Flexibility as a Service

W.M.P. van der Aalst[1,2], M. Adams[2], A.H.M. ter Hofstede[2], M. Pesic[1], and H. Schonenberg[1]

[1] Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
w.m.p.v.d.aalst@tue.nl
[2] Queensland University of Technology
GPO Box 2434, Brisbane QLD 4001, Australia.

Abstract. The lack of flexibility is often seen as an inhibitor for the successful application of workflow technology. Many researchers have proposed different ways of addressing this problem and some of these ideas have been implemented in commercial systems. However, a "one size fits all" approach is likely to fail because, depending on the situation (i.e., characteristics of processes and people involved), different types of flexibility are needed. In fact within a single process/organisation varying degrees of flexibility may be required, e.g., the front-office part of the process may require more flexibility while the back-office part requires more control. This triggers the question whether different styles of flexibility can be mixed and integrated into one system. This paper proposes the *Flexibility as a Service* (FAAS) approach which is inspired by the Service Oriented Architecture (SOA) and our taxonomy of flexibility. Activities in the process are linked to services. Different services may implement the corresponding activities using different workflow languages. This way different styles of modelling may be mixed and nested in any way appropriate. This paper demonstrates the FAAS approach using the YAWL, DECLARE, and WORKLET services.

1 Introduction

Web services, and more generally *Services-Oriented Architectures* (SOAs), have emerged as the standard way of implementing systems in a cross-organisational setting. The basic idea of SOA is to modularise functions and expose them as services using a stack of standards. Although initially intended for automated inter-organisational processes, the technology is also used for intra-organisational processes where many activities are performed by human actors. This is illustrated by the fact that classical workflow technology is being embedded in SOA-style systems (cf. the role of BPEL). Moreover, proposals such as BPEL4People [15] illustrate the need to integrate human actors in processes as has been done in workflow systems since the nineties [2, 16, 13].

Experiences with workflow technology show that it is relatively easy to support structured processes. However, processes involving people and organisations

tend to be less structured. Often, the process is not stable and changes continuously or there are many cases that require people to deviate from the predefined process [5, 7, 19, 20]. Therefore, numerous researchers proposed ways of dealing with flexibility and change. Unfortunately, few of these ideas have been adopted by commercial parties. Moreover, it has become clear that there is no "one size fits all" solution, i.e., depending on the application, different types of flexibility are needed. In this paper we use the taxonomy presented in [23] where four types of flexibility are distinguished: (1) *flexibility by design*, (2) *flexibility by deviation*, (3) *flexibility by underspecification*, and (4) *flexibility by change* (both at type and instance levels). This taxonomy shows that different types of flexibility exist. Moreover, different paradigms may be used, i.e., even within one flexibility type there may be different mechanisms that realise different forms of flexibility. For example, flexibility by design may be achieved by adding various modelling constructs depending on the nature of the initial language (declarative/imperative, graph-based/block-structured, etc.). Therefore, it is not realistic to think that a single language is able to cover all flexibility requirements.

To address the need for flexibility and to take advantage of today's SOAs, we propose *Flexibility as a Service* (FAAS). The idea behind FAAS is that different types of flexibility can be arbitrarily nested using the notion of a service. Figure 1 shows the basic idea. The proposal is not to fix the choice of workflow language but to agree on the interfaces between engines supporting different languages. Activities in one workflow may be subcontracted to a subprocess in another language, i.e., activities can act as *service consumers* while subprocesses, people, and applications act as *service providers*. As Figure 1 shows, languages can be arbitrarily nested.

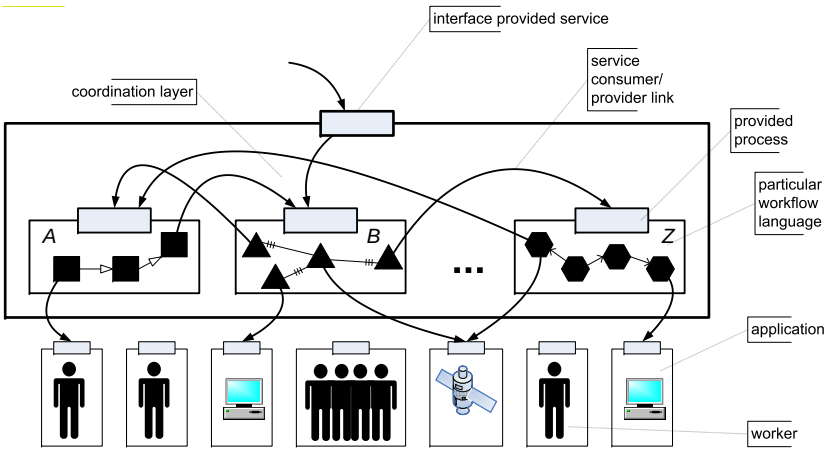

Fig. 1. Overview of FAAS approach: $A, B, \ldots, Z$ may use different languages. The solid arcs are used to connect a service consumer (arc source) to a service provider (arc target).

A particular case (i.e., workflow instance) triggers the invocation of a tree of services. For example, the top-level workflow is executed using language B, some of the activities in this workflow are directly executed by people and applications while other activities are outsourced, as shown by the links between activities and people or applications in Figure 1. This way some of the activities in the top-level workflow can be seen as service consumers that outsource work to service providers using potentially different workflow languages. The outsourced activities are atomic for the top-level workflow (service consumer), however, may be decomposed by the service provider. The service provider may use another workflow language, say A. However, activities expressed in language A may again refer to processes expressed in language C, etc. In fact any nesting of languages is possible as long as the interfaces match.

In this paper, we will show that the FAAS approach depicted in Figure 1 is indeed possible. Moreover, we will show that the different forms of flexibility are complementary and can be merged. This will be illustrated in a setting using YAWL [3], WORKLETS [9], and DECLARE [6]. YAWL is a highly expressive language based on the workflow patterns. Using the advanced routing constructs of YAWL, flexibility by design is supported. WORKLETS offer flexibility by underspecification, i.e., only at run-time the appropriate subprocess is selected and/or defined. DECLARE is a framework that implements various declarative languages, e.g., DecSerFlow and ConDec, and supports flexibility by design, flexibility by deviation, and flexibility by change [17].

The paper is organised as follows. First, we present related work using our taxonomy of flexibility. Section 3 defines the FAAS approach and describes the characteristics of three concrete languages supporting some form of flexibility (YAWL, WORKLETS, and DECLARE). Note that also other workflow languages/systems could have been used (e.g., ADEPT [19]). However, we will show that YAWL, WORKLETS, and DECLARE provide a good coverage. Section 4 describes the implementation. Finally, we show an example that joins all three types of flexibility and conclude the paper.

2 Related Work and Taxonomy of Flexibility

Since the nineties many researchers have been working on workflow modelling, analysis, and enactment [2, 16, 13]. Already in [13] it was pointed out that there are different workflow processes ranging from fully automated processes to ad-hoc processes with considerable human involvement. Nevertheless, most of the *commercial* systems focus on structured processes. One notable exception is the tool FLOWer by Pallas Athena that supports flexibility through case handling [7]. Although their ideas have not been adopted in commercial offerings, many researchers have proposed interesting approaches to tackle flexibility [5, 7, 19, 14, 17, 20].

Using the taxonomy of flexibility presented in [23], we now classify the flexibility spectrum and use this classification to discuss related work.

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. Consider for example a process model which allows the user to select a route; this is a form of flexibility by design. Also parallel processes are more flexible than sequential processes, e.g., a process that allows for A and B to be executed in parallel, also allows for the sequence B followed by A (and vice versa), and thus offers some form of flexibility. In this way, many of the workflow patterns [4] can be seen as constructs to facilitate flexibility by design. A constraint-based language provides a completely different starting point, i.e., anything is possible as long as it is not forbidden [6].

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 activities in the process model for a specific process instance; it does not allow for changes in the process definition or the activities that it comprises. Many systems allow for such functionality to some degree, i.e., there are mechanisms to deviate without changing the model or any modelling efforts. The concept of case handling allows activities to be skipped and rolled back as long as people have the right authorisation [7]. Flexibility by deviation is related to exception handling as exceptions can be seen as a kind of deviation. See [10, 21] for pointers to the extensive literature on this topic.

Flexibility by Underspecification is the ability to execute an incomplete process specification at runtime, 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 runtime, instead the model needs to be completed by providing a concrete realisation for the undefined parts. There are basically two types of underspecification: *late binding* and *late modelling*. Late binding means that a missing part of the model is linked to some pre-specified functionality (e.g., a subprocess) at runtime. Late modelling means that at runtime new (i.e., not predefined) functionality is modelled, e.g., a subprocess is specified. WORKLETS allow for both late modelling and late binding [9, 8]. The various approaches to underspecification are also discussed in [22].

Flexibility by Change is the ability to modify a process definition at runtime 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. There are two types of flexibility by change: (1) *momentary change* (also known as change at the instance level or ad-hoc change): a change affecting the execution of one or more selected process instances (typically only one), and (2) *evolutionary change* (also known as change at the type level): a change caused by modification of the process definition, affecting all new process instances. Changing a process definition leads to all kinds of problems as indicated in [5, 12, 19, 20]. In [12] the concept of the so-called dynamic change bug was introduced. In the context of ADEPT [19]

much work has been performed on workflow change. An excellent overview of problems and related work is given in [20].

Workflow flexibility has been a popular research topic in the last 15 years. Therefore, it is only possible to mention a small proportion of the many papers in this domain here. However, the taxonomy just given provides a good overview of the different approaches, and, more importantly, it shows that there are many different, often complementary, ways of supporting flexibility. Therefore, we propose not to use a single approach but to facilitate the arbitrary nesting of the different styles using the FAAS approach.

3 Linking Different Forms of Flexibility

The idea of FAAS was already introduced using Figure 1. The following definition conceptualises the main idea. The whole of what is shown in Figure 1 is a so-called *workflow orchestration* which consists of two types of services: *workflow services* and *application services*.

Definition 1. *A* workflow orchestration *is a tuple* $WO = (WFS, AS, W)$*, where*

- WFS *is the set of workflow services (i.e., composites of activities glued together using some "process logic"),*
- AS *is the set of application services (i.e., we abstract from the workflow inside such as service),*
- $S = WFS \cup AS$ *is the set of services,*
- *for any* $s \in WFS$*, we defined the following functions:*
  - $lang(s)$ *is the language used to implement the process,*
  - $act(s)$ *is the set of activities in* s*,*
  - $logic(s)$ *defines the process logic, i.e., the causal dependencies between the activities in* $act(s)$ *and expressed in* $lang(s)$*, and*
  - $impl(s) \in act(s) \rightarrow S$ *defines the implementation of each activity in* s*,*
- *from the above we can distill the following wiring relation:* $W = \{(cons, prov) \in WFS \times S \mid \exists_{a \in act(cons)}\ impl(cons)(a) = prov\}$.

The arcs (i.e., the links in wiring relation W) in Figure 1 represent service consumer/provider links. The source of a link is the service consumer while the target of a link is the service provider. The application services are the leaves in Figure 1, i.e., unlike workflow services they are considered to be black boxes and their internal structure is not revealed. They may correspond to a web service, a worklist handler which pushes work-items to workers, or some legacy application. The workflow services have the same "provider interface" but have their own internal structure. In Figure 1, several workflow services using languages $A, B, \ldots Z$ are depicted. Each workflow service s uses a $lang(s)$ to describe the workflow process. In Figure 1, the top-level workflow is using language B. In process s, $act(s)$ refers to the set of activities. Seen from the viewpoint of s, $act(s)$ are black boxes that are subcontracted to other services. $impl(s)(a)$ is the service that executes activity a for workflow service s. The solid

arcs connecting activities to services in Figure 1 describe the wiring relation W. Note that workflow orchestration WO does not need to be static, e.g., modelling activities at runtime may lead to the creation of new services.

The goal of FAAS is that different languages can be mixed in any way. This requires a standard interface between the service consumer and the service provider. There are different ways of implementing such an interface. (For the implementation part of this paper, we use the so-called "Interface B" of YAWL described later.) However, the minimal requirement can be easily described using the notion of a work-item. A work-item corresponds to an activity enabled for a particular case, i.e., an activity instance. A work-item has input data and output data. The input data is information passed on from the service consumer to the service provider and output data is information passed on from the provider to the consumer. Moreover, there are two basic interactions: (1) *check-in work-item* and (2) *check-out work-item*. When an activity is enabled for a particular case, a work-item is checked out, i.e., data and control are passed to the service provider. In the consumer workflow the corresponding activity instance is blocked until the work-item is checked in, i.e., data and control are passed to the service consumer. An interface supporting check-in and check-out interactions provides the bare minimum. It is also possible to extend the interface with other interactions, e.g., "Interface B" of YAWL supports cancellation. However, for the purpose of this paper this is less relevant. The important thing is that all engines used in the workflow orchestration use the same interface. This interface can be simple since we do not consider the process logic in the interface.

As a proof of concept, we demonstrate how three very different languages, YAWL [3], WORKLETS [9], and DECLARE [6], can be combined. First, we discuss the characteristics of the corresponding languages and why it is interesting to combine them. In Section 4 we briefly describe the implementation.

3.1 A Highly Expressive Language: YAWL

The field of workflow lacks commonly accepted formal and conceptual foundations. Many approaches, both in industry and in academia, to workflow specification exist, including (proposed) standards, but till recently, these have not had universal acceptance. In the late nineties the Workflow Patterns Initiative[3] distilled patterns from workflow control-flow specification approaches of a number of commercial systems and research prototypes [4]. These patterns, which are language-independent, can be used to gain comparative insight, can assist with workflow specification and can serve as a basis for language definition.

YAWL (Yet Another Workflow Language) [3] provides comprehensive support for these control-flow patterns and can thus be considered a highly expressive language. This support was achieved by taking Petri nets as a starting point and by observing that Petri nets have the following shortcomings in terms of control-flow patterns support:

- It is hard to capture cancellation of (parts of) a workflow;

[3] www.workflowpatterns.com

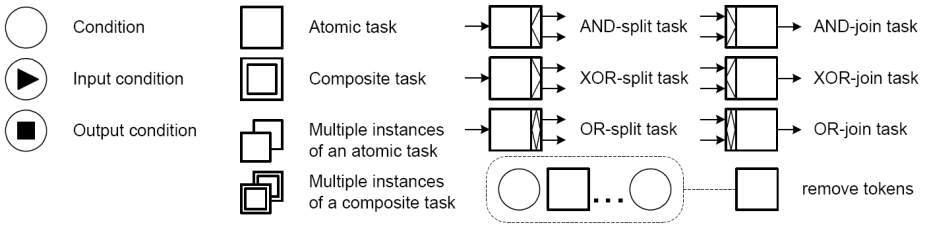

Fig. 2. YAWL control-flow concepts.

- Dealing with various forms of synchronisation of multiple concurrent instances of the same activity is difficult;
- There is no direct support for a synchronisation concept which captures the notion of "waiting only if you have to".

The graphical manifestations of the various concepts for control-flow specification in YAWL are shown in Figure 2. YAWL extends Workflow nets (see e.g. [2]) with concepts for the OR-split and the OR-join, for cancellation regions, and for multiple instance tasks. In YAWL terminology transitions are referred to as *tasks* and places as *conditions*. As a notational abbreviation, when tasks are in a sequence they can be connected directly (without adding a connecting place).

The expressiveness of YAWL allows for models that are relatively compact as no elaborate work-arounds for certain patterns are needed. Therefore the essence of a model is relatively clear and this facilitates subsequent adaptation should that be required. Moreover, by providing comprehensive pattern support YAWL provides flexibility by design and tries to prevent the need for change, deviation, or underspecification.

3.2 An Approach Based on Late Binding & Modelling: Worklets

Typically workflow management systems are unable to handle unexpected or developmental change occurring in the work practices they model, even though such deviations are a common occurrence for almost all processes. WORKLETS provide an approach for dynamic flexibility, evolution and exception handling in workflows through the support of flexible work practices, and based, not on proprietary frameworks, but on accepted ideas of how people actually work.

A *worklet* is in effect a small, self-contained, complete workflow process, designed to perform (or substitute for) one specific task in a larger parent process. The WORKLETS approach provides each task of a process instance with the ability to be associated with an extensible *repertoire* of actions (worklets), one of which is contextually and dynamically bound to the task at runtime.

The actual worklet selection process is achieved by the evaluation of a hierarchical set of rules, called the Ripple Down Rules (RDR) [11]. An RDR Knowledge Base is a collection of simple rules of the form "if condition, then conclusion"

conceptually arranged in a binary tree structure (Figure 3). The tree contains nodes that consist of a condition (a Boolean expression) and a conclusion (a reference to a worklet). During the selection, nodes are evaluated by applying the current context of the case and the task instance. After evaluation, the corresponding branch is taken for the evaluation of the next node. The right (exception) branch is taken when the condition holds, the left (else) branch is taken when the condition is violated. If the condition holds in a terminal node, then its conclusion is taken, i.e., that associated worklet is selected. In case the condition of the terminal node is violated, then for terminal nodes on a true branch the conclusion of the parent node is taken, whereas for terminal nodes on a false branch the conclusion of node with the last satisfied condition is taken.

For new exceptions, the RDR can be extended and new worklets may be added to the repertoire of a task at any time (even during process execution) as different approaches to completing the task are developed. Most importantly, the method used to handle an exception is captured by the system, and so a history of the event and the method used to handle it, is recorded for future instantiations. In this way, the process model undergoes a dynamic natural evolution, thus avoiding the problems normally associated with workflow change, such as migration and version control. The WORKLETS paradigm provides full support for flexibility by underspecification and allows to postpone the realisation of particular part of the process till runtime and be dependent on the context of each process instance.

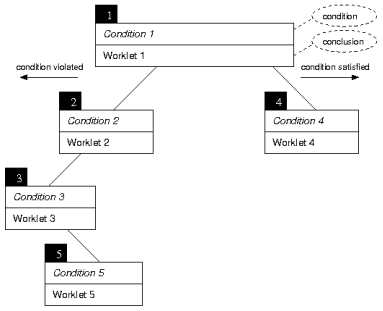

Fig. 3. RDR tree.

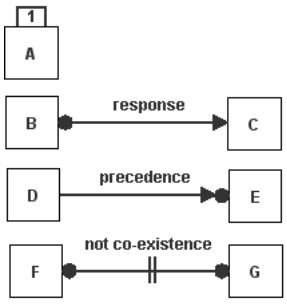

Fig. 4. Declare constraints.

3.3 An Approach Based on Constraints: Declare

In imperative modelling approaches, which are used by most process modelling languages, allowed behaviour is defined in terms of direct causal relations between activities in process models. Opposed to this, a declarative approach offers a wide variety of relations, called constraints, that restrict behaviour. Constraint-based languages are considered to be more flexible than traditional languages because of their semantics; everything that does not violate the constraints is

allowed. Regarding flexibility, imperative and declarative approaches are opposites; to offer more flexibility, for imperative approaches more execution paths need to be incorporated into the model, whereas for declarative approaches the number of constraints need to be reduced. Hence, a declarative approach is more suitable for loosely-structured processes, since it requires less effort to include a variety of behaviours in the design.

DECLARE has been developed as a framework for constraint-based languages and offers most features that traditional WFMSs have, e.g., model development and verification, automated model execution and decomposition of large processes, moreover is offers support for most flexibility types. Activities and constraints on activities are the key elements of a constraint-based model. Constraints are expressed in temporal logic [18], currently LTL is used, but other types of logic could be used as well. By using graphical representations of constraints (called templates), users do not have to be experts in LTL. The set of templates (called a constraint-based language) can easily be extended. DecSerFlow [6] is a such a language, available in DECLARE, and can be used for the specification of web services.

Figure 4 shows some of the constraints available in DecSerFlow. The existence constraint on A expresses that activity A has to be executed exactly once. The response constraint between B and C expresses that if B is executed, then C must eventually also be executed. The precedence constraint between D and E expresses that E can only be executed if D has been executed before, in other words, D precedes E. Finally, the not co-existence constraint between F and G expresses that if F has been executed, then G cannot be executed and vice versa.

DECLARE supports flexibility by deviation by offering two types of constraints: *mandatory* and *optional* constraints. Optional constraints are constraints that may be violated, mandatory constraints are constraints that may not be violated. DECLARE forces its users to follow all mandatory constraints and allows users to deviate by violating optional constraints. Graphically, the difference between mandatory and optional constraints is that the former are depicted by solid lines and the latter by dashed lines. Figure 4 only shows mandatory constraints. In DECLARE it is possible to change the process model during execution, both at type and instance level. A constraint-based model can be changed by adding constraints or activities, or by removing them. For declarative models it is straightforward to transfer instances. Instances for which the current trace satisfies the constraints of the new model, are mapped onto the new model. Hence the dynamic change bug, as described in [12] can be circumvented.

In this section we showed three approaches that provide flexibility. YAWL tries to prevent the need for change, deviation, or underspecification by providing a highly expressive language based on the workflow patterns. WORKLETS allow for flexibility by underspecification by providing an extensible repertoire of actions at runtime. DECLARE uses a constraint-based approach that supports flexibility by design, flexibility by deviation, and flexibility by change. Each of the approaches has its strengths and weaknesses. Therefore, the idea presented

in Figure 1 and Definition 1 is appealing as it allows to combine the strengths of the different approaches and wrap them as services.

4 Implementation

The idea of FAAS has been implemented in the context of YAWL. Since the initial set-up of YAWL was based on the service-oriented architecture, the existing interfaces could be used to also include WORKLETS and DECLARE services. Figure 5 shows the overall architecture of YAWL and the way WORKLETS and DECLARE have been included to allow for the arbitrary nesting of processes as illustrated by Figure 1. The WORKLETS and DECLARE services have been implemented as YAWL Custom Services [1]. Using the web-services paradigm, all external entities (users, applications, devices, organisations) are abstracted as services in YAWL. Custom services communicate with the YAWL engine using a set of pre-defined interfaces, which provide methods for object and data passing via HTTP requests and responses. All data are passed as XML; objects are marshalled into XML on each side of an interface for passing across it, then reconstructed back to objects on the other side.

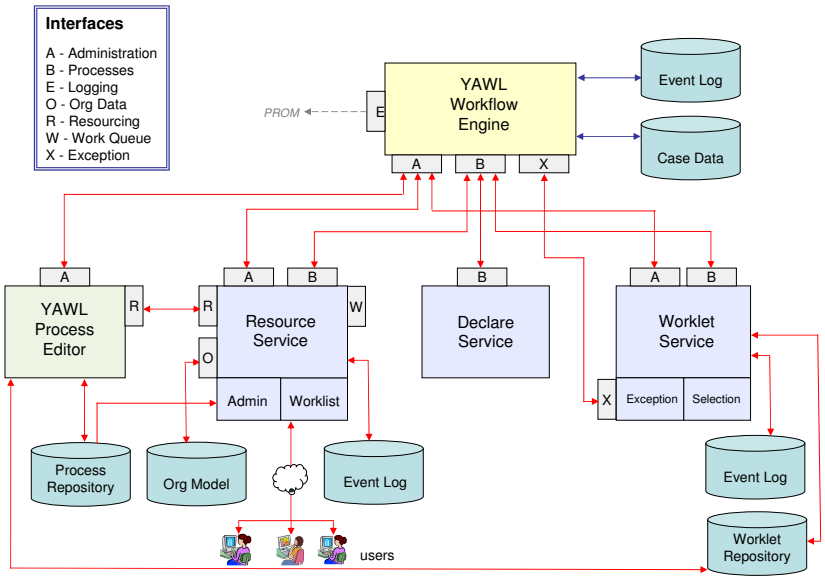

Fig. 5. High-level architecture of the YAWL environment.

As Figure 5 shows, Interface *B* plays a crucial role in this architecture. YAWL can subcontract work to Custom Services. Moreover, Interface *B* can also be used to start new process instances in YAWL. Therefore, the same interface

can be used to provide a service and to use a service. Figure 5 shows that the Worklets and Declare services are connected through Interface B. Note that the Resource Service is also connected to the engine through Interface B. This service offers work to end users through so-called worklists. Again work is subcontracted, but in this case not to another process but to a user or software program.

Note that a Yawl process may contain tasks subcontracted to the Declare and/or Worklets services. In turn, a Declare process may subcontract some of its task to Yawl. For each Declare task subcontracted to Yawl, a process is instantiated in the Yawl engine. This process may again contain tasks subcontracted to the Declare and/or Worklets services, etc. Note that it is not possible to directly call the Declare service from the Worklets service and vice versa, i.e., Yawl acts as the "glue" connecting the services. However, by adding dummy Yawl processes, any nesting is possible as shown in Figure 1. The reason that Yawl is used as an intermediate layer results from the fact that Interface B allows for additional functionalities not mentioned before. For example, custom services may elect to be notified by the engine when certain events occur in the life-cycle of nominated process instantiations (i.e. when a work-item becomes enabled, when a work-item is cancelled, or when a case completes), to signal the creation and completion of process instances and work-items, or to notify the occurrence of certain events or changes in the status of existing work-items and cases. This allows for additional functionality used for e.g. exception handling and unified logging.

The complete YAWL environment, together with source code and accompanying documentation, can be freely downloaded from www.yawl-system.com. The current version of Yawl includes the Worklets service. The Declare service is an optional component that can be downloaded from declare.sf.net.

5 Example

In this section the feasibility of FAAS is illustrated by means of an example which is inspired by the available documentation for the process of filling a vacancy at QUT.[4] The process starts with the preparation of documents and the formation of a selection panel. This is necessary to formally acknowledge the vacancy by the organisational area and to start the advertisement campaign to attract suitable applicants and fill the vacancy. Incoming applications are collected and a shortlist of potential candidates is made. From the shortlisted candidates, a potential candidate is selected based on the outcomes of an interview and one additional selection method. This selection method must be identical for all applicants of the same vacancy. Shortlisted candidates are notified and invited for an interview. In addition, they are informed about the additional selection method. Finally, the application, the interview and the additional selection method outcomes are

[4] Manual of Policies and Procedures (MOPP), Chapter B4 Human Resources - Recruitment, Selection and Appointment (http://www.mopp.qut.edu.au/B/), Queensland University of Technology.

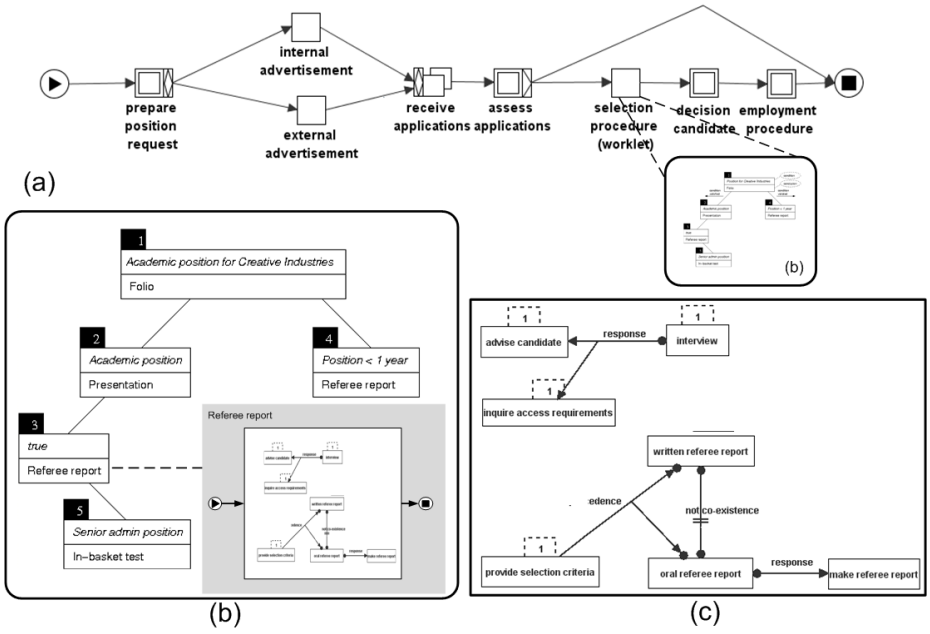

Fig. 6. The application procedure: (a) The high-level process as YAWL net with WORKLET-enabled task *selection procedure* and its RDR tree. (b) The RDR tree in more detail with a link to the Referee report YAWL net. (c) The contents of the Referee report net task is a DECLARE process.

evaluated and one potential candidate is selected to whom the position is offered. After the candidate accepts the offer, the employment procedure is started. In the remainder of this section we illustrate the FAAS ideas by modelling the relevant parts of this process, with YAWL, WORKLETS, and DECLARE.

At the highest level the process can be seen as a sequence of phases, such as the advertisement phase and the selection phase. There are two types of choices at this level. The choice to advertise internally, externally, or both, and the choice whether to go through the selection procedures with a selection of the shortlisted applicants, or to cancel the process in case there are no suitable shortlisted applicants. In addition, it is unknown in advance how many incoming applications will have to be considered. All flexibility requirements at this level can be supported by YAWL, by means of multi-instance tasks, the OR- and XOR-join.

Depending on the type and duration of a vacancy, there is a range of available additional selection methods, folio, referee report, psycho-metric test, in-basket test, presentation and pre-interview presentation. The selection and execution of a particular selection procedure for a particular applicant can be expressed with a WORKLET-enabled task in the YAWL net (cf. Section 3.2). Ripple Down Rules

(RDR [11]) define the selection of a suitable selection procedure for the vacancy at runtime (cf. Section 3.2). The advantage of this approach is that the selection rules can progressively evolve, e.g., for exceptions new rules can be added to the tree. Figure 6(b) depicts the RDR tree for WORKLET-enabled task *selection procedure* of Figure 6(a). Depending on the evaluation of the rule in a specific context, a YAWL net is selected, e.g., the referee process.

The selection method by referee reports is a good example of a loosely-structured process, i.e., a process with just a few execution constraints and many possible execution paths. Prior to the interview, the HR department must advise the candidate about preparation regarding the selection method, and where to find important information. Also the candidate must be asked for his/her requirements regarding access. The referee method consists of requesting a report from a referee, e.g., a former supervisor that knows the candidate professionally. The referee report can be available before or after the interview and be provided written or orally. In case of an oral report, a written summary should be made afterwards. When a referee report is requested, the selection criteria for the position should be provided.

The declarative approach offers a compact way for the specification of the referee report process. Figure 6(c) shows the process modelled in DECLARE using mandatory constraints, *precedence*, *response* and *not co-existence* and optional existence constraints *exactly 1* (cf. Section 3.3). The *not co-existence* allows the specification of the occurrence of receiving either a written report or an oral report from a referee, without specifying *when* it should be decided. The *optional* constraint *exactly 1* expresses that *ideally* the involved tasks are executed exactly once and it allows employees from the organisational area to deviate if necessary.

The example illustrates that through the FAAS approach different languages for (flexible) workflow specification can be combined, yielding a powerful solution to deal with different flexibility requirements. The synchronisation of multiple instances and the multi-choice construct can be solved by YAWL. The context-dependent selection of (sub)processes at runtime can be modelled conveniently using WORKLETS. For loosely-structured (sub)processes DECLARE provides a convenient solution. Moreover, it is not only possible to combine, but also to nest the different approaches, e.g., a YAWL model may contain a DECLARE process in which a WORKLET-enabled task occurs.

6 Conclusion

In this paper, we presented the *Flexibility as a Service* (FAAS) approach. The approach combines ideas from service oriented computing with the need to combine different forms of flexibility. Using a taxonomy of flexibility, we showed that different forms of flexibility are possible and argued that it is not realistic to assume a single language that suits all purposes. The flexibility paradigms are fundamentally different and therefore it is interesting to see how they can be combined without creating a new overarching and complex language. As a proof-of-concept we showed that YAWL, WORKLETS, and DECLARE can be com-

posed easily using the FAAS approach. This particular choice of languages was driven by the desire to cover a large parts of the flexibility spectrum. However, it is possible to include other languages. For example, it would be interesting to also include ADEPT (strong in flexibility by change, i.e., changing processes on-the-fly and migrating instances) and FLOWer (strong in flexibility by deviation using the case handling concept). As indicated in the introduction, a lot of research has been conducted on flexibility resulting in many of academic prototypes. However, few of these ideas have been incorporated in commercial systems. Using the FAAS approach it may be easier for commercial systems to offer specific types of flexibility without changing the core workflow engine.

References

1. W.M.P. van der Aalst, L. Aldred, M. Dumas, and A.H.M. ter Hofstede. Design and Implementation of the YAWL System. In A. Persson and J. Stirna, editors, *Advanced Information Systems Engineering, Proceedings of the 16th International Conference on Advanced Information Systems Engineering (CAiSE'04)*, volume 3084 of *Lecture Notes in Computer Science*, pages 142–159. Springer-Verlag, Berlin, 2004.
2. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems.* MIT press, Cambridge, MA, 2004.
3. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.
4. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
5. 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.
6. W.M.P. van der Aalst and M. Pesic. DecSerFlow: Towards a Truly Declarative Service Flow Language. In M. Bravetti, M. Nunez, and G. Zavattaro, editors, *International Conference on Web Services and Formal Methods (WS-FM 2006)*, volume 4184 of *Lecture Notes in Computer Science*, pages 1–23. Springer-Verlag, Berlin, 2006.
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. 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.
9. 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.

10. F. Casati, S. Ceri, S. Paraboschi, and G. Pozzi. Specification and Implementation of Exceptions in Workflow Management Systems. *ACM Transations on Database Systems*, 24(3):405–451, 1999.
11. P. Compton and B. Jansen. Knowledge in context: A strategy for expert system maintenance. In J.Siekmann, editor, *Proceedings of the 2nd Australian Joint Artificial Intelligence Conference*, volume 406 of *Lecture Notes in Artificial Intelligence*, pages 292–306, Adelaide, Australia, November 1988. Springer-Verlag.
12. C.A. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In N. Comstock, C. Ellis, R. Kling, J. Mylopoulos, and S. Kaplan, editors, *Proceedings of the Conference on Organizational Computing Systems*, pages 10 – 21, Milpitas, California, August 1995. ACM SIGOIS, ACM Press, New York.
13. D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.
14. P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein, and M. Teschke. A Comprehensive Approach to Flexibility in Workflow Management Systems. In G. Georgakopoulos, W. Prinz, and A.L. Wolf, editors, *Work Activities Coordination and Collaboration (WACC'99)*, pages 79–88, San Francisco, February 1999. ACM press.
15. M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, and I. Trickovic. WS-BPEL Extension for People BPEL4People. IBM Corporation, http://www-128.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/, 2005.
16. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.
17. 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.
18. A. Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th IEEE Annual Symposium on the Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, Providence, 1977.
19. M. Reichert and P. Dadam. ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
20. 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.
21. N. Russell, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Workflow Exception Patterns. In E. Dubois and K. Pohl, editors, *Proceedings of the 18th International Conference on Advanced Information Systems Engineering (CAiSE'06)*, volume 4001 of *Lecture Notes in Computer Science*, pages 288–302. Springer-Verlag, Berlin, 2006.
22. S. Sadiq, W. Sadiq, and M. Orlowska. Pockets of Flexibility in Workflow Specification. In *Proceedings of the 20th International Conference on Conceptual Modeling (ER 2001)*, volume 2224 of *Lecture Notes in Computer Science*, pages 513–526. Springer-Verlag, Berlin, 2001.
23. M.H. Schonenberg, R.S. Mans, N.C. Russell, N.A. Mulyar, and W.M.P. van der Aalst. Towards a Taxonomy of Process Flexibility (Extended Version). BPM Center Report BPM-07-11, BPMcenter.org, 2007.