Stefan; Annett Nagel; Peter M. Schuler: “Flexibilität im Geschäftsprozess-management-Kreislauf”

No 20 (August 2007)
Blau, Holger; Eicker, Stefan; Spies, Thorsten: “Reifegradüberwachung von Software”

No 19 (June 2007)
Schauer, Carola: “Relevance and Success of IS Teaching and Research: An Analysis of the ‚Relevance Debate’

No 18 (May 2007)
Schauer, Carola: “Rekonstruktion der historischen Entwicklung der Wirtschaftsinformatik: Schritte der Institutionalisierung, Diskussion zum Status, Rahmenempfehlungen für die Lehre”

No 17 (May 2007)
Schauer, Carola; Schmeing, Tobias: "Development of IS Teaching in North-America: An Analysis of Model Curricula"

No 16 (May 2007)
Müller-Clostermann, Bruno; Tilev, Milen: "Using G/G/m-Models for Multi-Server and Mainframe Capacity Planning"

No 15 (April 2007)
Heise, David; Schauer, Carola; Strecker, Stefan: "Informationsquellen für IT-Professionals – Analyse und Bewertung der Fachpresse aus Sicht der Wirtschaftsinformatik"

No 14 (March 2007)
Eicker, Stefan; Hegmanns, Christian; Malich, Stefan: "Auswahl von Bewertungsmethoden für Softwarearchitekturen"

No 13 (February 2007)
Eicker, Stefan; Spies, Thorsten; Kahl, Christian: "Softwarevisualisierung im Kontext serviceorientierter Architekturen"

No 12 (February 2007)
Brenner, Freimut: "Cumulative Measures of Absorbing Joint Markov Chains and an Application to Markovian Process Algebras"

No 11 (February 2007)
Kirchner, Lutz: "Entwurf einer Modellierungssprache zur Unterstützung der Aufgaben des IT-Managements – Grundlagen, Anforderungen und Metamodell"

No 10 (February 2007)
Schauer, Carola; Strecker, Stefan: "Vergleichende Literaturstudie aktueller einführender Lehrbücher der Wirtschaftsinformatik: Bezugsrahmen und Auswertung"

No 9 (February 2007)
Strecker, Stefan; Kuckertz, Andreas; Pawlowski, Jan M.: "Überlegungen zur Qualifizierung des wissenschaftlichen Nachwuchses: Ein Diskussionsbeitrag zur (kumulativen) Habilitation"

No 8 (February 2007)
Frank, Ulrich; Strecker, Stefan; Koch, Stefan: "Open Model - Ein Vorschlag für ein Forschungsprogramm der Wirtschaftsinformatik (Langfassung)"

2006

No 7 (December 2006)
Frank, Ulrich: "Towards a Pluralistic Conception of Research Methods in Information Systems Research"

No 6 (April 2006)
Frank, Ulrich: "Evaluation von Forschung und Lehre an Universitäten – Ein Diskussionsbeitrag"

No 5 (April 2006)
Jung, Jürgen: "Supply Chains in the Context of Resource Modelling"

No 4 (February 2006)

Lange, Carola: "Development and status of the Information Systems / Wirtschaftsinformatik discipline: An interpretive evaluation of interviews with renowned researchers, Part III – Results Wirtschaftsinformatik Discipline"

2005

No 3 (December 2005)

Lange, Carola: "Development and status of the Information Systems / Wirtschaftsinformatik discipline: An interpretive evaluation of interviews with renowned researchers, Part II – Results Information Systems Discipline"

No 2 (December 2005)

Lange, Carola: "Development and status of the Information Systems / Wirtschaftsinformatik discipline: An interpretive evaluation of interviews with renowned researchers, Part I – Research Objectives and Method"

No 1 (August 2005)

Lange, Carola: „Ein Bezugsrahmen zur Beschreibung von Forschungsgegenständen und -methoden in Wirtschaftsinformatik und Information Systems"

Research Group	Core Research Topics
Prof. Dr. H. H. Adelsberger Information Systems for Production and Operations Management	E-Learning, Knowledge Management, Skill-Management, Simulation, Artificial Intelligence
Prof. Dr. P. Chamoni MIS and Management Science / Operations Research	Information Systems and Operations Research, Business Intelligence, Data Warehousing
Prof. Dr. F.-D. Dorloff Procurement, Logistics and Information Management	E-Business, E-Procurement, E-Government
Prof. Dr. K. Echtle Dependability of Computing Systems	Dependability of Computing Systems
Prof. Dr. S. Eicker Information Systems and Software Engineering	Process Models, Software-Architectures
Prof. Dr. U. Frank Information Systems and Enterprise Modelling	Enterprise Modelling, Enterprise Application Integration, IT Management, Knowledge Management
Prof. Dr. M. Goedicke Specification of Software Systems	Distributed Systems, Software Components, CSCW
Prof. Dr. V. Gruhn Software Engineering	Design of Software Processes, Software Architecture, Usability, Mobile Applications, Component-based and Generative Software Development
PD Dr. C. Klüver Computer Based Analysis of Social Complexity	Soft Computing, Modeling of Social, Cognitive, and Economic Processes, Development of Algorithms
Prof. Dr. T. Kollmann E-Business and E-Entrepreneurship	E-Business and Information Management, E-Entrepreneurship/E-Venture, Virtual Marketplaces and Mobile Commerce, Online-Marketing
Prof. Dr. B. Müller-Clostermann Systems Modelling	Performance Evaluation of Computer and Communication Systems, Modelling and Simulation
Prof. Dr. K. Pohl Software Systems Engineering	Requirements Engineering, Software Quality Assurance, Software-Architectures, Evaluation of COTS/Open Source-Components
Prof. Dr. R. Unland Data Management Systems and Knowledge Representation	Data Management, Artificial Intelligence, Software Engineering, Internet Based Teaching
Prof. Dr. S. Zelewski Institute of Production and Industrial Information Management	Industrial Business Processes, Innovation Management, Information Management, Economic Analyses

For more information visit us on the Web: **http://www.icb.uni-due.de**

ISSN 1860-2770 (Print)
ISSN 1866-5101 (Online)