Process Flexibility: a Survey of Contemporary Approaches

M.H. Schonenberg, R.S. Mans, N.C. Russell, N.A. Mulyar and W.M.P. van der Aalst

Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{m.h.schonenberg,r.s.mans,n.c.russell,nmulyar,w.m.p.v.d.aalst}@tue.nl

Abstract. Business processes provide a means of coordinating interactions between workers and organisations in a structured way. However the dynamic nature of the modern business environment means these processes are subject to a increasingly wide range of variations and must demonstrate flexible approaches to dealing with these variations if they are to remain viable. The challenge is to provide flexibility and offer process support at the same time. Many approaches have been proposed in literature and some of these approaches have been implemented in flexible workflow management systems. However, a comprehensive overview of the various approaches has been missing. In this paper, we take a deeper look into the various ways in which flexibility can be achieved and we propose an extensive taxonomy of flexibility. This taxonomy is subsequently used to evaluate a selection of systems and to discuss how the various forms of flexibility fit together.
Keywords: taxonomy, flexibility, design, change, deviation, underspecification.

1 Introduction

In order to retain their competitive advantage in today's dynamic marketplace, it is increasingly necessary for enterprises to streamline their processes so as to reduce costs and to improve performance. Moreover, it is clear that the economic success of an organisation is highly dependent on its ability to react to changes in its operating environment.

To this end, Process-Aware Information Systems (PAISs) are an desirable technology as these systems support the business operations of an enterprise based on models of both the organisation and its constituent processes. PAISs encompass a broad range of technologies ranging from systems which rigidly enforce adherence to the underlying process model, e.g., workflow systems or tracking systems, to systems which are guided by an implied process model but do nothing to ensure that it is actually enforced, e.g., groupware systems.

Typically, these systems utilise an idealised model of a process which may be overly simplistic or even undesirable from an operational standpoint. Furthermore the models on which they are based tend to be rigid in format and are not able to easily encompass either foreseen or unforeseen changes in the context or environment in which they operate. Up to now, there have not been any broadly adopted proposals or standards offering guidance for developing flexible process models able to deal with these sorts of changes. Instead most standards focus on a particular notation (e.g., XPDL, BPEL, BPMN, etc.) and these notations typically abstract from flexibility issues.

Process flexibility can be seen as the ability to deal with both foreseen and unforeseen changes, by varying or adapting those parts of the business process that are affected by them, whilst retaining the essential format of those parts that are not impacted by the variations. Or, in other words, flexibility is as much about what should stay the same in a process as what should be allowed to change[15]. Different kinds of flexibility are needed during the BPM life cycle of a process. Based on an extensive survey of literature and flexibility support offered by existing tools[1], a range of approaches to achieve process flexibility have been identified. These approaches have been described in the form of a taxonomy which provides a comprehensive catalogue of process flexibility approaches for the control-flow perspective.

The remainder of this paper is organised as follows. Section 2 presents the taxonomy for process flexibility. In Section 3, we use the taxonomy to evaluate the support of process flexibility in several contemporary PAISs, namely ADEPT1, YAWL, FLOWer and Declare. In Section 4 we discuss related work. Finally, we conclude the paper and identify opportunities for future work in Section 5.

2 Taxonomy of Flexibility

In this section, we present a comprehensive description of four distinct approaches that can be taken to facilitate flexibility within a process. All of these strategies improve the ability of business processes to respond to changes in their operating environment without necessitating a complete redesign of the underlying process model, however they differ in the timing and manner in which they are applied. Moreover they are intended to operate independently of each other. These approaches are presented in the form of a taxonomy which aims to define each of them in detail. The taxonomy is applicable to both classical (imperative) and constraint-based (declarative) specifications.

2.1 Specification Approaches

Generally, process behaviour depends on the structure of a process, which can be defined in an imperative or a declarative way. An *imperative approach* focuses on the precise definition of how a given set of tasks has to be performed (i.e., the task order is explicitly defined). In imperative languages, constraints on the execution order are described either via links (or connectors) between tasks and/or data

[1] See [10] for full details of the approach pursued and the literature and tools examined.

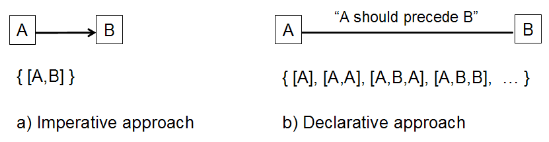

Fig. 1. Execution variances between imperative and declarative approaches.

conditions associated with them. A *declarative approach* focuses on *what* should be done instead of *how*. It uses constraints to restrict possible task execution options. By default all execution paths are allowed, i.e., allowing all executions that do not violate the constraints. In general, the more constraints are defined for a process, the less execution paths are possible, i.e., constraints limit process flexibility. In declarative languages, constraints are defined as relations between tasks. Mandatory constraints are strictly enforced, while optional constraints can be violated, if needed. Figure 1 provides an example of both approaches. For both of them, a set of possible execution paths are illustrated. Note that a declarative approach offers many more execution paths.

2.2 Flexibility Types in Detail

In this section we discuss the individual flexibility types. Each of them is described in detail using a standard format including: a motivation, definition, scope, realisation options, i.e., the situations and domains to which the flexibility type applies, an example and discussion.

Flexibility by Design

Motivation When a process operates in a dynamic environment it is desirable to incorporate support for the various execution alternatives that may arise within the process model. At runtime, the most appropriate execution path can be selected from those encoded in the design time process model.

Definition *Flexibility by Design* is the ability to incorporate alternative execution paths within a process model at design time allowing the selection of the most appropriate execution path to be made at runtime for each process instance.

Scope Flexibility by design applies to any process which may have more than one distinct execution trace.

Realisation options The most common options for realisation of flexibility by design are listed below. It is not the intention of the authors to give a complete overview of all options.

- parallelism – the ability to execute a set of tasks in parallel;
- choice – the ability to select one or more tasks for subsequent execution from a set of available tasks;

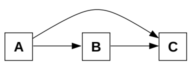

Fig. 2. Flexibility by design: a choice of execution paths is specified.

- iteration – the ability to repeatedly execute a task[2];
- interleaving – the ability to execute each of a set of tasks in any order such that no tasks execute concurrently;
- multiple instances – the ability to execute multiple concurrent instances of a task; and
- cancellation – the ability to withdraw a task from execution now or at any time in the future.

The notions above are thoroughly described by the workflow patterns [20] and have been widely observed in a variety of imperative languages. We argue that these concepts are equally applicable in a declarative setting which has a much broader repertoire of constraints that allow for flexibility by design. Note that both approaches really differ with respect to flexibility. To increase flexibility in an imperative process, more execution paths have to be modeled explicitly, whereas increasing flexibility in declarative processes is accomplished by reducing the number of constraints, or weakening existing constraints.

Example Figure 2 exemplifies a choice construct in an imperative model. The figure depicts that after executing A, it is possible to either execute B, followed by C, or to execute C directly. Using the choice construct, the notion of skipping tasks can be predefined in the process model.

Discussion Realisation options can be implemented differently in different ways. For example there are different variants of the choice construct, such as exclusive choice and deferred choice, which can be effected in different ways. Interested readers are referred to the workflow patterns [20].

Describing all possible execution paths in a process model completely at design-time may be either undesirable from the standpoint of model complexity or impossible due to an unknown or unlimited number of possible execution paths. The following three flexibility types provide alternative mechanisms for process flexibility.

Flexibility by Deviation

Motivation Some process instances need to temporarily deviate from the execution sequence described by the associated process model in order to accommodate changes in the operating environment encountered at runtime. For example, it may be appropriate to swap the ordering of the *register patient* and *perform*

[2] Note that iteration can be seen as a particular type of choice, where the join precedes the split

triage tasks in an emergency situation. The overall process model and its constituent tasks remain unchanged.

Definition *Flexibility by Deviation* is the ability for a process instance to deviate at runtime from the execution path prescribed by the original process without altering its process model. The deviation can only encompass changes to the execution sequence of tasks in the process for a specific process instance, it does not allow for changes in the process model or the tasks that it comprises.

Scope The concept of deviation is particularly suited to the specification of process models which are intended to guide possible sequences of execution rather than restrict the options that are available (i.e., they are descriptive rather than prescriptive). These specifications contain the preferred execution of the process, but other scenarios are also possible.

Realisation options The manner in which deviation is achieved depends on the specification approach utilised. Deviation can be seen as varying the actual tasks that will be executed next, from those that are implied by the current set of enabled tasks in the process instance. In imperative languages this can be achieved by applying deviation operations. For declarative approaches, deviation basically occurs through violation of optional constraints. The following set of operations characterise support for deviation by imperative languages:

- Undo *task A*: Moving control to the moment before the execution of *task A*. One point to consider with this operation is that it does not imply that the actions of the task are undone or reversed. This may be an issue if the task uses and changes data elements during the course of its execution. In such situations, it may also be desirable to roll-back or compensate for the consequences of executing the task in some way, although it is not always possible to do so, e.g., the effects of sending a letter can not be reversed.
- Redo *task A*: Executing a disabled, but previously executed *task A* again, without moving control. This operation provides the ability to repeat a preceding task. One possible use for the operation is to allow incorrectly entered data during task execution to be entered again. For example after registering a patient in a hospital and undertaking some examinations, the registration task can be repeated to adjust outdated or incorrect data. Note that updating registration data should not require medical examinations to be performed again.
- Skip *task A*: Passing the point of control to a task subsequent to an enabled *task A*. There is no mechanism to compensate for the skipped task by executing it at a later stage of the execution. This operation is useful for situations, where a (knowledgeable) user decides that it is necessary to continue execution, even though some preceding actions have not been performed. For example, in life threatening situations it should be possible to start surgery immediately, whereas normally the patient's health status is evaluated before commencing surgery.
- Create additional instance of *task A*: Creating an additional instance of a task that will run in parallel with those instances created at the moment

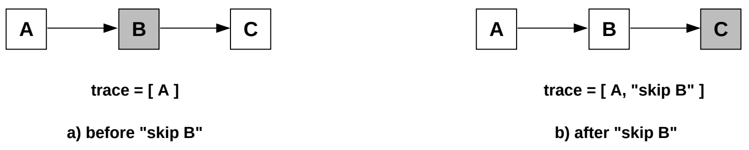

Fig. 3. Flexibility by deviation: the point of control is moved.

of task instantiation. It should be possible to limit the maximal number of task instances running in parallel. For example, a travel agency making trip arrangements for a group of people has to do the same arrangements if the number of travelling people increase (i.e., a separate reservation has to be done for each person).

– Invoke *task A*: Allows a task in the process model that is not currently enabled, and has not yet been executed, to be initiated. This task is initiated immediately. For example, when reviewing an insurance claim, it is suspected that the information given may be fraudulent. In order to determine how to proceed, the next task to be executed is deferred and a detailed investigation task (which normally occurs later in the process) is invoked. The execution of the investigation task does not affect the thread of control in the process instance and upon completion of the invoked task, execution continues from this point. Should the thread of control reach a previously invoked task at a later time in a process instance, it may be executed again or skipped on a discretionary basis.

Note that although we define deviation operations for imperative approaches only, this does not mean that there is no notion of these deviations in declarative approaches. Consider for example constraint *"A precedes B"*, which is defined as an optional constraint. By executing B before any occurrence of A, A is actually skipped by violating the optional precedence constraint. In this paper we clearly make a distinction between deviation for imperative and declarative approaches, due to the subtle difference in the act of deviating. Providing a full mapping of deviation operations to declarative constraints is beyond the scope of this paper.

Example Figure 3 exemplifies flexibility by deviation by applying a skip operation. In Figure 3(a) task B is enabled. After applying *skip B* (Figure 3(b)), it is possible to execute a (currently not enabled) successor of an enabled task B.

Discussion Deviation operations can be implemented in different ways, but for process mining purposes it should be possible to identify where deviations occurred during process execution. Furthermore additional requirements for the operators can be given, e.g., the *"undo A"* operation only has any effect when task A has been executed previously. When undoing task A, it may be recorded in one of two possible ways in the execution trace: either the undo task is explicitly marked as an execution action, or the occurrence of task A being undone is removed from the trace.

Flexibility by Underspecification

Motivation When specifying a process model it might be foreseen that in the future, during run-time execution, more execution paths are needed which must be incorporated within the existing process model. Furthermore, often only during the execution of a process instance does it become clear what needs to be done at a specific point in the process. When all execution paths cannot be defined in advance, it is useful to be able to execute an incomplete process model and dynamically add process fragments expressing missing scenarios to it.

Definition *Flexibility by Underspecification* is the ability to execute an incomplete process model at run-time, i.e., one which does not contain sufficient information to allow it to be executed to completion. Note that this type of flexibility does not require the model to be changed at run-time, instead the model needs to be completed by providing a concrete realisation for the undefined parts.

Scope The concept of underspecification is mostly suitable for processes where it is clearly known in advance that the process model will have to be adjusted at *specific points*, although the exact content at this point is not yet known (and may not be known until the time that an instance of the process is executed). This approach to process design and enactment is particularly useful where distinct parts of an overall process are designed and controlled by different work groups, but the overall structure of the process is fixed. In this situation, it allows each of them to retain some degree of autonomy in regard to the tasks that are actually executed at runtime in their respective parts of the process, whilst still complying with the overall process model.

Realisation options An incomplete process model is deemed to be one which is well-formed but does not have a detailed definition of the ultimate realisation of every task. An incomplete process model contains one or more so-called *placeholders*. Placeholders are nodes which are marked as *underspecified* (i.e., their content is unknown) and whose content is specified during the execution of these placeholders. We distinguish two types of *placeholder enactment*:

- *Late binding*: where the realisation of a placeholder is selected from a set of available process fragments. Note that to realise a placeholder one process fragment has to be selected from an existing set of predefined process fragments. This approach is limited to selection, and does not allow a new process fragment to be constructed.
- *Late modelling*: where a new process fragment is constructed in order to realise a given placeholder. Not only can a process fragment be constructed from a set of currently available process fragments, but also a new process fragment can be developed from scratch[3]. Therefore late binding is encompassed by late modelling. Some approaches [21] limit the construction of new models by (declarative) constraints.

For both approaches, the realisation of a placeholder can occur at a number of distinct times during process execution. Here, two distinct *moments for realisation* are recognised:

[3] However, this can only be done by highly skilled persons

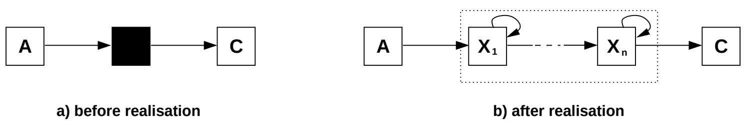

Fig. 4. Flexibility by underspecification: realisation of a placeholder.

- *before placeholder execution*: the placeholder is realised at commencement of a process instance or during execution before the placeholder has been executed for the first time.
- *at placeholder execution*: the placeholder is realised when it is executed.

Placeholders can be either realised once, or for every subsequent execution of the placeholder. We distinguish two distinct *realisation types*:

- *static realisation*, where the process fragment chosen to realise the placeholder during the first execution is used to realise the placeholder for every subsequent execution.
- *dynamic realisation*, where the realisation of a placeholder can be chosen again for every subsequent execution of the placeholder.

Example Figure 4(a) shows an incomplete process model with a placeholder task between A and C. Figure 4(b) illustrates the realisation of the placeholder, by a process fragment from a linked repository of process fragments. This figure shows the realisation as a sequence of self-looping tasks, but it can be realised by any well-formed process fragment.

Discussion The process fragments available for placeholder realisation can be stored in a so called repository. A repository can be available for one or more processes, just for a particular task or a set of tasks.

Flexibility by Change

Motivation In some cases, events may occur during process execution that were not foreseen during process design. Sometimes these events cannot be addressed by temporary deviations from the existing process model, but require the addition or removal of tasks or links from the process model on a permanent basis. This may necessitate changes to the process model for one or several process instances; or where the extent of the change is more significant, it may be necessary to change the process model for all currently executing instances.

Definition *Flexibility by Change* is the ability to modify a process model at runtime such that one or all of the currently executing process instances are migrated to a new process model. Unlike the previously mentioned flexibility types the model constructed at design time is modified and one or more instances need to be transferred from the old to the new model.

Scope Flexibility by change allows processes to adapt to changes that are identified in the operating environment. Changes may be introduced both at the level of the process instance and also at that of the process model (also known as change at instance level, and type or model level respectively).

Realisation options For flexibility by change we distinguish the following variation points, which are partly based on [2].

Effect of change defines whether changes are performed on the level of a process instance or on the level of the process model, and what the impact of the change on the new process instances is.

- *Momentary change*: a change affecting the execution of one or more selected process instances. The change performed on a given process instance does not affect any future instances.
- *Evolutionary change*: a change caused by modification of the process model, affecting all new process instances.

Moment of allowed change specifies the moment at which changes can be introduced in a given process instance or a process model.

- *Entry time*: changes can be performed only at the moment the process instance is created. After the process instance has been created, no further changes can be introduced to the given process instance. Momentary changes performed at entry time affect only a given process instance. The result of evolutionary changes performed at entry time is that all new process instances have to be started after the change of the process model has been performed, and no existing process instances are affected (they continue execution according to the process model with which they are associated).
- *On-the-fly*: changes can be performed at any time during process execution. Momentary changes performed on-the-fly correspond to customisation of a given process instance during its execution. Evolutionary changes performed on-the-fly impact both existing and new process instances. The new process instances are created according to the new process model, while the existing process instances may need to migrate from the existing process model to the new process model.

Migration strategy defines what to do with running process instances that are impacted by an evolutionary change.

- *Forward recovery*: affected process instances are aborted.
- *Backward recovery*: affected process instances are aborted (compensated if necessary) and restarted.
- *Proceed*: changes introduced are ignored by the existing process instances. Existing process instances are handled the old way, and new process instances are handled the new way.
- *Transfer*: the existing process instances are transferred to a corresponding state in the new process model.

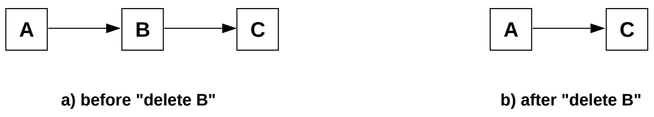

Fig. 5. Flexibility by change: a task is deleted.

Example In Figure 5(a) we show a process model that is changed into the process model depicted in Figure 5(b) by removing task B. The effect of this change is that instances of the new process model will skip task B permanently.

Discussion A very detailed description of change operations can be found in [24]. The authors propose using high level change patterns rather than low level change primitives and give full descriptions for the identified patterns. Based on these change patterns and features, they provide a detailed analysis and evaluation of selected systems from both academia and industry.

3 Evaluation of Contemporary Offerings

In this section, we apply the taxonomy presented in Section 2 to evaluate a selection of PAISs, namely, ADEPT1 [17], YAWL[4] (version 8.2b) [1, 5, 4], FLOWer (version 3.0) [3] and Declare (version 1.0) [11, 12]. This evaluation provides an insight into the manner and extent to which the individual flexibility types are actually implemented in practice. The selection of PAISs has been based on the criterion of supporting process flexibility, which excludes classical workflow systems and most commercial systems. Moreover, the selected systems cover distinct areas of the PAIS technology spectrum, such as adaptive workflow (ADEPT1), case handling (FLOWer) and declarative workflow (Declare).

Although we focus on the evaluation of flexibility support, it is worthwhile mentioning that there is a huge difference in the maturity of the selected offerings. FLOWer has been widely used in industry where its flexible approach to case handling has proven to be extremely effective for a variety of purposes (e.g., insurance claim handling). ADEPT1 has also been successfully applied in different areas, such as health care [17]. Throughout its development lifetime, the designers of ADEPT [16] have focussed on supporting various forms of change [16, 19, 24]. The intention of the next version (ADEPT2) is to provide full support for changes, including transfer. YAWL is a more recent initiative based on formal foundations that shows significant promise in the support of a number of distinct flexibility approaches. Declare is the most recent of the offerings examined and its declarative basis provides a number of flexibility features. Interestingly, it supports transfer of existing process instances to the new process model. In the declarative setting, transfer is easily supported because it is not necessary to find a matching state in the new process for each instance [12].

The evaluation results are depicted in Table 1, which shows whether a system provides full (+), partial (+/−) or no support (−) for an evaluation criterion. For

[4] The evaluation of YAWL includes the so-called Worklet Service.

Table 1. Product evaluations

	ADEPT1	**YAWL**	**FLOWer**	**Declare**
Flexibility by design				
Parallelism	+	+	+	+
Choice	+	+	+	+
Iteration	+	+	+	+
Interleaving	−	+	+/−	+
Multiple instances	−	+	+	+
Cancellation	−	+	−	+
Flexibility by deviation				
Deviation operations (imperative languages)				
Undo	−	−	+	
Redo	−	−	+	
Skip	−	−	+	
Create additional instance	−	−	+/−	
Invoke task	−	−	+	
Deviation operations (declarative languages)				
Violation of constraints				+
Flexibility by underspecification				
Late binding	−	+	−	−
Late modelling	−	+	−	−
Static, before placeholder	−	−	−	−
Dynamic, before placeholder	−	−	−	−
Static, at placeholder	−	−	−	−
Dynamic, at placeholder	−	+	−	−
Flexibility by change				
Effect of change				
Momentary change	+	−	−	+
Evolutionary change	−	+	−	+
Moment of allowed change				
Entry time	+	−	−	+
On–the–fly	+	+	−	+
Migration strategies for evolutionary change				
Forward recovery	−	+	−	−
Backward recovery	−	+	−	−
Proceed	−	−	−	+
Transfer	−	+	−	+

the full description of the evaluation criteria and the detailed evaluations for each of the offerings, we refer readers to the associated technical report [10]. The remainder of this section discusses the evaluation results.

Flexibility by design can be provided in several ways. Parallelism, choice and iteration are fully supported by all systems. Interleaving, multiple instances and cancellation are not supported by all systems, but they are all supported by YAWL and Declare, although in different ways. Due to the nature of declarative languages, the designer is encouraged to leave choices to users at run-time.

Flexibility by deviation is similarly supported by both FLOWer and Declare despite their distinct conceptual foundations. FLOWer achieves this by supporting almost all of the deviation operations, whereas Declare allows for violation of optional constraints. *Flexibility by underspecification* is only supported by YAWL (through its Worklet service). *Flexibility by change* is supported by ADEPT1, YAWL and Declare. ADEPT1 supports momentary change, which is allowed both at entry-time and on-the-fly. As mentioned earlier, the ADEPT developers have undertaken comprehensive research into the issue of dynamic process change and it will be interesting to see this incorporated in the next release (ADEPT2) when it becomes available. Evolutionary change is supported by YAWL and Declare, but unlike Declare, YAWL only supports changes to the process model. For this reason, YAWL does not support momentary change, entry time change and a proceed strategy. Declare supports changes for process instances and for the process model and offers proceed and transfer strategies. Transfer will be applied to those instances for which the execution trace does not violate the new process model, otherwise the proceed strategy will be applied, see [12] for details.

None of the evaluated systems provides the full range of flexibility alternatives. YAWL focusses on providing flexibility by design and underspecification, ADEPT1 on flexibility by change, FLOWer on flexibility by deviation and Declare provides flexibility in several different areas: design, deviation, and change.

4 Related work

The need for process flexibility has long been recognised [8, 18] in the workflow and process technology communities as a critical quality of effective business processes in order for organisations to adapt to changing business circumstances. It ensures that the "fit" between actual business processes and the technologies that support them are maintained in changing environments. The notion of flexibility is often viewed in terms of the ability of an organisation's processes and supporting technologies to adapt to these changes [22, 7]. An alternate view advanced by Regev and Wegmann [13] is that flexibility should be considered from the opposite perspective i.e., in terms of what stays the same not what changes. Indeed, a process can only be considered to be flexible if it is possible to change it without needing to replace it completely [14]. Hence flexibility is effectively a balance between change and stability that ensures that the identity of the process is retained [13].

There have been a series of proposals for classifying flexibility, both in terms of the factors which motivate it and the ways in which it can be achieved within business processes. Snowdon et al. [22] identify three causal factors: type flexibility (arising from the diversity of information being handled), volume flexibility (arising from the amount of information types) and structural flexibility (arising from the need to operate in different ways). Soffer [23] differentiates between short-term flexibility, which involves a temporary deviation from the standard way of working, and long-term flexibility, which involves changes to the usual way of working. Kumar and Narasipuram [9] distinguish pre-designed flexibility which is anticipated by the designer and forms part of the process definition and just-in-time responsive flexibility which requires an “intelligent process manager” to deal with the variation as it arises at runtime. Carlsen et al. [6] identify a series of desirable flexibility features for workflow systems based on an examination of five workflow offerings using a quality evaluation framework. Heinl et al. [8] propose a classification scheme with distinct approaches – flexibility by selection, where a variety of alternative execution paths are designed into a process, and flexibility by adaption, where a workflow is “adapted” (i.e., modified) to meet with the new requirements. Two distinct approaches to achieving each of these approaches are recognised: flexibility by selection can be implemented either by advance modelling (before execution time) or late modelling (during execution time) where as flexibility by adaption can be handled either by type or instance adaption. Van der Aalst and Jablonski [2] adopt a similar strategy for supporting flexibility. Moreover they propose a scheme for classifying workflow changes in detail based on six criteria: (1) reason for change, (2) effect of change, (3) perspectives affected, (4) kind of change, (5) when are changes allowed and (6) what to do with existing process instances. Regev et al. [14] provide an initial attempt at defining a taxonomy of the concepts relevant to business process flexibility. This taxonomy has three orthogonal dimensions: the abstraction level of the change, the subject of the change and the properties of the change. Whilst it incorporates elements of the research initiatives discussed above, it is not comprehensive in form and does not describe the relationships that exist between these concepts or link them to possible realisation approaches.

The individual flexibility types discussed in this paper are informed by a multitude of research initiatives in the workflow and BPM fields. It is not possible to discuss these in detail in the confines of this paper, however there is a detailed literature review of the area in [10].

5 Conclusion

In this paper we have proposed a comprehensive taxonomy of flexibility approaches achieved based on an extensive survey of contemporary offerings and literature in the field. The four distinct flexibility types, that make up the proposed taxonomy, differ with respect to the moment and the manner in which both foreseen and unforeseen behaviour can be handled in a process. On the basis of this taxonomy, we have evaluated the support for flexibility provided by various commercial and research offerings (these were the only offerings that demonstrated any sort of flexibility features).

The evaluation process revealed varying approaches to flexibility support in the selected offerings. Moreover, individual offerings tended to exhibit a degree of specialisation in their approach to process flexibility. Such a strict specialisation limits the use of offerings in practice, since they are not capable of accommodating foreseen and unforeseen behaviours in processes during different phases of the BPM cycle. We hope that insights provided in this paper might trigger the enhancement of existing tools and/or development of new ones with a view for providing a greater support for flexibility.

A logical future step in researching the process flexibility is the establishment of a formal taxonomy/ontology for process flexibility, which would allow realisations of each of the flexibility types to be compared in an objective and language-independent way. Furthermore, another interesting line of research is to assess the extent of flexibility that current processes demand with a view to determining which of the flexibility approaches are of most use in practice.

In this paper we concentrated on the control-flow perspective of a business process, other perspectives addressing data, resources and applications used in a process are also subject to change. Thus, it would be worthwhile to extend the taxonomy in order to incorporate these perspectives. Additionally, there are some interesting process mining challenges presented by systems that support deviation or change operations, as in these offerings there is the potential for individual process instances to execute distinct process models.

References

1. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.
2. W.M.P. van der Aalst and S. Jablonski. Dealing with workflow change: Identification of issues and solutions. *International Journal of Computer Systems, Science, and Engineering*, 15(5):267–276, 2000.
3. W.M.P. van der Aalst, M. Weske, and D. Grünbauer. Case Handling: A New Paradigm for Business Process Support. *Data and Knowledge Engineering*, 53(2):129–162, 2005.
4. M. Adams, A.H.M. ter Hofstede, W.M.P. van der Aalst, and D. Edmond. Dynamic, Extensible and Context-Aware Exception Handling for Workflows. In F. Curbera, F. Leymann, and M. Weske, editors, *Proceedings of the OTM Conference on Cooperative information Systems (CoopIS 2007)*, volume 4803 of *Lecture Notes in Computer Science*, pages 95–112. Springer-Verlag, Berlin, 2007.
5. M. Adams, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In R. Meersman and Z. Tari et al., editors, *Proceeding of the OTM Conference on Cooperative Information Systems (CoopIS 2006)*, volume 4275 of *Lecture Notes in Computer Science*, pages 291–308. Springer-Verlag, Berlin, 2006.
6. S. Carlsen, J. Krogstie, A. Slvberg, and O.I. Lindland. Evaluating flexible workflow systems. In *Proceedings of the Thirtieth Hawaii International Conference on System Sciences (HICSS-30)*, Maui, Hawaii, 1997. IEEE Computer Society Press.
7. F. Daoudi and S. Nurcan. A benchmarking framework for methods to design flexible business processes. *Software Process Improvement and Practice*, 12:51–63, 2007.
8. P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein, and M. Teschke. A comprehensive approach to flexibility in workflow management systems. In *WACC '99: Proceedings of the international joint conference on Work activities coordination and collaboration*, pages 79–88, New York, NY, USA, 1999. ACM.
9. K. Kumar and M. M. Narasipuram. Defining requirements for business process flexibility. In *Workshop on Business Process Modeling, Design and Support (BP-MDS06), Proceedings of CAiSE06 Workshops*, pages 137–148, 2006.

10. N.A. Mulyar, M.H. Schonenberg, R.S. Mans, N.C. Russell and W.M.P. van der Aalst. Towards a Taxonomy of Process Flexibility (Extended Version). BPM Center Report BPM-07-11, BPMcenter.org, 2007.
11. M Pesic and W.M.P. van der Aalst. A declarative approach for flexible business processes management. In *Business Process Management Workshops*, pages 169–180, 2006.
12. M. Pesic, M. H. Schonenberg, N. Sidorova, and W.M.P. van der Aalst. Constraint-Based Workflow Models: Change Made Easy. In F. Curbera, F. Leymann, and M. Weske, editors, *Proceedings of the OTM Conference on Cooperative information Systems (CoopIS 2007)*, volume 4803 of *Lecture Notes in Computer Science*, pages 77–94. Springer-Verlag, Berlin, 2007.
13. G. Regev, I. Bider, and A. Wegmann. Defining business process flexibility with the help of invariants. *Software Process Improvement and Practice*, 12:65–79, 2007.
14. G. Regev, P. Soffer, and R. Schmidt. Taxonomy of flexibility in business processes. In *Proceedings of the 7th Workshop on Business Process Modelling, Development and Support(BPMDS'06)*, 2006.
15. G. Regev and A. Wegmann. A regulation-based view on business process and supporting system flexibility. In *Workshop on Business Process Modeling, Design and Support (BPMDS05), Proceedings of CAiSE05 Workshops*, pages 35–42, 2005.
16. M. Reichert and P. Dadam. ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
17. Manfred Reichert, Stefanie Rinderle, and Peter Dadam. Adept workflow management system. In Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Mathias Weske, editors, *Business Process Management, International Conference, BPM 2003, Eindhoven, The Netherlands, June 26-27, 2003, Proceedings*, volume 2678 of *Lecture Notes in Computer Science*, pages 370–379. Springer, 2003.
18. H.A. Reijers. Workflow flexibility: The forlorn promise. In *15th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2006), 26-28 June 2006, Manchester, United Kingdom*, pages 271–272. IEEE Computer Society, 2006.
19. S. Rinderle, M. Reichert, and P. Dadam. Correctness Criteria For Dynamic Changes in Workflow Systems: A Survey. *Data and Knowledge Engineering*, 50(1):9–34, 2004.
20. N. Russell, A.H.M. ter Hofstede, W.M.P. van der Aalst, and N. Mulyar. Workflow control-flow patterns: A revised view. Technical Report BPM-06-22, 2006. `http://www.BPMcenter.org`.
21. S.W. Sadiq, W. Sadiq, and M.E. Orlowska. Pockets of flexibility in workflow specification. In *ER '01: Proceedings of the 20th International Conference on Conceptual Modeling*, pages 513–526, London, UK, 2001. Springer-Verlag.
22. R.A. Snowdon, B.C. Warboys, R.M. Greenwood, C.P. Holland, P.J. Kawalek, and D.R. Shaw. On the architecture and form of flexible process support. *Software Process Improvement and Practice*, 12:21–34, 2007.
23. P. Soffer. On the notion of flexibility in business processes. In *Workshop on Business Process Modeling, Design and Support (BPMDS05), Proceedings of CAiSE05 Workshops*, pages 35–42, 2005.
24. B. Weber, S.B. Rinderle, and M.U. Reichert. Change support in process-aware information systems - a pattern-based analysis. Technical Report Technical Report TR-CTIT-07-76, ISSN 1381-3625, Centre for Telematics and Information Technology, University of Twente, Enschede, 2007. `http://eprints.eemcs.utwente.nl/11331/`.