QUALITY OF SERVICE IN FLEXIBLE WORKFLOWS THROUGH PROCESS CONSTRAINTS

Shazia Sadiq, Maria Orlowska
School of Information Technology and Electrical Engineering
The University of Queensland, St Lucia, Brisbane, Australia

Joe Lin, Wasim Sadiq
SAP Research Centre, Brisbane, Australia

Keywords: Flexible Workflows, Workflow Modelling, Process Constraints

Abstract: Workflow technology has delivered effectively for a large class of business processes, providing the requisite control and monitoring functions. At the same time, this technology has been the target of much criticism due to its limited ability to cope with dynamically changing business conditions which require business processes to be adapted frequently, and/or its limited ability to model business processes which cannot be entirely predefined. Requirements indicate the need for generic solutions where a balance between process control and flexibility may be achieved. In this paper we present a framework that allows the workflow to execute on the basis of a partially specified model where the full specification of the model is made at runtime, and may be unique to each instance. This framework is based on the notion of process constraints. Where as process constraints may be specified for any aspect of the workflow, such as structural, temporal, etc. our focus in this paper is on a constraint which allows dynamic selection of activities for inclusion in a given instance. We call these cardinality constraints, and this paper will discuss their specification and validation requirements.

1 INTRODUCTION

Process enforcement technologies have a dominant role in current enterprise systems development. It has been long established that automation of specific functions of enterprises will not provide the productivity gains for businesses unless support is provided for overall business process control and monitoring. Workflows have delivered effectively in this area for a class of business processes, but typical workflow systems have been under fire due to their lack of flexibility, i.e., their limited ability to adapt to changing business conditions. In the dynamic environment of e-business today, it is essential that technology supports the business to adapt to changing conditions. However, this flexibility cannot come at the price of process control, which remains an essential requirement of process enforcement technologies.

Providing a workable balance between flexibility and control is indeed a challenge, especially if generic solutions are to be offered. Clearly there are parts of the process which need to be strictly controlled through fully predefined models. There can also be parts of the same process for which some level of flexibility must be offered, often because the process cannot be fully predefined due to lack of data at process design time. For example, in call centre responses, where customer inquiries and appropriate response cannot be completely pre-defined, or in higher education, where study paths resulting from individual student preferences cannot be entirely anticipated.

In general, a process model needs to be capable of capturing multiple perspectives (Jablonki & Bussler, 1996), in order to fully capture the business process. There are a number of proposals both from research and academia, as well as from industry on the modelling environment (language) that allows these perspectives to be adequately described.

Different proposals offer different level of expressiveness in terms of these perspectives, see e.g. (Sadiq & Orlowska, 1999), (Casati et al 1995), (van der Aalst, 2003), although most focus on the control flow (what activities are performed and in what order).

Basically these perspectives are intended to express the constraints under which the business process can be executed such that the targeted business goals can be effectively met. We see two fundamental classes of these constraints:

Process level constraints: This constitutes the specification of what activities must be included within the process, and the flow dependencies within these activities including the control dependencies (such as sequence, alternative, parallel etc.) and temporal dependencies (such as relative deadlines).

Activity level constraints: This constitutes the specification of various properties of the individual activities within the process, including activity resources (applications, roles and performers), data (produced and/or consumed), and time (duration and deadline constraints).

In this paper, we focus on the flexible definition of process level constraints. We see the level of definition of these constraints along a continuum of specification There is the completely predefined model on one end, and the model with no predefinition on the other. Thus the former only has strong constraints (e.g. A and B are activities of a given process, and B must follow A), and the latter no constraints at all. The former extreme is too prescriptive and not conducive to dynamic business environments; and the latter extreme defeats the purpose of process enforcement, i.e. with insufficient constraints, the process goals may be compromised and quality of service for the process cannot be guaranteed. Finding the exact level of specificity along this continuum will mostly be domain dependent. However, technology support must be offered at a generic level. There is a need to provide a modelling environment wherein the level of specification can be chosen by the process designer such that the right balance between flexibility and control can be achieved.

The work presented in this paper basically discusses flexible process definition for a particular class of constraints. In essence, a small number of constraints are specified at design time, but the process instances are allowed to follow a very large number of execution paths. As long as the given constraints are met, any execution path dynamically constructed at runtime is considered legal. This ensures flexible execution while maintaining a desired level of control through the specified constraints.

In the following sections, we first present the modelling framework which allows flexible process definition. We will then present the details of the constraint specification and validation. In the remaining sections, we will present some background related work to appropriately position this work, and finally a summary of this work and its potential extensions.

2 MODELING FRAMEWORK

The modelling framework required for the specification of process constraints is simple and has minimal impact on the underlying workflow management system. We assume that the underlying WFMS supports a typical graph-based process model and a state-based execution model. Such process models support typical constructs like sequence, fork, choice etc (Figure 1(a)), and activity execution is based on a finite state machine with typical states such as available, commenced, suspended, completed (Figure 1(b)).

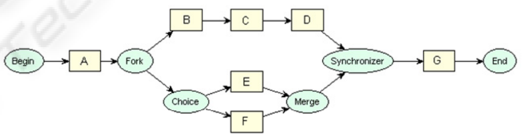

Figure 1: (a) Process Model

The workflow model (W) is defined through a directed graph consisting of nodes (N) and Flows (F). Flows show the control flow of the workflow. Thus W = <N, F> is a Directed Graph where N: Finite Set of Nodes, F: Flow Relation $F \subseteq N \times N$. Nodes are classified into tasks (T) and coordinators (C), where $C \cup T$, $C \cap T = \phi$.

Task nodes represent atomic manual / automated activities or sub processes that must be performed to satisfy the underlying business process objectives. Coordinator nodes allow us to build control flow structures to manage the coordination requirements. Basic modelling structures supported through these coordinators include Sequence, Exclusive Or-Split (Choice), Exclusive Or-Join (Merge), And-Split (Fork), And-Join (Synchronizer), and explicit begin and end coordinators.

A process model will have several activities. An activity t ∈ T is not a mere mode in the workflow graph, but has rich semantics which are defined through its properties, such as input and output data, temporal constraints, resources requirements etc.

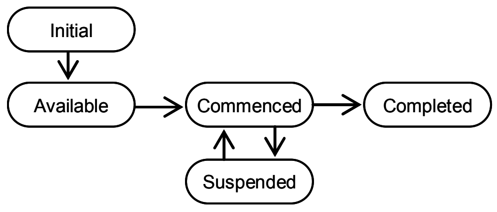

Figure 1: (b) Activity Execution Model

An *instance* within the workflow graph represents a particular case of the process. An *instance type* represents a set of instances that follow the same execution path within the workflow.

Let i be an instance for W.
∀ t ∈ N, we define ActivityState(t, i) → {Initial, Available, Commenced, Completed, Suspened}

We propose to extend the above environment with the following two functions:

A design time function of constraint specification. To provide a facility to specify a pool of activities (including sub-processes) and associated constraints in addition to the core process model. These activities are allowed to be incorporated in the process at any time during execution, but under the given constraints. The core process defines the un-negotiable part of the process, and the pool of activities and associate constraints define the flexible part – thus attempting to strike a balance between flexibility and control.

We associate with every process W, these two additional elements of specification, namely the pool of activities given by P, and a set of constraints given by C. The definition of the flexible workflow W^f is thus given by <W, P, C>.

A run time function of dynamic instance building. To allow the execution path of given instance(s) to be adapted in accordance with the particular requirements for that instance which become known only at runtime. Thus the process for a given instance can be dynamically built based on runtime knowledge, but within the specified constraint set C.

In order to provide explicit terminology, we call the instance specification prior to building, an *open instance*. The instance specification after building we call an *instance template.* Thus the instance template is a particular composition of the given activities within the flexible workflow W^f. The instance templates in turn have a schema-instance relationship with the underlying execution. In traditional terms, the instance template acts as the process model for the particular instance. Execution takes place with full enforcement of all coordination constraints as in a typical production workflow. However, template building is progressive. The process may be changed several times through the available pool of activities and associated constraints. As such the template remains *open* until the process has reached completion.

The main feature of this approach is the utilization of the constraint set C. In previous work, we proposed the use of so called ***structural*** and ***containment*** constraints for flexible workflows (Sadiq et al, 2001), (Sadiq et al, 2004). The constraints belonging to the structural class impose restrictions on how activities can be composed in the templates. The constraints belonging to the containment class identify conditions under which combinations of activities can(not) be contained in the templates.

For example *serial* is a type of structural constraint, where given activities must be executed serially, i.e. not concurrently. However the choice of order remains flexible and is determined by the user during the build. A practical example of a serial constraint can be found in healthcare. Pathologies and medical imaging labs need to schedule a large number of tests in different departments. A number of tests can be prescribed for a given patient e.g. blood test, X-Ray, ECG. These tests can be done in any order but only one at a time. A serial constraint on these activities will ensure this for a given patient or instance.

In this paper, we introduce a new class of constraints for flexible workflows. We call these ***cardinality*** constraints. This new class is especially interesting, because it provides a new means of dealing with two well known challenges in workflow specification, namely n-out-of-m joins and implicit termination. In the sections below, we introduce the framework for the specification of cardinality constraints in flexible workflows. We will also present a means of validating the dynamically built instance (templates) against the specified constraints.

3 CARDINALITY CONSTRAINTS

Cardinality constraints basically define the set of tasks that must be executed within the process, to guarantee that intended process goals will be met. In other words, which tasks must essentially be executed for a process to be considered complete.

The completion of W is explicit due to the presence of an *end* coordinator and also since the tasks within an instance type are pre-determined. However, completion of W^f is not explicit, since the user may make different selections at run time from the available pool of activities.

To further explain this, we define the function Complete (W, i) $\rightarrow$ {True, False}, where

Complete (W, i) = True iff
$\forall$ t $\in$ T, ActivityState(t, i) = Completed | Initial
AND $\exists$ t $\in$ T, ActivityState(t, i) = Completed

Complete (W^f, i) = True iff
Complete (W, i) = True
AND $\exists$ $P_k \subseteq P$, such that
$\forall$ t $\in$ P_k, ActivityState(t, i) = Completed

The interesting question is, how to define the set of tasks that constitute P_k. This requires consideration at both the conceptual and implementation level. As an example, consider the tertiary education domain.

Today's student communities are constantly changing, with more and more part time, mature age and international students with a wide variety of educational, professional and cultural backgrounds. These students have diverse learning needs and styles. Where as degree programs are generally well defined in terms of overall process constraints, it is difficult to judge the quality of specific choices made by students. Tertiary programs often offer a diverse collection of courses that allow specialisation on various aspects of a program. The wide variety of valid combinations of courses that satisfy a particular program's requirement indicates a high degree of flexibility.

The study of a simple program structure was conducted. The program consisted of nine courses of compulsory material and a further three courses of elective material which are selected from a schedule of 14 available electives. This was found to yield a total of some 364 instance types, when considering also the sequence in which the courses can be taken. A further illustration considers a less structured program, such as Arts or Science, where some 20 to 30 courses are required from a schedule that can contain thousands of courses. A number of factors impact on the choices made by the students including changing areas of interest, changing workload requirements and changing program rules. The multiplicity of valid combinations of courses that can be undertaken, and ensuring that these satisfy the requirements of programs, particularly where these requirements have changed during the duration of the student's enrolment constitute a complex problem.

Although academic courses are not currently deployed as workflow tasks in the typical sense, the appropriateness of workflow modelling concepts has been demonstrated (Sadiq & Orlowska, 2002). Academic courses equate to process tasks, these courses are interdependent and the academic program represents a long duration business process. At the same time, there is an inherent flexibility in these processes, required for diverse student requirements, which makes their modelling in traditional prescriptive process definition languages very difficult.

In the sections below, we will demonstrate how the use of cardinality constraints within a flexible workflow modelling framework provides an elegant means of capturing the requirements of such processes. Furthermore, the presented framework also provides a simple means of ensuring that the specified constraints are met for a given instance, thus providing the essential validation support.

3.1 Specification

The specification of the task set P_k that satisfies the completion condition for W^f can be done in three ways:

1. Providing a static set of mandatory tasks which must all be performed in a given instance. In this case, the flexibility is found only in when these tasks will be executed, not which ones. We call this constraint *include.* Specification on *include* is rather straightforward and can be made as *include*: P_k
2. Providing a set of tasks, together with a minimal cardinality for selection, that is at least n out of m tasks must be performed in a given instance. We call this constraint *select.* Specifying select is also simple and can be made by providing the set of tasks, together with an integer n, i.e. *select*: (P, n), where P is the available pool of activities for the flexible workflow. In this case P_k, is any subset of P where $|P_k|$ = n
3. Providing a set of tasks, a minimal cardinality for selection, as well as prescribing some tasks as mandatory. Thus, at least n tasks must be preformed for a given instance, but this selection of n tasks must include the prescribed

mandatory tasks. We call this constraint *minselect.*

Specifying *minselect* requires further consideration, which we present below.

We first introduce the notion of a *family of set A*, as a collection of subsets of A. A notation to represent a family of set A is given by (A`, k; A) and is defined as follows:

$|A| = n$
$A` \subseteq A$ such that $|A`| = m$,
Let k be such that $m+k \leq n$,
$(A`, k; A) = \{ A` \cup B \mid B \in 2^{A \backslash A`}$ and $|B| = k\}$

(A`, k; A) represent a collection of subsets of set A, such that each member of the collection is composed from A` and B. To illustrate further, we present the following simple example:

Let A = {a, b, c, d} and family F = (A`, k; A)
where A` = {a, b} and k = 1,
then F = {{a, b, c}, {a, b, d}}

There are $\begin{pmatrix} k \\ n-m \end{pmatrix}$ number of elemental subsets in the (A`, k; A) family. i.e. the cardinality of the family can be computed by $|(A', k; A)| = (n - m)!/k!(n - m - k)!$.

The notation of (A`, k; A) has the expressive power to represent a collection of subsets without listing every single one. Basically all members of (A`, k; A) family shares the common subset A` in the set, and the remaining subset is selected from the power set of the set difference A\A` where cardinality equals to k. Thus A' represents the mandatory selection.

Since the list of all elements within the (A`, k; A) family may become very large, modelling the *minselect* constraint as (A`, k; A), provides an effective means of capturing a large number of choices effectively. Thus specification of this constraint can be given as *minselect:* (P`, k; P), *where* P is the available pool of activities for the flexible workflow.

For example, the higher education degree program referred to earlier has 14 courses to select from (n = 14, i.e. |A| = n), 9 of which are compulsory courses (m = 9, i.e. |A'| = 9), and with a requirement to take at least 12 courses, students can choose any three, or at least three from the remaining courses, which indicates k = 3.

3.2 Validation

Once the flexible workflow has been defined, including the core process, pool of activities, and process constraints (which may include a number of structural, cardinality or other constraints), instances of the workflow may be created. Instance execution will take place as in typical workflow engines, until the time when a special purpose *build* function is invoked. This function basically allows the instance template to be modified. The next section will elaborate further on how the flexible workflow is managed by the WFMS.

In this section we are interested in what happens, once the build function is invoked, and the instance template has been modified. Clearly the ability to modify the instance template on the fly provides the much desired flexibility. However, the question is, does the modification conform to the prescribed process constraints?

Thus validating an instance template against a given set of constraints needs to be provided. In the context of cardinality constraints, this can be achieved as follows.

In order to validate a dynamically built instance template, we have to ensure that all tasks in P_k are part of the node set of the newly defined instance template. This is required since the condition for completeness of an instance of W^f is dependent on the task set P_k. It can be observed that determining P_k in case of *include* and *select* constraints is a relatively straight forward procedure. In the case of *minselect*, P_k is defined as an element in the family of set P. That is, an instance *i* of W^f , for which a constraint of type *minselect* has been defined, can be guaranteed to complete satisfactorily under the following conditions:

Complete (W^f, i) = True iff
Complete (W, i) = True
AND $\exists P_k \subseteq P$, such that
$P_k \in (P`, k; P)$
AND $\forall t \in P_k$, ActivityState(t, i) = Completed

A very important question to ask is: what happens if we want to specify several cardinality constraints for the same workflow? Could potential conflicts or redundancy arise within the constraint set itself. If so, it must be resolved at design time, that is before any instance templates are built under that constraint set.

A number of relationships may exist between constraints. For example two *minselect* constraints may be specified:

minselect1: (P1', k; P1)
minselect2: (P2', k; P2)

where P1 and P2 are subsets of P, the given pool of activities for W^f.

How do we reason with the constraint set when $P1 \cap P2 \neq \phi$. The full scope of this reasoning is beyond the scope of this paper, however, (Lin & Orlowska, 2004), presents an investigation into dependencies between an arbitrary pair of (A', k; A) families. Three relationships have been identified and analysed, namely Equivalent, Subsume and Imply. This reasoning provides the first step towards a complete analysis of the set of cardinality constraints, in particular *minselect*.

Another important question to be asked is, when is the instance template validated against prescribed process constraints? Clearly, this must be done prior to the instance resuming execution. In the next section, we will provide a detailed view of the procedure to manage the flexible workflow W^f.

3.3 Managing W^f

Below we explain the functions of the flexible workflow management system based on the concepts presented in this paper. The discussion is presented as a series of steps in the specification and deployment of an example process. Figure 2 provides an overview diagram of these steps and associated functions of the flexible workflow engine.

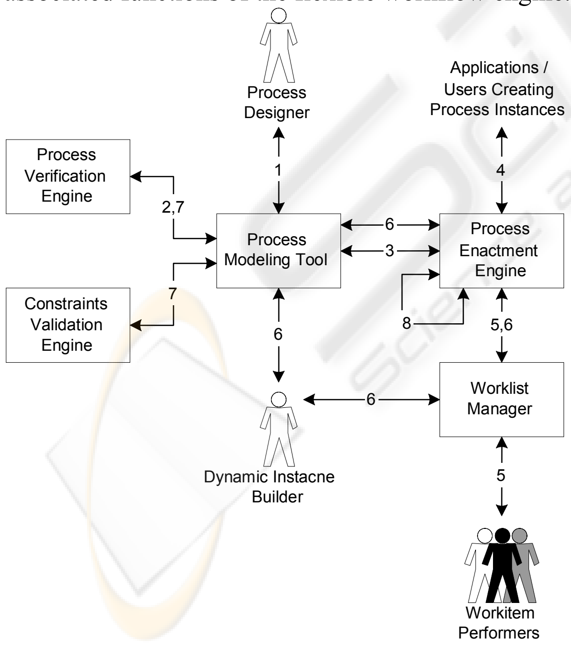

Figure 2: Deployment of a Flexible Workflow

Step 1: The definition of the (flexible) workflow model takes place. The core process, pool of activities and associated constraints are defined.

Step 2: The process is verified for structural errors. The validation of the given constraint set may also takes place at this time.

Step 3: The process definition created above is uploaded to the workflow engine. This process model is now ready for deployment.

Step 4: For each case of the process model, the user or application would create an instance of the process model. On instantiation, the engine creates a copy of the process definition and stores it as an instance template. This process instance is now ready for execution.

Step 5: The available process activities of the newly created instance are assigned to performers (workflow users) through work lists and activity execution takes place as usual, until the instance needs to be dynamically adapted to particular requirements arising at runtime.

Step 6: The knowledge worker or expert user, shown as the dynamic instance builder, will invoke a special build function, and undertake the task of dynamically adapting the instance template with available pool of activities, while guided by the specified constraint set. This revises the instance template.

The build function is thus the key feature of this approach which requires extension of the typical WFMS functionality to include this additional feature. Essentially the build function is the capability to load and revise instance templates for active instances.

Step 7: The next step is to verify the new template, to ensure that it conforms to the correctness properties of the language as well as the given constraints.

Step 8: On satisfactory verification results the newly defined (or revised) instance template resumes execution. Execution will now continue as normal, until completion or until re-invocation of the build function, in which case steps 6-8 will be performed again.

4 RELATED WORK

There have been several works reported in research literature that aim towards providing the necessary support for flexible workflows. So much so, that the term flexible workflows has become rather overloaded. It can range from process evolution, to dealing with workflow exceptions, to flexible modelling frameworks. We position this work in the area of flexible modelling frameworks.

Where as there has been substantial work on the first two aspects, namely process evolution, see e.g. (Ellis, Keddara and Rozenberg, 1995), (Joeris and Herzog, 1998), (Kradolfer and Geppert, 1999), (Sadiq, Marjanovic and Orlowska, 2000). and exception handling, see e.g. (Reichert and Dadam, 1998), (Casati and Pozzi, 1999). In the area of flexible workflow definition, the closest to our approach is the approach followed by rule-based workflows.

(Knolmayer, Endl and Pfahrer, 2000), for example provides a rule based description of a business process and transforms it, by applying several refinement steps, to a set of structured rules which represent the business process at different levels of abstraction.. The underlying concept of developing a workflow specification from a set of rules describing the business processes is similar in principle to the work presented here. However the approach is primarily directed towards the development of coordinated processes that span enterprise boundaries by providing a layered approach that separates the transformation of business (sub-) processes and the derivation of workflow specifications and does not address the issue of catering for processes that cannot be completly predefined.

Moving to the other end of our continuum for organisational processes that spans from highly specified and routine processes to highly unspecified and dynamic processes we acknowledge the significant work that has been performed in the coordination of collaboration intensive processes in the field of CSCW, see e.g. (Bogia and Kaplan, 1995). The complete relaxation of coordination, to support ad-hoc processes is not conducive to the processes targeted by our work.

However, structured ad-hoc workflows, where patterns can be derived form the activities in the process as a result of underlying rules to achieve certain goals, have also been proposed (Han and Shim, 2000). This allows the workflow system to derive the workflow incrementally from workflow fragments and avoids the need to predefine the process prior to enactment. The completion of a structured ad-hoc workflow instance allows flows to be derived for other instances of workflows that share the same process rules. Although defining parts of a process incrementally rather than enacting on a predefining process is similar to the underlying assumption in our work. However the development of this concept to address the modeling of processes that cannot be eloquently predefined contains significant differences as a result of the rules being more explicit and consistent across instances.

Rule based approaches that make use of inference mechanisms have also been proposed for flexible workflows, (Abrahams, Eyers, and Bacon, 2002), (Kappel, Rausch-Schott, and Retschitzegger 2000), (Zeng et al, 2002). For example (Zeng et al, 2002) proposes PLM Flow, which provides a set of business inference rules designed to dynamically generate and execute workflow. The process definition in PLMflow is specified as business rule templates, which include backward-chain rules and forward-chain rules. PLM Flow is a task centric process model. The workflow schema is determined by inferring backward-chain and forward-chain tasks at runtime.

Some researchers have also made use of agent technologies for flexible workflow definition e.g. ADEPT (Jennings et al, 2000), AgFlow (Zeng et al, 2001), and RSA (Debenham, 1998). We present brief summaries below.

ADEPT provides a method for designing agent-oriented business process management system. It demonstrates how a real-world application can be conceived of as a multi-agent system.

AgFlow is an agent-based workflow system built upon a distributed system. The system contains a workflow specification model and the agent-based workflow architecture. The process definition is specify by defining the set of tasks and the workflow process tuple. The control flow aspects can be reflected in the task specific ECA rule.

RSA is an experimental distributed agent-based system based on a 3-layer Believe-Desire-Intension (BDI) architecture, hence the process definition is reflected by the conceptual architecture of the system as a whole.

Inspite of substantial interest from research communities, our study shows that industry acceptance of rule based approaches has been low. Most commercial products continue to provide much more visual languages for workflow specification. Often some variant of Petri-nets, these languages have the dual advantage of intuitive representation as well as verifiability. A key distinguishing feature of our approach from typical rule based approaches is that the the core process as well as the instance template can still be visualized in a graphical language, as well as be supported by essential verification.

5 CONCLUSIONS

Difficulties in dealing with change in workflow systems has been one of the major factors limiting the deployment of workflow technology. At the same time, it is apparent that change is an inherent characteristic of today’s business processes. In this paper we present an approach that recognizes the presence of change, and attempts to integrate the process of defining a change into the workflow process itself. Our basic idea is to provide a powerful means of capturing the logic of highly flexible processes without compromising the simplicity and genericity of the workflow specification language. This we accomplish through process constraints in workflow specifications, which allow workflow processes to be tailored to individual instances at runtime.

Process constraints can be defined for a number of aspects of workflow specification, including selection of activities, as demonstrated in this paper. In addition to selection, they can be defined for structural, resource allocation, as well as temporal constraints for and between workflow activities. One can observe that the design of an appropriate means to facilitate the specification of process constraints is an interesting and challenging issue.

Another interesting and beneficial outcome of the above approach is that ad-hoc modifications can also be provided through essentially the same functionality. Ad-hoc modification means that any unexecuted part of the instance template may be modified at runtime. This is possible since the workflow engine provides the facility to modify instance templates even in the absence of process constraints. However, it is important to point out that we advocate the approach using process constraints over ad-hoc modification because it provides greater control over allowable changes at runtime.

The key feature of this approach is the ability to achieve a significantly large number of process models, from a relatively small number of constraints. Extensions to the constraint set may be envisaged, although it is arguable if such a complete and generic set can be found, and hence achieving flexibility still remains a matter of degree.

REFERENCES

van Der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., Barros, A. P. Workflow Patterns, *Distributed and Parallel Databases*, vol.14 no.1, p.5-51, July 2003.

Abrahams, A., Eyers, D., and Bacon, J. An asynchronous rule-based approach for business process automation using obligations. *ACM SIGPLAN workshop on Rule-based programming*, 2002.

Casati, F., Ceri, S., Pernici, B., Pozzi, G. Conceptual Modeling of Workflows. *Proceedings of the 14th International Conference on Object-Oriented and Entity-Relationship Modelling*, vol. 1021 LNCS, pages: 341 – 354, Springer-Verlag, 1995.

Casati, F., Pozzi, G. Modeling Exception Behaviors in Commercial Workflow Management Systems. *Proceedings of the Fourth IFCIS International Conference on Cooperative Information Systems* (CoopIS99). Edinburgh, Scotland. Sep 2-4, 1999.

Debenham, J. Constructing an Intelligent Multi-agent Workflow System. *Lecturer Notes in Computer Science*: vol. 1502, Springer Verlag, 1998, pp. 119 - 130.

Ellis, S., Keddara, K., Rozenberg, G.. Dynamic Changes within Workflow Systems. *Proceedings of ACM Conference on Organizational Computing Systems* COOCS 95 (1995).

Han, D. and Shim, J. Connector-oriented workflow system for the support of structured ad hoc workflow, *Proceedings of the 33rd Hawaii International Conference on System Sciences*. 2000

Herrmann, T. Evolving workflows by user-driven coordination, *Proceedings of DCSCW*, Munich, Germany, 102–114, September 2000

Jablonski, S., Bussler, C. *Workflow Management-Modeling, Concepts, Architecture and Implementation*, International Thomson Computer Press, 1996.

Jennings, N. R., Faratin, P., T. Norman ,J., O'Brien, P., Odgers, B., and Alty, J. L. Implementing a Business Process Management System using ADEPT: a Real-World Case Study. *International Journal of Applied Artificial Intelligence*, vol. 14, pp. 421--463, 2000

Joeris, G., Herzog, O.. Managing Evolving Workflow Specifications. *Proceedings of the third IFCIS International Conference on Cooperative Information Systems* (CoopIS 98). NewYork, USA. Aug (1998).

Kappel, G., Rausch-Schott, S., and Retschitzegger, W. A Framework for Workflow Management Systems Based on Objects, Rules and Roles. *ACM Computing Surveys*, vol. 32, pp. 27 - 27, 2000.

Knolmayer, G., Endl R. and Pfahrer, M. Modeling processes and workflows by business rules, van der Aalst W. et al. (Eds.) *Business Process Management*, LNCS 1806: 16–29. 2000

Kradolfer, M., Geppert, A.. Dynamic Workflow Schema Evolution based on Workflow Type Versioning and Workflow Migration. *Proceedings of the Fourth IFCIS International Conference on Cooperative Information Systems* (CoopIS99). Edinburgh, Scotland. Sep 2-4, 1999.

Lin, J., Orlowska, M. A new class of constraints for business process modelling. School of Information Technology and Electrical Engineering, The University of Queensland. Technical Report No. 453. Nov 2004.

Reichert, M., Dadam, P. ADEPTflex - Supporting Dynamic Changes of Workflow without loosing control. *Journal of Intelligent Information Systems (JIIS),* Special Issue on Workflow and Process Management 1998.

Sadiq W., Orlowska, M. On capturing Process Requirements of Workflow Based Information Systems. *Proceedings of the 3rd International Conference on Business Information Systems* (BIS ’99), Poznan, Poland. April 14-16, 1999.

Sadiq, S., Marjanovic, O., Orlowska, M. Managing Change and Time in Dynamic Workflow Processes. *The International Journal of Cooperative Information Systems*. Vol 9, Nos 1&2. March-June 2000.

Sadiq, S., Sadiq, W., Orlowska, M. Pockets of Flexibility in Workflow Specifications. *20th International Conference on Conceptual Modeling*, ER’2001, Yokohama Japan, 2001.

Sadiq, S., Sadiq, W., Orlowska, M. Workflow Driven e-Learning – Beyond Collaborative Environments. *Networked Learning in a Global Environment. Challenges and Solutions for Virtual Education*. Berlin, Germany May 1 - 4, 2002.

Sadiq, S., Sadiq, W., Orlowska, M. Specification and Validation of Process Constraints for Flexible Workflows. *Information Systems* (To appear).

Zeng, L., Flaxer, D., Chang, H., and Jeng, J.. PLMflow: Dynamic Business Process Composition and Execution by Rule Inference. *3rd VLDB Workshop on Technologies for E-Services* (TES'02), HongKong P.R.China, 24-25 Aug 2002.

Zeng, L., Ngu, A., Bentallah, B., and O'Dell, M. "An agent-based approach for supporting cross-enterprise workflows," presented at 12th-Australasian-Database-Conference.-ADC-2001, 2001.