Towards a Taxonomy of Process Flexibility (Extended Version)

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. Effective business processes must be able to accommodate changes to the environment in which they operate, e.g., new laws, changes in business strategy. The ability to encompass such changes is termed *process flexibility*. In this paper, we take a deeper look into the various ways in which flexibility can be achieved and propose a comprehensive taxonomy of these methods which identifies both the manner in which each of them is facilitated, and also the various configuration options and alternatives that exist in each case. This taxonomy is subsequently used to evaluate a selection of process-aware information systems and identify their potential to deploy flexible business processes.
Keywords: taxonomy, flexibility, design, change, deviation, underspecification.

1 Introduction

Business Process Management (BPM) has become an area of intense focus in recent years as it is increasingly recognised that the notion of the business process is a key concept underpinning organisational activities and that these processes need managing in the same way as other more tangible corporate assets.

Typically the enactment of business processes is facilitated using some form of process technology. There is a broad range of potential offerings on which these processes can be based, ranging from workflow systems which rigidly enforce adherence to the underlying process definition to groupware systems which are guided by an implied process definition but do nothing to ensure that it is actually adhered to. The broad range of technologies that are founded on some form of underlying process definition are termed *Process-Aware Information Systems* or PAISs.

A key consideration of effective processes is their ability to deal with both foreseen and unforeseen changes in the context or environment in which they operate. This quality of a process – termed *flexibility* – reflects its ability to deal with such 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. Indeed, as has been noted [50], flexibility is

as much about what should stay the same in a process as what should be allowed to change.

Whilst the notion of flexibility is relatively simple at a conceptual level, it is somewhat more difficult to achieve in practice. PAISs can only capture an abstraction of the business process that they facilitate, and recognition that there is a deviation between this process definition and the "real-life" process that they are intended to support requires external (typically human) input. Moreover, once a variation is identified between the expected and actual process enactment some means of minimising this "gap" is required in order to ensure that the process proceeds in accordance with expectations.

In this paper, we identify a range of approaches for achieving process flexibility. We describe these approaches in the form of a taxonomy, with which we intend to provide a *comprehensive catalogue for the control-flow perspective of process flexibility*. As each of these approaches can be implemented in a number of distinct ways, a series of realisation options are identified that allow individual flexibility approaches to be tailored to specific needs.

The remainder of this paper proceeds as follows. Section 2 presents a detailed description of the taxonomy for process flexibility. Section 3 uses this taxonomy to evaluate the capabilities of a number of contemporary PAIS and discusses the evaluation results. Section 4 discusses related work. The conclusion and future work are presented in Section 5.

2 Taxonomy of Flexibility

In this section, we present a comprehensive description of the various 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 the complete redesign of the underlying process specification, however they differ in the timing and manner in which they are effected.

2.1 Overview of Flexibility Types

Four distinct approaches to process flexibility are delineated in the taxonomy, each having its own application area. We distinguish **flexibility by**

design: for handling anticipated changes in the operating environment, where supporting strategies can be defined at design-time.

deviation: for handling occasional unforeseen behaviour, where differences with the expected behaviour are minimal.

underspecification: for handling anticipated changes in the operating environment, where strategies cannot be defined at design-time, because the final strategy is not known in advance or is not generally applicable.

change: either for handling occasional unforeseen behaviour, where differences require process adaptations, or for handling permanent unforeseen behaviour.

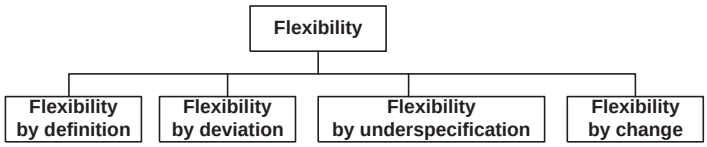

Fig. 1. Flexibility types

Precise definitions of each flexibility type can be found in the remainder of this section. The flexibility types are presented in the form of a taxonomy (see Figure 1) which aims to define each of them in detail, providing a precise description of the manner in which they operate, the concepts relevant to each of them and details of the various configuration options (termed variation points) which vary the manner in which they function.

2.2 Illustrating Flexible Behaviour

Throughout this paper, we illustrate the operational effects of each flexibility type using the concept of *traces*. A trace is simply a possible execution sequence in a process instance, e.g., $[A, B, C]$. Typically what is of interest when illustrating specific flexibility types is the range of possible execution sequences. This is usually denoted in terms of a set of traces (e.g., $\{[A, B, D], [A, C, D]\}$). Generally the set of traces depends on the structure of a process as well as the specification approach used to describe control-flow within the process.

2.3 Specification Approaches

Generally, process behaviour depends on the structure of a process, which can be defined in an imperative or a declarative way.

Imperative approach 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 conditions associated with them.

Declarative approach 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, as more constraints are defined for a process, less execution paths are possible, i.e., constraints limit the flexibility. In declarative languages, constraints are defined as relations between tasks. Mandatory constraints have to be strictly enforced, while optional constraints can be violated, if needed.

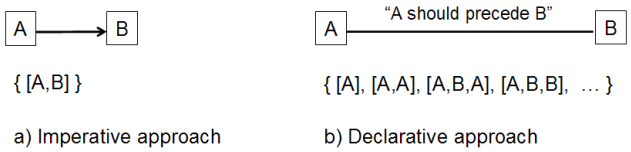

Fig. 2. Specification types.

Figure 2 provides an example of both approaches. For both of them, a set of possible execution paths are illustrated. Note that these approaches really differ with respect to flexibility. To increase flexibility in an imperative process, more execution paths have to be modelled explicitly, whereas increasing the flexibility in declarative processes is accomplished by reducing the number of constraints, or weakening the existing constraints. A declarative model is most flexible when it does not have any constraints. In this case, all its tasks can be executed in any order, any number of times.

2.4 Flexibility Types in Detail

We now move on to a detailed discussion of each of the individual flexibility types. Each of these is described in detail using a standard format which includes the following items:

- *motivation* – the rationale for the flexibility type;
- *definition* – a concise description of the flexibility type;
- *scope* – the situations and domains to which the flexibility type applies;
- *realisation options* – a set of attributes and/or operational criteria which characterise the manner in which the flexibility type can be effected;
- *example* – an illustrative example of its usage; and
- *discussion* – problems that could be encountered when implementing the flexibility type and recommendations on how these problems might be addressed.

Flexibility by Design

Motivation Where a process operates in a varying operational environment it is desirable to incorporate support for the various known execution alternatives within the process definition. At runtime, the most appropriate execution path can be selected from those encoded in the design time process definition.

Definition *Flexibility by Design* is the ability to incorporate alternative execution paths within a process definition at design time such that selection of the most appropriate execution path can 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.

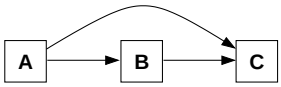

Fig. 3. Example of flexibility by design.

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;
- iteration – the ability to repeatedly execute a task[1];
- 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 in the workflow patterns [57] 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. Furthermore, in declarative approaches the design of flexible processes is stimulated. In contrast to imperative approaches, the designer does not have to explicitly include the concepts in the model, i.e., putting effort to include flexibility to the design. A declarative model without constraints captures all these concepts and designers include constraints to limit its flexibility.

Example Figure 3 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 definition.

Discussion Realisation options can be implemented differently in various available tools. For example there are different variants of choice construct, such as exclusive choice and deferred choice, which can be realised in different ways. Interested readers are referred to the workflow patterns [57].

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

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

Flexibility by Deviation

Motivation Some process instances need to temporarily deviate from the execution sequence described by the associated process definition 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 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 definition. The deviation can only encompass changes to the execution sequence of tasks in the process model for a specific process instance, it does not allow for changes in the process definition or the tasks that it comprises.

Scope The concept of deviation is particularly suited to the specification of process definitions 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 by violation of optional constraints. The following set of operations characterise the support of deviation by imperative languages:

- Undo *task A*: Shifting 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., it is not possible to reverse the effects of sending a letter.
- Redo *task A*: Executing disabled, but previously executed *task A* again, without shifting 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 of 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

Fig. 4. Example of flexibility by deviation.

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 process instances created on the moment of task instantiation. To control the flexibility, 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 people travelling increases (i.e. a separate reservation has to be done for each person).
- Invoke *task A*: Allows a task in the process definition 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 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 4 exemplifies flexibility by deviation by applying a skip operation. In Figure 4(a) task B is enabled. After applying *skip B* (Figure 4(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 nevertheless it should be possible to identify which deviations have been made during process execution. Furthermore additional requirements for the operators

can be given, e.g., the *"undo A"* operation only has 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 definition, it might be foreseen that in the future, during run-time execution, more execution paths will be needed which have to be handled in accordance with the existing process definition. Furthermore, often only during the execution of a process instance it becomes clear what needs to be done at some specific point in the process. Where all execution paths cannot be defined in advance, it is useful to be able to execute an incomplete process definition and dynamically add process fragments expressing missing scenarios to it.

Definition *Flexibility by underspecification* is the ability to execute an incomplete process specification 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 definition 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 definition.

Realisation options An incomplete process definition is deemed to be one which is well-formed but does not have a detailed definition of the ultimate realization of every task. An incomplete process specification 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 fully 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 more complex process fragment be

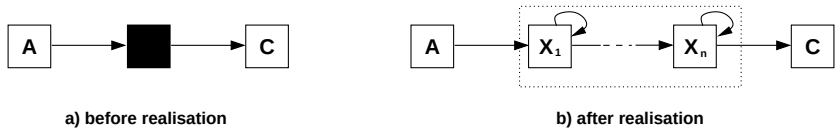

Fig. 5. Example of flexibility by underspecification.

constructed from a set of currently available process fragments, but also a new process fragment can be developed from scratch[2]. Therefore late binding is encompassed by late modelling. Some approaches [60] 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:

- *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 the placeholder is executed.

Placeholders can be either realised once, or be realised 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 this placeholder.

Example Figure 5(a) shows an incomplete process definition with a placeholder task between A and C. Figure 5(b) illustrates the realisation of the placeholder, by a process fragment from a linked repository of process fragments. In this figure the realisation is done by 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 for 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

[2] However, this should only be done highly by skilled persons

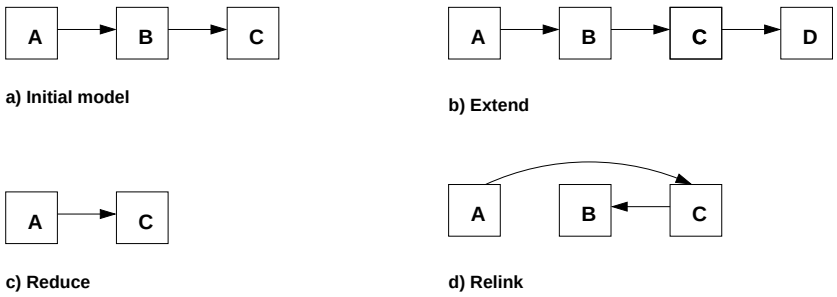

Fig. 6. Process definition changes.

by temporary deviations from the existing process definition, but require the addition or removal of tasks or links from the process definition 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 and also those that will execute in the future. The process definition can be changed by extending or by reducing the activities in the definition, or by relinking the activities, as depicted in Figure 6.

Definition *Flexibility by Change* is the ability to modify a process definition at run-time such that one or all of the currently executing process instances are migrated to a new process definition. Unlike the previous three 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 process instance and process type levels.

Realisation options For flexibility by change we distinguish the following variation points, which are partly based on [6].

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

- *Momentary change* (also known as change at instance level): 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* (also known as change at type level): a change caused by modification of the process definition, 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 definition.

Fig. 7. Example of flexibility by change.

- *Entry time*: changes can be performed only at the moment the process instance is created. After the process instance has been created, no changes can be introduced to the given process instance any more. 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 definition has been performed, and no existing process instances are affected (they continue execution according to the associated process definition).
- *On-the-fly*: changes can be performed at any point in time during process execution. Momentary changes performed on-the-fly correspond to customization 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 description, while the existing process instances may need to migrate from the existing process definition to the new process definition.

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 definition.

Example In Figure 7(a) we show a process definition that at some moment is changed into the process definition depicted in Figure 7(b) by removing task B. The effect of this change is that instances of the new process definition are skipping task B permanently.

Discussion A very detailed description on change operations can be found in [69]. 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.

2.5 Summary of Flexibility Types

Each of the flexibility types operates in different ways. Figure 8 provides an illustration of the distinction between each of the flexibility types in isolation, in terms of the time at which the specific flexibility options need to be configured – at design time, as part of the process definition or at runtime via facilities in the process execution environment. It also shows the anticipated completeness of the process definition for each flexibility type.

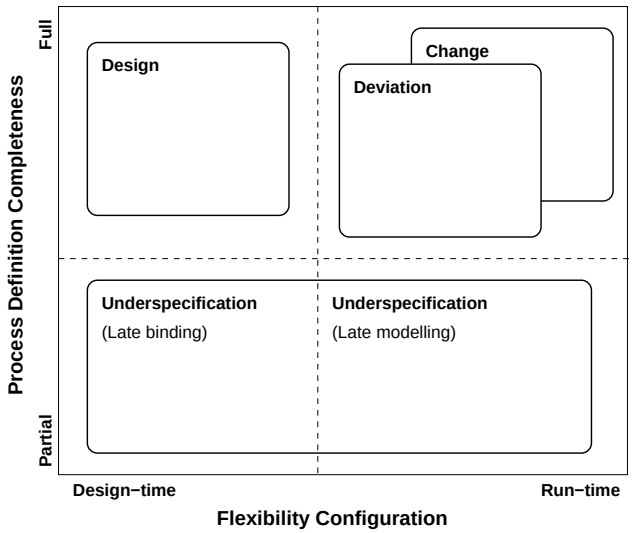

Fig. 8. Flexibility type spectrum

Flexibility by underspecification works on the basis of an incomplete process definition. Combined with late binding only, it just offers design-time configuration options, i.e., only the fragments that have been defined during design-time can be selected at run-time. Whereas, combined with late modelling, also run-time configuration options are offered by providing means to define and select fragments at run-time.

In the spectrum of options, *flexibility by design* distinguishes itself by being the flexibility type that works for full process definitions, whilst only being configurable at design-time. At run-time only predefined paths can be chosen.

Both *flexibility by deviation* and *change* work with complete process definitions. For both types, the configuration options are only available at run-time. Although very similar, only flexibility by change affects the process definition both at instance and type level, whereas flexibility by deviation does not affect the process definition at all.

3 Evaluation of Contemporary Offerings

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 optional 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				
Forward recovery	–	–	–	–
Backward recovery	–	–	–	–
Proceed	–	–	–	+
Transfer	–	–	–	+

In this section, we apply the taxonomy presented in Section 2 to evaluate a selection of PAISs. Respectively, ADEPT1 [53], YAWL[3] (version 8.2b) [5, 12, 10], FLOWer (version 3.0) [7] and Declare (version 1.0) [46, 47] will be evaluated.

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

These systems have been chosen as they allow for more flexibility than classical workflow systems and, in this way, they are interesting to evaluate. Moreover, they cover distinct areas of the PAIS technology spectrum, like adaptive workflow (ADEPT1), case handling (FLOWer) and declarative workflow (Declare). The detailed evaluations for each of the systems mentioned can be found in the appendix of this report, whereas Table 1 only shows whether a system receives full (+), partial (+/-) or no support (-) for an evaluation criterion.

Flexibility by design is 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, not reflected in Table 1, Declare offers more flexibility by design than the other systems. Due to the declarative nature of the language, the designer is enabled and stimulated to leave choices to users at run-time. *Flexibility by deviation* is mostly supported by 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 only supported by ADEPT1 and Declare. Although it is possible to upload new process definitions at run-time in YAWL and FLOWer, there is no system support and in the case of FLOWer, it is even highly dissuaded in the user guide [67], since it may lead to unforeseen events and even deadlocks. Declare is the only offering supporting both momentary and evolutionary change. The migration strategy for evolutionary change in Declare is to either transfer an instance to the new process definition, if possible, or otherwise, to proceed.

Although not reflected by Table 1, ADEPT1 is far superior to the other approaches in offering flexibility by change. Since the beginning the designers of ADEPT [52] have been focussing on supporting various forms of change [52, 56, 69]. The next version (ADEPT2) will provide full support for changes, including transfer. Compared to Declare, ADEPT1 is more mature and has been successfully applied in different areas, like health care [53]. Interestingly, Declare supports transfer of existing process instances to the new process model. In the declarative setting, transfer is easily supported because in this setting it is not necessary to find a matching state in the new process for each instance [47].

None of the evaluated systems provides the full range of flexibility. YAWL focusses on providing flexibility by design and underspecification (worklets), ADEPT1 on flexibility by change (adaptive workflow), FLOWer on flexibility by deviation (case handling) and Declare provides flexibility in different areas: design and deviation, and change.

4 Related work

The need for process flexibility has long been recognised [35, 54] 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 [50]. 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 [62, 22]. An alternate view advanced by Regev and Wegmann [48] 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 [49]. Hence flexibility is effectively a balance between change and stability that ensures that the identity of the process is retained [48, 51].

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. [62] 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 [63] 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 [41] 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. [16] 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. [35] 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 adaption (where the process definition is changed but individual process instances currently running are unaffected) or instance adaption where selected (or all) process instances are changed to meet with new operational requirements. Van der Aalst and Jablonski [6] 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. [49] 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 describe above, it is not comprehensive in form and does not describes the relationships that exist between these concepts or link them to possible realisation approaches.

There are a variety of approaches to incorporating flexibility within a design-time process definition. Traditional process design methods [54, 42, 15] have centered on the separation of business logic from the actual application processing and utilising constructs such as hierarchy, conditional elements and business rules within the process definition to explicitly cater for various execution scenarios that might be encountered. Whilst effective, these strategies require that all possible situations be captured a priori at design-time, an assumption that proves to be unrealistic in practice [35]. The use of exceptions [58, 65, 25] provides one means of handling expected but infrequently occurring processing errors without requiring their explicit inclusion in the process definition. Various techniques to implementing exception handling strategies in workflow systems have been demonstrated by offerings including WAMO [24], ConTracts [55], Exotica [14], OPERA [31, 32], TREX [64] and WIDE [17].

Another approach that has been investigated for embedding flexible constructs in business processes involve the augmentation of control-flow routing constructs operators based on fuzzy logic [9]. Indeed one area that offers significant opportunity for increasing the potential flexibility of a business process is the replacement of the strict graph-based structures that are generally used to describe control-flow dependencies between the tasks in a process with other means of describing these dependencies. ConDec [46, 47] is a declarative language that specifies control-flow dependencies using linear temporal logic expressions. CIGDec [44] proposes a similar strategy for enhancing flexibility when modelling and enacting clinical guidelines. Other research initiatives in this area have investigated a variety of other means of defining control-flow including the use of process grammars to specify dependencies between tasks and documents (i.e. data elements) in a process [28], the introduction of the notion of "anticipation" [29] which allows the execution of sequential tasks to overlap at the discretion of workflow users where there are not specific data dependencies between them, the inclusion of flexible elements in process definitions that describe alternate execution options, alternate task orderings and optional tasks [40] and basing control-flow on rule-based invariants that must hold during process execution [48] or constraints based on task pre and postconditions [66] that determine when individual tasks can start and complete. In [23], Dustdar examines the issue from another perspective and investigates the fundamental aspects of process-aware collaboration and the capabilities that are required from a technological solution for supporting team-based business processes. These are illustrated in the context of the Caramba system.

The potential for increasing process flexibility by allowing deviations from the specified process definition at runtime is supported in PROSYT [21] which allows a deviation policy to be specified for a process, identifying which forms of deviation are tolerated, together with a consistency handling policy, which ensures any allowed deviations do not impact the overall correctness of the system. In the context of the WASA system, Weske [21] nominates three user-initiated operations – SkipActivity, StopActivity and RepeatActivity – that allow for deviations from normal workflow execution.

Several approaches have been proposed that support the underspecification of processes thus allowing for greater flexibility in the actual tasks initiated at runtime. Noll [45] advocates the use of low fidelity models which specify the major tasks and main sequence in a process, but leave the actual sequence of execution at the discretion of the user. This essentially corresponds to a more general notion of the case handling paradigm [8] as it allows distinct tasks in a given process instance to be undertaken by differing users. In a similar vein, Herrmann and Loser [36] advocate the inclusion of "vagueness" in socio-technical process definitions allowing concepts such as arc conditions and task ordering to be deliberately omitted from models and also supporting the inclusion of specific modelling constructs to identify aspects of the model that are incomplete or unspecified. Van der Aalst advances the notion of generic process definitions [1, 2] which allow placeholders elements (termed generic processes) to be specified in models which correspond to fragments of the overall process whose actual composition is determined at runtime. Mangan and Sadiq [43] propose an analogous scheme where a process is partially specified as a set of fragments and the actual format of the process definition undertaken for a given instance of the process is deferred to runtime at the discretion of individual users. In a subsequent paper Sadiq et al. [60] describe a flexible workflow modelling language which incorporates "pockets of flexibility" which denote regions of the process who actual content is determined at runtime based on workflow fragments (tasks or subprocesses) and composition rules that are associated with them. The OPENflow system [33] is an example of an actual system that supports this approach to process flexibility. Adams et al [12] propose the notion of *worklets* which allow the implementation of tasks to be dynamically evolved by associating a distinct subworkflow implementation with each of them depending in the actual context that is encountered at runtime. A similar notion is advanced for exception handling in the form of *exlets* [10].

The issue of managing dynamic change to executing processes has been widely researched in the fields of adaptive and evolutionary workflow [38, 26, 18, 37, 71, 20, 61, 4, 59, 39]. A number of significant research prototypes have been developed in this area including ADEPTflex [52], ADOME [19], CBRFlow [70], DYNAMITE [34], MILANO [13], WASA2 [71] and YAWL worklets [11]. A comprehensive evaluation of various approaches (both conceptual and implementation-based) to managing dynamic changes to workflow processes is presented by Rinderle et al. in [56]. As a means of comparing various approaches to process change, Weber et al [68] have proposed a set of 17 change patterns and six change support features. In [27], Ellis and Keddara propose ML-DEWS, a modelling language for specifying changes in workflow systems.

One difficulty associated with managing dynamic change is avoiding the "dynamic change bug" [26] where the migration of individual process instances from the old process model introduces errors. Van der Aalst [3] proposes an approach for managing this issue by calculating the *safe change region* which a case must be in if a workflow change is to be successfully facilitated. Günter et al [30] have developed a framework for integrating adaptive process management and

process mining techniques which enables the exploitation of change information extracted from process change logs.

5 Conclusion

In this paper we have presented a taxonomy that integrates a broad spectrum of alternative approaches aimed at promoting process flexibility. Furthermore we evaluated offerings from four distinct areas of the PAIS technology spectrum. The result of these evaluations clearly identifies that different technologies adopt different approaches to promoting process flexibility.

Interestingly, none of the offerings examined provides support across all flexibility types. This suggests that each of these individual flexibility types seem to be particularly suited to different technological foundations. We hope that the insights provided in this paper are a first step towards a universal flexibility model (in a suitable format e.g., ontology, meta-model).

In the future we plan to extend the taxonomy to incorporate other perspectives. Note that although we have defined flexibility types in terms of the control-flow aspects of a given process, other perspectives of a process are also a subject to change. In particular, changes can be applied to the *organizational perspective* that is related to the organizational structure, resources and their roles; *information perspective* that is related to control and production data used in a process; and *application perspective* that is related to the applications used during execution of a given process. Additionaly, 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. Flexible workflow management systems: An approach based on generic process models. In *DEXA '99: Proceedings of the 10th International Conference on Database and Expert Systems Applications*, pages 186–195, London, UK, 1999. Springer-Verlag.
2. W.M.P. van der Aalst. Generic workflow models: How to handle dynamic change and capture management information? In *COOPIS '99: Proceedings of the Fourth IECIS International Conference on Cooperative Information Systems*, pages 115–126, Washington, DC, USA, 1999. IEEE Computer Society.
3. W.M.P. van der Aalst. Exterminating the Dynamic Change Bug: A Concrete Approach to Support Workflow Change. *Information Systems Frontiers*, 3(3):297–317, 2001.
4. W.M.P. van der Aalst and T. Basten. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoretical Computer Science*, 270(1-2):125–203, 2002.
5. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.

6. 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.
7. 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.
8. 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.
9. O. Adam and O. Thomas. A fuzzy based approach to the improvement of business processes. In C. Bussler and A. Haller, editors, *Proceedings of the 1st International Workshop on Business Process Intelligence (BPI 2005)*, volume 3812 of *Lecture Notes in Computer Science*, pages 183–189, Nancy, France, 2005. Springer.
10. 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.
11. M. Adams, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Facilitating flexibility and dynamic exception handling in workflows through worklets. In O. Belo, J. Eder, O. Pastor, and J. Falcão e Cunha, editors, *Proceedings of the CAiSE'05 Forum*, volume 161 of *CEUR Workshop Proceedings*, pages 45–50, Porto, Portugal, 2005. FEUP.
12. 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, *On the Move to Meaningful Internet Systems 2006, OTM Confederated International Conferences, 14th International Conference on Cooperative Information Systems (CoopIS 2006)*, volume 4275 of *Lecture Notes in Computer Science*, pages 291–308. Springer-Verlag, Berlin, 2006.
13. A. Agostini and G. De Michelis. Improving Flexibility of Workflow Management Systems. 1806:218–234, 2000.
14. G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, G. Gunthor, and C. Mohan. Advanced transaction models in workflow contexts. In S.Y.W. Su, editor, *Proceedings of the 12th International Conference on Data Engineering*, pages 574–581, New Orleans, USA, 1996.
15. J.M. Bhat and N. Deshmukh. Methods for modeling flexibility in business processes. In *Workshop on Business Process Modeling, Design and Support (BPMDS05), Proceedings of CAiSE05 Workshops*, 2005. `http://lamswww.epfl.ch/conference/bpmds05/program/Bhat_12.pdf`.
16. 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.
17. F. Casati, S. Ceri, S. Paraboschi, and G. Pozzi. Specification and implementation of exceptions in workflow management systems. *ACM Transactions on Database Systems*, 24(3):405–451, 1999.
18. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. In B. Thalheim, editor, *Conceptual Modeling - ER'96, 15th International Conference on Conceptual Modeling*, volume 1157 of *Lecture Notes in Computer Science*, pages 438–455. Springer, Cottbus, Germany, 1996.

19. D.K.W. Chiu, Q. Li, and K. Karlapalem. ADOME-WFMS: Towards cooperative handling of workflow exceptions. In *Advances in Exception Handling Techniques*, pages 271–288. Springer-Verlag, New York, NY, USA, 2001.
20. V. Christophides, R. Hull, A. Kumar, and J. Simeon. Workflow mediation using vortexml. *IEEE Data Engineering Bulletin*, 24(1):40–45, 2001.
21. G. Cugola. Tolerating deviations in process support systems via flexible enactment of process models. *IEEE Trans. Softw. Eng.*, 24(11):982–1001, 1998.
22. F. Daoudi and S. Nurcan. A benchmarking framework for methods to design flexible business processes. *Software Process Improvement and Practice*, 12:51–63, 2007.
23. S. Dustdar. Caramba - A Process-Aware Collaboration System Supporting Ad Hoc and Collaborative Processes in Virtual Teams. *Distributed and Parallel Databases*, 15(1):45–66, 2004.
24. J. Eder and W. Liebhart. The workflow activity model (WAMO). In S. Laufmann, S. Spaccapietra, and T. Yokoi, editors, *Proceedings of the Third International Conference on Cooperative Information Systems (CoopIS-95)*, pages 87–98, Vienna, Austria, 1995. University of Toronto Press.
25. J. Eder and W. Liebhart. Workflow recovery. In *Proceedings of the First IFCIS International Conference on Cooperative Information Systems (CoopIS'96)*, pages 124–134, Brussels, Belgium, 1996. IEEE Computer Society.
26. C. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *COCS '95: Proceedings of conference on Organizational computing systems*, pages 10–21, New York, NY, USA, 1995. ACM.
27. C.A. Ellis and K. Keddara. A Workflow Change Is a Workflow. 1806:201–217, 2000.
28. N.S. Glance, D.S. Pagani, and R. Pareschi. Generalized process structure grammars gpsg for flexible representations of work. In *CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 180–189, New York, NY, USA, 1996. ACM.
29. Daniela Grigori, François Charoy, and Claude Godart. Anticipation to enhance flexibility of workflow execution. In Heinrich C. Mayr, Jirí Lazanský, Gerald Quirchmayr, and Pavel Vogel, editors, *Database and Expert Systems Applications, 12th International Conference, DEXA 2001 Munich, Germany, September 3-5, 2001, Proceedings*, volume 2113 of *Lecture Notes in Computer Science*, pages 264–273. Springer, 2001.
30. C.W. Günther, S. Rinderle, M. Reichert, and W.M.P. van der Aalst. Change Mining in Adaptive Process Management Systems. 4275:309–326, 2006.
31. C. Hagen and G. Alonso. Flexible exception handling in the OPERA process support system. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS'98)*, pages 526–533, Amsterdam, The Netherlands, 1998. IEEE Computer Society.
32. C. Hagen and G. Alonso. Exception handling in workflow management systems. *IEEE Transactions on Software Engineering*, 26(10):943–958, 2000.
33. J.J. Halliday, S.K. Shrivastava, and S.M. Wheater. Flexible workflow management in the OPENflow system. In *EDOC '01: Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing*, pages 82–98, Washington, DC, USA, 2001. IEEE Computer Society.
34. P. Heimann, G. Joeris, C. Krapp, and B. Westfechtel. Dynamite: Dynamic task nets for software process management. In *Proceedings of the 18th International Conference on Software Engineering (ICSE 18)*, Berlin, Germany, 1996. IEEE Press.

35. 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.
36. T. Herrmann and K.U. Loser. Vagueness in models of socio-technical systems. *Behaviour & Information Technology*, 18(5):313–323, 1999.
37. G. Joeris. Defining flexible workflow execution behaviors. In P. Dadam and M. Reichert, editors, *Workshop Informatik '99: Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications*, volume 24 of *CEUR Workshop Proceedings*, pages 49–55, Paderborn, Germany, 1999. CEUR-WS.org.
38. P.J. Kammer, G.A. Bolcer, R.N. Taylor, A.S. Hitomi, and M. Bergman. Techniques for supporting dynamic and adaptive workflow. *Computer Supported Cooperative Work*, 9(3/4):269–292, 2000.
39. M. Klein, C. Dellarocas, and A. Bernstein, editors. *Adaptive Workflow Systems*, volume 9 of *Special issue of Computer Supported Cooperative Work*, 2000.
40. J. Klingemann. Controlled flexibility in workflow management. In B. Wangler and L. Bergman, editors, *Proceedings of the 12th International Conference on Advanced Information Systems Engineering (CAiSE'00)*, volume 1789 of *Lecture Notes in Computer Science*, Stockholm, Sweden, 2000. Springer.
41. K. Kumar and M. M. Narasipuram. Defining requirements for business process flexibility. In *Workshop on Business Process Modeling, Design and Support (BPMDS06), Proceedings of CAiSE06 Workshops*, pages 137–148, 2006.
42. F. Leymann and D. Roller. Workflow-based applications. *IBM Systems Journal*, 36(1):102–123, 1997.
43. P. Mangan and S. Sadiq. On building workflow models for flexible processes. In *ADC '02: Proceedings of the 13th Australasian database conference*, pages 103–109, Darlinghurst, NSW, Australia, 2002. Australian Computer Society, Inc.
44. N. Mulyar, M. Pesic, W.M.P. van der Aalst, and M. Peleg. Declarative and Procedural Approaches for Modelling Clinical Guidelines: Addressing Flexibility Issues. pages 17–28, 2007.
45. J. Noll. Flexible process enactment using low-fidelity models. In *Proceedings of the International Conference on Software Engineering and Applications (SEA 03)*, 2003.
46. M. Pesic and W.M.P. van der Aalst. A Declarative Approach for Flexible Business Processes. In J. Eder and S. Dustdar, editors, *Business Process Management Workshops, Workshop on Dynamic Process Management (DPM 2006)*, volume 4103 of *Lecture Notes in Computer Science*, pages 169–180. Springer-Verlag, Berlin, 2006.
47. 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.
48. 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.
49. 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. `http://lamswww.epfl.ch/conference/bpmds06/taxbpflex`.
50. 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. http://lamswww.epfl.ch/conference/bpmds05/program/Regev_11.pdf.

51. G. Regev and A. Wegmann. Business process flexibility: Weick's organizational theory to the rescue. In *Proceedings of the 7th Workshop on Business Process Modelling, Development and Support(BPMDS'06)*, 2006. http://lamswww.epfl.ch/conference/bpmds06/taxbpflex.
52. M. Reichert and P. Dadam. ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
53. 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.
54. 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.
55. A. Reuter and F. Schwenkreis. ConTracts – a low-level mechanism for building general-purpose workflow management-systems. *Data Engineering Bulletin*, 18(1):4–10, 1995.
56. Stefanie Rinderle, Manfred Reichert, and Peter Dadam. Correctness Criteria For Dynamic Changes in Workflow Systems: A Survey. *Data and Knowledge Engineering*, 50(1):9–34, 2004.
57. 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.
58. H. Saastamoinen and G.M. White. On handling exceptions. In N. Comstock and C. Ellis, editors, *Proceedings of the ACM Conference on Organizational Computing Systems (COCS'95)*, pages 302–310, Milpitas, CA, USA, 1995. ACM Press.
59. S. Sadiq, O. Marjanovic, and M.E. Orlowska. Managing Change and Time in Dynamic Workflow Processes. *International Journal of Cooperative Information Systems*, 9(1-2):93–116, 2000.
60. 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.
61. A. Sheth. From contemporary workflow process automation to adaptive and dynamic work activity coordination and collaboration. In *DEXA '97: Proceedings of the 8th International Workshop on Database and Expert Systems Applications*, page 24, Washington, DC, USA, 1997. IEEE Computer Society.
62. 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.
63. 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. http://mis.haifa.ac.il/~spnina/publications/flexibility%20Soffer.pdf.
64. R. van Stiphout, T.D. Meijler, A. Aerts, D. Hammer, and R. Le Comte. TREX: Workflow transaction by means of exceptions. In H.-J. Schek, F. Saltor, I. Ramos,

and G. Alonso, editors, *Proceedings of the Sixth International Conference on Extending Database Technology (EDBT'98)*, pages 21–26, Valencia, Spain, 1998. http://citeseer.ist.psu.edu/487690.html.
65. D.M. Strong and S.M. Miller. Exceptions and exception handling in computerized information processes. *ACM Transactions on Information Systems*, 13(2):206–233, 1995.
66. J. Wainer and F. de Lima Bezerra. Constraint-based flexible workflows. In J. Favela and D. Decouchant, editors, *Groupware: Design, Implementation, and Use, 9th International Workshop, CRIWG 2003, Autrans, France, September 28 - October 2, 2003, Proceedings*, volume 2806 of *Lecture Notes in Computer Science*, pages 151–158. Springer, 2003.
67. Wave-Front. *FLOWer 3 Designers Guide*. Wave-Front BV, Apeldoorn, Netherlands, 2004.
68. B. Weber, S. Rinderle, and M. Reichert. Change patterns and change support features in process-aware information systems. In J. Krogstie, A.L. Opdahl, and G. Sindre, editors, *Advanced Information Systems Engineering, 19th International Conference, CAiSE 2007, Trondheim, Norway, June 11-15, 2007, Proceedings*, volume 4495 of *Lecture Notes in Computer Science*, pages 574–588. Springer, 2007.
69. 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/.
70. B. Weber, W. Wild, and R. Breu. CBRFlow: Enabling adaptive workflow management through conversational case-based reasoning. In P. Funk and P.A. González-Calero, editors, *Advances in Case-Based Reasoning, 7th European Conference, ECCBR 2004, Madrid, Spain, August 30 - September 2, 2004, Proceedings*, volume 3155 of *Lecture Notes in Computer Science*, pages 434–448. Springer, 2004.
71. M. Weske. Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System. In R. Sprague, editor, *Proceedings of the Thirty-Fourth Annual Hawaii International Conference on System Science (HICSS-34)*. IEEE Computer Society Press, Los Alamitos, California, 2001.

A Evaluation Criteria

Table 2: Evaluation criteria

Flexibility type	**Full support**	**Partial support**
Flexibility by design		
Parallelism	There is a means (either implicit or explicit) allowing for the thread of control, at a given point in the model, to be split into two or more concurrent branches.	Not applicable.
Choice	There is a means allowing for the thread of control to be diverged into one or more branches.	Not applicable.
Iteration	There is a means allowing for repeatedly executing an activity or sub-process.	Not applicable.

Interleaving	There is a means of executing each member of a set of activities only once and in sequential order and it is not possible to suspend one task during its execution to work on another.	Partial support is given if it has limitations on the set of tasks that can be interleaved or if tasks can be suspended during execution.
Multiple instances	There is a means allowing for multiple concurrent instances of a task to be created.	
Cancellation	There is a means providing the ability to disable a set of tasks.	The process model must be changed in any way (e.g. use of subprocesses, inclusion of bypass tasks) to accommodate the ability to disable a set of tasks.
Flexibility by deviation		
Deviation operations (imperative languages)		
Undo	There is a means allowing for shifting control to the state before execution of a task.	
Redo	There is a means allowing for a previously executed, but now disabled task, to be executed again without shifting control.	
Skip	There is a means passing the point of control over to a state subsequent to a currently enabled task. There is no mechanism to compensate for the skipped task by executing it at a later stage of the execution.	
Create additional instance	There is a means allowing for initiating additional instances of a task that will run in parallel with process instances created on the moment of task instantiation.	
Invoke task	There is a means allowing for a task in the process definition, that is not currently enabled and has not yet been executed, to be initiated. After initiation, the task is undertaken immediately and completed. After completion of the task,the point of control is unaffected.	
Deviation operations (declarative languages)		
Violation of optional constraints	There is mechanism allowing for performing tasks which are not allowed due to constraints.	Not applicable.
Flexibility by underspecification		
Late binding	The realisation of a placeholder is selected from a set of available process fragments which are fully predefined. It is not allowed to construct a new process fragment.	Not applicable

Late modeling	The implementation of a placeholder is selected from a set of available process fragments which are fully predefined but may may also be constructed (either by composition or from scratch).	
Static, before placeholder	The process fragment chosen to complete the placeholder in between process instance commencement and before the first execution of the placeholder, is used to realise the placeholder for each subsequent execution.	Not applicable.
Dynamic, before placeholder	The realisation of the placeholder can be defined in between process instance commencement and the first execution of the placeholder and can be redefined in between the last and next execution of the placeholder.	Not supported.
Static, at placeholder	The process fragment chosen to complete the placeholder during its first execution is used to realise the placeholder for each subsequent execution.	Not applicable.
Dynamic, at placeholder	The realisation of a placeholder can be chosen again for every subsequent execution of the placeholder.	Not supported.
Flexibility by change		
Effect of change		
Momentary change	1) The change is performed at instance level. 2) The change only affects the execution of one or more selected running process instances.	Not applicable.
Evolutionary change	1) The change is performed at type level. 2) Only running and all new process instances are affected by the change.	Not applicable.
Moment of allowed change		
Entry time	It is only possible to perform changes at the time a process instance is created (either by momentary or evolutionary change). After creation, no changes are allowed anymore.	Not applicable.
On-the-fly	It is only possible to perform changes for one or more running process instances (either via momentary or evolutionary change). The changes can be performed at any point in time during process execution.	
Migration strategies		
Forward recovery	It is possible to abort a running process instance which is impacted by evolutionary change.	Not applicable.

Backward recovery	It is possible to abort and restart (compensate if necessary) a running process instance which is impacted by evolutionary change.	Not applicable.
Proceed	Running process instances are handled according to the old process definition and new process instances are handled according to the new process definition. Only applies for evolutionary change.	Not applicable.
Transfer	It is possible to transfer a running process instance, which is impacted by evolutionary change, to a corresponding state in the new process definition.	Not applicable.

B Results for ADEPT1

Table 3: Flexibility types supported by ADEPT1

Flexibility type	**Support**	**Motivation**
Flexibility by design		
Parallelism	+	A node can have AND–split/join semantics.
Choice	+	A node can have XOR–split/join semantics.
Iteration	+	Supported by the loop construct.
Interleaving	–	Not supported.
Multiple instances	–	Not supported.
Cancellation	–	Not supported.
Flexibility by deviation		
Deviation operations (imperative languages)		
Undo	–	Not supported.
Redo	–	Not supported.
Skip	–	Not supported.
Create additional instance	–	Not supported.
Invoke task	–	Not supported.
Deviation operations (declarative languages)		
Violation of optional constraints	–	Not supported.
Flexibility by underspecification		
Late binding	–	Not supported.
Late modeling	–	Not supported.
Static, before placeholder	–	Not supported.
Dynamic, before placeholder	–	Not supported.
Static, at placeholder	–	Not supported.
Dynamic, at placeholder	–	Not supported.
Flexibility by change		
Effect of change		
Momentary change	+	The process of only one process instance can be adapted and thereby guaranteeing model correctness.

Evolutionary change	–	Although ADEPT1 does not offer support for evolutionary change, it is possible to manually withdraw an old process definition and upload a new one at run-time.
Moment of allowed change		
Entry time	–	Changes at type level can be made at the moment a process instance is created.
On–the–fly	+	Changes at type level can be made at any point in time during process execution.
Migration Strategies		
Forward recovery	–	Not supported.
Backward recovery	–	Not supported.
Proceed	–	Not supported.
Transfer	–	Not supported.

C Results for YAWL

Table 4: Flexibility types supported by YAWL

Flexibility type	**Support**	**Motivation**
Flexibility by design		
Parallelism	+	A node can have AND–split/join semantics.
Choice	+	A node can have XOR/OR–split/join semantics.
Iteration	+	Supported by using XOR splits and joins.
Interleaving	+	Supported by using a semaphore.
Multiple instances	+	Supported by a multiple atomic task.
Cancellation	+	Supported by the cancellation region construct.
Flexibility by deviation		
Deviation operations (imperative languages)		
Undo	–	Not supported.
Redo	–	Not supported.
Skip	–	Not supported.
Create additional instance	–	Not supported.
Invoke task	–	Not supported.
Deviation operations (declarative languages)		
Violation of optional constraints	–	Not supported.
Flexibility by underspecification		
Late binding	+	In YAWL, a placeholder is represented by a task which connected to the worklet service. For each placeholder, the right process fragment (called worklet in YAWL) will be chosen from a global collection of process fragments. The actual selection is based on a set of rules.
Late modeling	+	A new process fragment can be modeled from scratch.
Static, before placeholder	–	Not supported.
Dynamic, before placeholder	–	Not supported.
Static, at placeholder	–	Not supported.

Dynamic, at placeholder	+	The appropriate process fragment will be invoked each time the task, which is connected to the worklet service, is executed.
Flexibility by change		
Effect of change		
Momentary change	–	Not supported.
Evolutionary change	–	Although it is not supported, it is possible to upload new models during execution.
Moment of allowed change		
Entry time	–	Not supported
On–the–fly	–	Not supported
Migration strategies		
Forward recovery	–	Not supported.
Backward recovery	–	Not supported.
Proceed	–	Not supported.
Transfer	–	Not supported.

D Results for FLOWer

Table 5: Flexibility types supported by FLOWer

Flexibility type	**Support**	**Motivation**
Flexibility by design		
Parallelism	+	Nodes can have AND–split/join semantics.
Choice	+	Nodes can have XOR/OR–split/join semantics.
Iteration	+	Iteration can be achieved through the use of the sequential plan construct.
Interleaving	+/–	Due to the case metaphor there is only one actor working on the case. Therefore, there is no true concurrency and any parallel routing is interleaved. Since true concurrency is not possible, a partial support rating is given.
Multiple instances	+	Directly supported through dynamic subplans. One can specify whether a user is allowed to initiate additional instances of the task.
Cancellation	–	Not supported.
Flexibility by deviation		
Deviation operations (imperative languages)		
Undo	+	Shifting control to the moment before execution of a task can be done by applying the "redo" option on a task. When a task has been undone all the succeeding tasks that have already been completed are putted back on the wavefront of the Case Guide which means that they are enabled but have to be completed again. Completing again these tasks has as consequence that depending conditions will be re–evaluated again as well.

Redo	+	Supported by filling in again a case form for a task. However, the positions of the nodes on the Case Guide stay the same which means that the moment of control is not changed. This has as effect that succeeding tasks need not to be done again and that depending conditions before the point of control are not re–evaluated again.
Skip	+	It is possible to skip tasks by applying the "skip" option on a task. On the Case Guide, the task is put on the right side of the wavefront which means that is completed.
Create additional instance	–	Not supported.
Invoke task	+	Not supported.
Deviation operations (declarative languages)		
Violation of optional constraints		Not supported.
Flexibility by underspecification		
Late binding	–	Not supported.
Late modeling	–	Not supported.
Static, before placeholder	–	Not supported.
Dynamic, before placeholder	–	Not supported.
Static, at placeholder	–	Not supported.
Dynamic, at placeholder	–	Not supported.
Flexibility by change		
Effect of change		
Momentary change	–	Not supported.
Evolutionary change	–	Although FLOWer does not support evolutionary change, it offers the possibility to update process definitions during execution. The user guide [67] warns the user for performing such updates, as this can lead to unforeseen events and even deadlocks.
Moment of allowed change		
Entry time	–	Not supported.
On–the–fly	–	Not supported.
Migration Strategies		
Forward recovery	–	Not supported.
Backward recovery	–	Not supported.
Proceed	–	Not supported.
Transfer	–	Not supported.

E Results for Declare

Table 6: Flexibility types supported by Declare

Flexibility type	**Support**	**Motivation**
Flexibility by design		
Parallelism	+	A constraint can be defined which exhibits parallelism semantics.
Choice	+	A constraint can be defined which exhibits choice semantics.

Iteration	+	Supported by defining cardinality for tasks.
Interleaving	+	Supported.
Multiple instances	+	Supported by defining cardinality for tasks. Multiple instances of a task can be initiated by starting a task multiple times without completing it.
Cancellation	–	Not supported.
Flexibility by deviation		
Deviation operations (imperative languages)		
Undo		Not supported.
Redo		Not supported.
Skip		Not supported.
Create additional instance		Not supported.
Invoke task		Not supported.
Deviation operations (declarative languages)		
Violation of optional constraints	+	Supported.
Flexibility by underspecification		
Late binding	–	Not supported.
Late modeling	–	Not supported.
Static, before placeholder	–	Not supported.
Dynamic, before placeholder	–	Not supported.
Static, at placeholder	–	Not supported.
Dynamic, at placeholder	–	Not supported.
Flexibility by change		
Effect of change		
Momentary change	+	The process of one process instance can be adapted.
Evolutionary change	+	Changes can be performed at type level.
Moment of allowed change		
Entry time	+	Changes at type level can be made at the moment a process instance is created.
On–the–fly	+	Changes at type level can be made at any point in time during process execution.
Migration Strategies		
Forward recovery	–	Not supported.
Backward recovery	–	Not supported.
Proceed	+	In case that a change is not allowed for a process instance, the affected instance continues executing to the previous process definition.
Transfer	+	In case that a change is allowed, the change is automatically applied to each affected process instance.