Eindhoven, August 12th, 1999

Process Design by Discovery: Harvesting Workflow Knowledge from Ad-hoc Executions

W.M.P van der Aalst
Department of Mathematics and Computing Science (HG 7.73), Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands, +31 40 2474295/2733,wsinwa@win.tue.nl

A.J.M.M. Weijters
Department of Technology Management (PAV D.20), Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands, +31 40 2473857/2290,wsinwa@win.tue.nl

Summary

Both ad-hoc changes and evolutionary changes in workflow processes are hardly supported by actual workflow management systems. The limitations stem from a rigid separation of *design* (i.e., the construction of predefined workflow models) and *enactment* (the actual execution of workflows). In this project proposal design and execution of the system are not separated: The actual executions of cases are used to create an initial design or to revise an existing design. The term *process mining* could be used describe the method of distilling a structured process description from a set of real executions. To achieve this, we use inductive learning techniques. These learning techniques have to employ relations between structured entities as workflow designs. A general investigation of relations between workflow representations based on inheritance is an important general foundation for these learning techniques. Inheritance is one of the cornerstones of object-oriented programming and object-oriented design. Traditional inheritance notions are restricted to the structure of a class (i.e., attributes and methods). These notions only refer to the *static* aspects. In our approach we will elaborate the inheritance notations to the *dynamic* aspects of classes. This allows us to relate and compare the dynamic behavior of several versions/variants of a given workflow process.
The project will experiment with several notions of inheritance and learning techniques. The most promising learning technique will be integrated into an existing workflow management system to enable real experiments. The feedback of these experiments will be used to get insight into the practical limitations of the approach and to improve the robustness of the learning technique.

Problem statement

Today's workflow management systems [Jablonski and Bussler, 1996; Koulopoulos, 1995; Lawrence, 1997] have problems dealing with both ad-hoc changes and evolutionary changes. As a result, the workflow management system is not used to support dynamically changing workflow processes or the workflow process is supported in a rigid manner, i.e., changes are not allowed or handled outside of the workflow management system. At the moment, there are more than 200 workflow products commercially available and many organizations are introducing workflow technology to support their business processes. A critical challenge for workflow management systems is their ability to respond effectively to changes, cf. [Van der Aalst et al., 1998], [Van der Aalst et al., 1999a], [Van der Aalst et al., 1999b], [Agostini and De Michelis, 1998], [Casati et al., 1998], [Ellis et al., 1995], [Ellis et al., 1998], [Han and Sheth, 1998], [Heinl et al. 1998], [Herbst and Karagiannis, 1999], [Ibrahim and Drabble, 1999], [Klein et al., 1998], [Reichert and Dadam, 1998], [Sheth, 1997], [Voorhoeve and Van der Aalst, 1996], [Voorhoeve and Van der Aalst, 1997], and [Wolf and Reimer, 1996]. Changes may range from ad-hoc modifications of the process for a single customer to a complete restructuring of the workflow process to improve efficiency. Today's workflow management systems are ill suited to dealing with change. They typically support a more or less idealized version of the preferred process. However, the real run-time process is often much more variable than the process specified at design-time. The only way to handle changes is to go behind the system's back. If users are forced to bypass the workflow management system quite frequently, the system is more a liability than an asset. Therefore, we take up the challenge to find techniques to add flexibility without loosing the support provided by today's systems.

Some of the perspectives relevant for different kinds of changes and their consequences are:

- *process perspective*, e.g., tasks are added or deleted or their ordering is changed,
- *resource perspective*, e.g., resources are classified in a different way or new classes are introduced,

- *control perspective*, e.g., changing the way resources are allocated to processes and tasks,
- *task perspective*, e.g., upgrading or downgrading tasks,
- *system perspective*, e.g., changes to the infrastructure or the configuration of the engines in the enactment service.

In this proposal, we focus on changes in the process perspective. The process perspective is the most dominant perspective for workflow management and entails challenging problems such as the dynamic change problem, cf. [Van der Aalst et al., 1999b], [Agostini and De Michelis, 1998], [Ellis et al., 1995], and [Ellis et al., 1998]. The other perspectives are also relevant but outside of the scope of this project where we focus on *process mining*.

Figure 1 shows that two kinds of change are identified:

- *Individual (ad-hoc) changes*, i.e., ad-hoc adaptation of the workflow process: a single case (or a limited set of cases) is affected. A good example is that of admitting a person to a hospital: If someone enters the hospital with a cardiac arrest, you are not going to ask him for his ID, although the workflow process may prescribe this. *Figure 1* distinguishes entry time changes (changes that occur when a case is not yet in the system) and on-the-fly changes (while in the system, the process definition for a case changes).
- *Structural (evolutionary) changes*, i.e., evolution of the workflow process: all new cases benefit from the adaptation. A structural change is typically the result of a BPR effort. An example of such a change is the change of a 4-year curriculum at a university to a 5-year one.

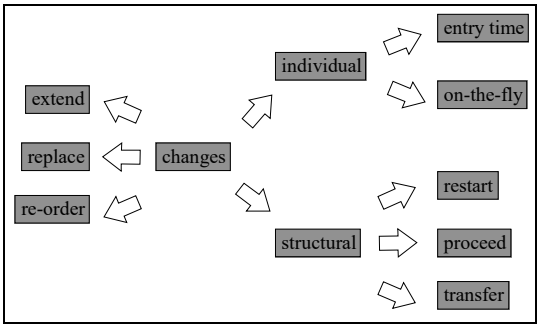

Figure 1. Classification of change.

On the left-hand side in *Figure 1*, we see the three different ways in which a workflow can be changed:

- the process definition is *extended* (e.g., by adding new tasks to cover process extensions),
- tasks are *replaced* by other tasks (e.g., a task is refined into a subprocess), and
- tasks in the process are *re-ordered* (e.g., two sequential tasks are put in parallel).

Figure 1 gives three possible alternatives for handling existing cases in the system when a process definition changes. Dealing with existing cases is only relevant in the case of a structural change because individual changes will always be (similar to) exceptions and, as such, will be dealt with by the one who initiated the change explicitly.

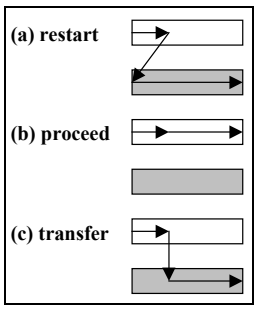

Figure 2. How to handle running cases?

Figure 2 shows the three alternatives: (a) *restart*: running cases are rolled back and restarted at the beginning of the new process, (b) *proceed*: changes do not affect running cases by allowing for multiple versions of the process, and (c) *transfer*: a case is transferred to the new process.

Based on the classification illustrated by both figures, we describe the capabilities of today's workflow management systems with respect to change. Most of the workflow management systems (e.g., Staffware, FlowMark, COSA, etc.) only support highly structured production workflows, i.e., no ad-hoc changes are possible and a versioning mechanism is used to support evolutionary change. The versioning mechanism prevents anomalies such as the dynamic change problem mentioned earlier to occur and corresponds to the proceed policy shown in *Figure 2*. Only a few workflow management systems support ad-hoc workflows (e.g., InConcert and Ensemble) and (to our knowledge) no system truly supports the transfer of cases because of the dynamic change problem. The workflow management systems supporting ad-hoc workflows use a mechanism where each case (i.e., workflow instance) has a private workflow process definition. In other words, the case caries its own description. This mechanism provides flexibility. However, from a management perspective, it is not really a solution. There is no control, i.e., a case can follow an arbitrary route since changes are made local outside of the scope of the workflow designer and manager. Moreover, there is no aggregated management information because the information is at an instance level rather than the process level.

The limitations of today's systems stem from a rigid separation of *design* (i.e., the construction of predefined workflow models) and *enactment* (the actual execution of workflows). The systems supporting production workflows do not allow for modifications, i.e., the design is fixed during enactment. In systems supporting ad-hoc workflow, there is no interaction between design and enactment, i.e., the design (typically some kind of template) can be altered in any way during the actual enactment and the enactment service does not provide feedback for the design.

Approach

The approach used in this project *supports ad-hoc change* and *avoids the problems related to evolutionary change* by *learning*, i.e., the actual executions of cases are used as input for revising the design. The workflow engine (i.e., the enactment service) is no longer just a mechanism to route cases: It also provides controlled flexibility and captures historical data. This historical data is used to create an initial design or to revise an existing design (process mining). Consider for example the processing of insurance claims. Assume there is no process description. First, claims are handled by employees in an ad-hoc manner. The way these claims are handled is recorded. Then, the information recorded is used to distill an appropriate design. This design is used for handling new claims. The design can be seen as a template, i.e., ad-hoc modifications are needed (e.g., added or skipping a task). Not every modification is acceptable, i.e., the design is parameterized by one or more degrees of freedom. The execution of new insurance claims is recorded to detect discrepancies between the initial design and the actual processing. If needed, the design is revised based in this information. Compared to existing tools/approaches the main difference is the use of actual data to construct/revise workflow process definitions, i.e., "design learns from enactment" thus closing the control loop. Clearly such an approach improves the support of ad-hoc change. Moreover, evolutionary changes are easier to handle since the workflow process definitions are kept up-to-date and the degree of freedom is modeled explicitly, i.e., rigorous changes causing all kinds of anomalies can be avoided.

The following questions need to be answered:

- How to represent individual cases and workflow designs?
- How to distill a workflow design from individual cases (process mining)?
- How to model the degree of freedom?
- How to decide when and how to revise an existing design based on actual data?
- How to generate aggregate management information?
- How to build a run-time environment which integrates design and enactment?

Answering these questions we will combine tools and techniques out of the domain of workflow management and knowledge engineering with great emphasis on learning. In this project, we will use Petri nets to represent individual cases and workflow designs, cf. [Van der Aalst, 1998a], [Van der Aalst, 1998b], and [Ellis and Nutt, 1993]. The graphical representation and clearly defined semantics of Petri nets, allows us to express the concepts in a compact and unambiguous manner. Moreover, (extended) Petri nets are close to diagramming techniques used in today's workflow management systems and ERP systems, e.g., the EPC's used in ARIS/SAP [Van der Aalst, 1999b]. However, the use of a highly structured representation language such as Petri nets is from a learning perspective a complicating factor. Standard learning techniques, as developed within machine learning (a sub field of knowledge engineering) are for the greater part based on the use of feature vectors as knowledge

representation language and therefore not straightly applicable on the highly structured representation languages such as Perti nets. For this reason, the answering of the question “How to distill a workflow design from individual cases?” is from a learning perspective a real challenge.

As stated above, machine learning research is typically restricted to classification using feature value vector representations. However, there is a growing interest in developing learning techniques capable of dealing with rich representations, i.e. labeled graphs that can be used for describing relationships among symbols. The most successful approaches are in the sub fields of *Case Based Reasoning* (CBR), learning based on *Genetic Algorithms* (GA), and the *candidate-elimination algorithm*. However, new machine learning techniques are under development.

Case Based Reasoning (CBR) is a learning paradigm that classifies new query instances by analyzing similar instances stored in a database with old cases. Old instances are mostly represented as real or nominal valued feature vectors. However, more rich symbolic descriptions can be used, and the methods used to retrieve similar instances are correspondingly more elaborate [Mitchell, 1997, pp. 240]. CBR has been applied to problems such as conceptual design of mechanical devices based on a stored library of previous design [Sycara et al., 1992], process engineering [Surma and Braunschweig, 1996], and planning [Veloso, 1994; Veloso, 1997]. There is a close connection between the retrieval of similar cases and knowledge about basic relation between instances.

Genetic Algorithms (GA) provide an approach to learning that is based loosely on simulated evolution (therefore the use of the synonym *evolutionary learning*). Learning hypotheses are often described by bit strings whose interpretation depends on the application, though hypothesis may also be described by symbolic expressions or even computer programs [Koza, 1992]. Starting with a population of initial hypotheses, the next generation is developed by applying genetic operations such as crossover and random mutation. At each step, the members of the current generation are evaluated relative to a given measure of fitness hypotheses; the fittest members are probabilistically selected for producing the next generation. Genetic algorithms have been applied successfully to a variety of learning tasks and optimization problems.
The challenge in our project is to reformulate our workflow learning problem into genetic terms. A first attempt can be based on the following approach. The members of the initial population are individual cases. The crossover operations we are looking for are such that there is a strong tendency to combine population members to new more general population elements (i.e. workflow designs). In other words, the crossover operations have a strong tendency to combine individual cases to workflow designs and workflow designs to more general workflow designs. Finally, the fitness of a population element (workflow design) depends of two measurements:

- How well does the workflow design explain the set of individual cases?
- How general is the workflow design (if two workflow designs explain an equal number of individual cases the more general workflow design is fitter)?

The key idea in the *candidate-elimination algorithm* [Mitchell, 1979] is to output a description of a set of all hypotheses (in our case workflow designs) consistent with the individual cases. Surprisingly, the candidate elimination algorithm computes the description of this set without explicitly enumerating all of the members. This is accomplished by using the *more_general_than* partial ordering, to maintain a compact representation of the set of consistent hypotheses and to incrementally refine this representation as each new individual case is encountered. The candidate-elimination algorithm has been applied to problems such as learning regularities in chemical mass spectroscopy [Mitchell, 1979] and learning control rules for heuristic search [Mitchell et al., 1983]. The practical application of the candidate-elimination algorithm is limited by the fact that it performs poorly when given noisy training data. We have to investigate if this is a real limitation in our workflow design domain or if we can overcome this limitation [Hirsh, 1994].

A common property of the mentioned learning techniques is the use of functions on and relations between workflow designs. Examples are the similarity measure in CBR, the fitness function in GA, and the *more_general_than* in the candidate-elimination algorithm. For a more general investigation of functions on and relations between workflow representations we propose an approach based on *inheritance*. Inheritance is one of the cornerstones of object-oriented programming and object-oriented design. The basic idea of inheritance is to provide mechanisms which allow for constructing *subclasses* that inherit certain properties of a given *superclass*. In most object-oriented methods a *class* is

characterized by a set of *attributes* and a set of *methods*. Attributes are used to describe properties of an object (i.e., an instance of the class). Methods symbolize operations on objects (e.g., create, destroy, and change attribute). The structure of a class is specified by the attributes and methods of that class.
The traditional notions of inheritance are applicable to the information and operation perspective. However, for the process perspective the traditional notions of inheritance fall short. For this perspective, a *class* corresponds to a *workflow process definition* (i.e., a workflow schema) and *objects* (i.e., instances of the class) correspond to *cases*. Traditional inheritance notions are restricted to the structure of a class (i.e., attributes and methods). These notions only refer to the *static* aspects of the *interface*. The *dynamic behavior* of a class is either hidden inside the methods or modeled explicitly (in UML the life-cycle of a class is modeled in terms of state machines). Although the dynamic behavior is an intrinsic part of the class description (either explicit or implicit), inheritance of dynamic behavior is not well-understood. (See [] for an elaborate discussion on this topic and pointers to related work.) Given the widespread use of inheritance concepts/mechanisms for the static aspect, this is remarkable. Moreover, the dynamic behavior is the essence of the process perspective. In fact, dynamic behavior is the essence of workflow management. To our knowledge, the work presented in [Basten, 1998] is the only work which deals with inheritance of dynamic behavior in a comprehensive manner. This work is based on a particular class of Petri nets: the so-called *sound workflow nets* (cf. [Van der Aalst, 1998b]) mentioned earlier. This class of Petri nets corresponds to workflow processes without deadlocks, livelocks, and other anomalies. Other inheritance-based approaches abstract from the causal relations between tasks/methods.
Defining inheritance notions for workflow processes (i.e., processes defined by routing diagrams) is far from trivial. Consider two workflow processes x and y. When is x a subclass of y? x is a subclass of superclass y if x inherits certain features of y. Intuitively, one could say that x is a subclass of y if and only if x can do what y can do. Clearly, all tasks present in y should also be present in x. Moreover, x will typically add new tasks. Therefore, it is reasonable to demand that x can do what y can do with respect to the tasks present in y. In fact, the behavior with respect to the existing tasks should be identical. For distinguishing x and y we only consider the old tasks (i.e., the tasks already present in y). All other tasks are renamed to τ. One can think of these tasks as silent, internal, or not observable. Since *branching bisimulation* is used as an equivalence notion, we abstract from transitions with a τ label, i.e., for deciding whether x is a subclass of y only the tasks with a label different from τ are considered. The behavior with respect to these tasks is called the *observable behavior*. With respect to new tasks (i.e., tasks present in x but not in y) there are basically two mechanisms which can be used. The first mechanism simply blocks all new tasks and then compares the observable behavior. This mechanism leads to the following notion of inheritance.

> *If it is not possible to distinguish x and y when only tasks of x that are also present in y are executed, then x is a subclass of y.*

Intuitively, this definition conforms to *blocking* or *encapsulating* tasks new in x. The resulting inheritance concept is called *protocol inheritance*; x inherits the protocol of y, i.e., the old routing patterns are contained in the new process. In other words, if the new tasks are not executed (i.e., blocked), one cannot distinguish any differences. Another mechanism would be to allow for the execution of new tasks but consider only the old ones.

> *If it is not possible to distinguish x and y when arbitrary tasks of x are executed, but when only the effects of tasks that are also present in y are considered, then x is a subclass of y.*

This inheritance notion is called *projection inheritance*; x inherits the projection of the workflow process y onto the old tasks. Projection inheritance conforms to *hiding* or *abstracting* from tasks new in x. In other words, one can still enact the old routing patterns as long as one is willing to execute the appropriate new tasks.
The two mechanisms (i.e., blocking and hiding) result in two orthogonal inheritance notions. Therefore, we also consider combinations of the two mechanisms. A workflow process is a subclass of another workflow process under *protocol/projection inheritance* if by both hiding and blocking one cannot detect any differences, i.e., it is a subclass under both protocol and projection inheritance. The two mechanisms can also be used to obtain a weaker form of inheritance. A workflow process is a subclass of another workflow process under *life-cycle inheritance* if by blocking some newly added tasks and hiding others one cannot distinguish between them.
In [Van der Aalst and Basten, 1997; Basten, 1998] several inheritance preserving transformation rules have been defined. The rules correspond to design constructs that are often used in practice, namely iteration, sequential composition, and parallel composition. If the designer sticks to these rules, inheritance is guaranteed!

These inheritance notions and transformation rules can be used to compare processes. Based on these notions we can define the so-called 'Greatest Common Divisor' (GCD) or the 'Least Common Multiple' (LCM) of a set of workflow process definitions [Van der Aalst and Basten, 1999]. Suppose we have a set of ad-hoc workflows. The GCD of these ad-hoc workflows is the part they all agree on. The LCM captures all possible behaviors. Clearly, such notions are useful to distill a workflow design from individual cases. Moreover, inheritance can also be used to define specific degrees of freedom and to generate aggregate management information [Van der Aalst and Basten, 1999].

Scientific relevance

The scientific relevance of the project is fourfold:

1. *Improving the scientific insight into flexible workflow management systems*. Common workflow management systems have problems dealing with ad-hoc changes and evolutionary changes. The limitations stem from a rigid separation of *design* (i.e., the construction of predefined workflow models) and *enactment* (the actual execution of workflows). In this project proposal design and execution of the system are not separated: the actual executions of cases are used to create an initial design or to revise an existing design.
2. *Improving the scientific insight into the use of machine learning techniques in highly structured domains*. In this project, we will use Petri nets to represent individual cases and workflow designs. The use of the highly structured Petri-net language of is from a learning perspective a challenge. Standard learning techniques as developed within machine learning are for the greater part based on the use of feature vectors as knowledge representation language and therefore not straightforwardly applicable to highly structured representation languages such as Petri nets. Structured representations are ubiquitous in different fields such as knowledge representation, language modeling, and pattern recognition. There is a growing interest in developing learning techniques that can be used for describing complex relationships among symbols.
3. *Insight in the practical usefulness of new inductive learning techniques in the context of workflow management systems*. The implementation of the most promising learning technique into an existing workflow management system is important for insight into the practical limitations of the approach. The practical feedback can be used to improve the robustness of the new learning technique.
4. *The development of new notions for comparing the dynamics of different processes*. The learning techniques used in this project are based on inheritance relations between workflow designs. Traditional inheritance notions are restricted to the structure of a class (i.e., attributes and methods). These notions only refer to the *static* aspects. The inheritance notations deployed in this project focus on the *dynamic* aspects of a class. Many notions are possible, four of which have been mentioned earlier, and further research is needed to select the best one. Moreover, the inheritance notions also lead to intriguing concepts such as the GCD and LCM for processes.

The need for more flexibility has been recognized by the users and vendors of today's workflow management systems and has attracted the interest of many researchers. Several workshops in the past have focused on this issue, cf., [Wolf and Reimer, 1996], [Van der Aalst et al., 1998], [Klein et al., 1998], [Van der Aalst et al., 1999a], and [Ibrahim and Drabble, 1999]. Recently, the issue of "adaptive workflow" has also attracted the attention of several AI researchers. In the near future there will be several workshops focussing on the combination of AI and workflow. Despite this interest in adaptive workflow there are still many open problems. Existing work can be classified as follows:

- There are many survey papers describing the problems related to change, frameworks and architectures, e.g., [Van der Aalst et al., 1999b], [Casati et al., 1998], [Han and Sheth, 1998], [Heinl et al. 1998], and [Sheth, 1997].
- Some papers address specific problems such as the dynamic change problem, e.g., [Van der Aalst and Basten, 1999], [Agostini and De Michelis, 1998], [Ellis et al., 1995], [Ellis et al., 1998], and [Reichert and Dadam, 1998].
- Only a few papers discuss the concept of process mining, e.g., [Agrawal et al., 1998] and [Herbst and Karagiannis, 1999].

The project will build on existing work. It should be noted that the papers focusing on data mining have problems dealing with concurrency. Moreover, none of the above papers above deals with the learning perspective in any detail. Therefore, the project is very relevant from both a practical and scientific perspective.

Goal and planning

The project target is, in the form of a Ph.D. thesis and several papers, to answer the research questions posed before:

- How to represent individual cases and workflow designs?
- How to distill a workflow design from individual cases?
- How to model the degree of freedom?
- How to decide when and how to revise an existing design based on actual data?
- How to generate aggregate management information?
- How to build a run-time environment which integrates design and enactment?

The project starts with a search for appropriate machine learning techniques useful (after adaptation) in the workflow domain. During this search and possible adaptations of existing techniques, performance experiments play a key role.

The next project target is the practical implementation of the more theoretical results in an existing workflow management system. An interesting candidate platform for such an implementation is the ad-hoc workflow management system InConcert [InConcert, 1998]. InConcert has unique features such as the on-the-fly construction of a workflow process definition, which are required for an approach based on learning. The implementation can borrow parts of the software package Woflan developed at EUT [Van der Aalst, 1999a]. Woflan has been developed for the verification of workflow process definitions. However, it can also be used test some of the more complex requirements for the inheritance preserving transformation rules mentioned earlier.

The duration of the project is four years. The following activities are planned:

Year 1
- Participating in the BETA educational program (760 hours workload).
- Reading of the key publication in the domain of workflow management, machine learning and the combination of these two domains.
- Obtaining hands-on experience in using workflow management systems and designing workflows.
- Writing of a definitive research proposal and presentation of this proposal.

Year 2
- Participating in the BETA educational program (200 hours workload).
- Searching for (and adapting of) existing representation languages and machine learning techniques. Preparing the data for the first experiments with selected machine learning techniques. Definition of a methodological sound experimental setup.
- Preparing one or two conference papers.

Year 3
- Carrying out supplementary experiments to get more insight into the first four research questions:
  1. How to represent individual cases and workflow designs?
  2. How to distill a workflow design from individual cases?
  3. How to model the degree of freedom?
  4. How to decide when and how to revise an existing design based on actual data?
- Implementing of the most successful experimental learning module into an existing workflow management system.
- Preparing one or two conference/journal publications.

Year 4
- Continuation of the implementation of the learning module into a workflow management system.
- Carrying out practical experiments with the new workflow management system.
- Writing of the Ph.D. thesis.

References

[Van der Aalst, 1998a] W.M.P. van der Aalst. Chapter 10: Three Good reasons for Using a Petri-net-based Workflow Management System. In T. Wakayama et al., editors, *Information and Process Integration in Enterprises: Rethinking Documents*, pages 161-182. Kluwer Academic Publishers, 1998.

[Van der Aalst, 1998b] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21-66, 1998.

[Van der Aalst, 1999a] W.M.P. van der Aalst. Woflan: A Petri-net-based Workflow Analyzer. *Systems Analysis - Modelling - Simulation*, 35(3):345-357, 1999.

[Van der Aalst, 1999b] W.M.P. van der Aalst. Formalization and Verification of Event-driven Process Chains. *Information and Software Technology*, 41(10):639-650, 1999.

[Van der Aalst and Basten, 1997] W.M.P. van der Aalst and T. Basten. Life-cycle Inheritance: A Petri-net-based approach. In P. Azéma and G. Balbo, editors, Application and Theory of Petri Nets 1997, volume 1248 of Lecture Notes in Computer Science, pages 62-81. Springer, Berlin, Germany, 1997.

[Van der Aalst and Basten, 1999] W.M.P. van der Aalst and T. Basten. Inheritance of Workflows: An Approach Tackling the Problems Related to Change. Computing Science Report 99/6, Eindhoven University of Technology, Eindhoven, the Netherlands, 1999.

[Van der Aalst et al., 1998] W.M.P. van der Aalst, G. De Michelis, and C.A. Ellis (editors). Proceedings of Workflow Management: Net-based Concepts, Models, Techniques and Tools (WFM'98). Computing Science Report 98/7, Eindhoven University of Technology, Eindhoven, the Netherlands, 1998.

[Van der Aalst et al., 1999a] W.M.P. van der Aalst, J. Desel and R. Kaschek (editors). Proceedings of the CAiSE'99 workshop on Software Architectures for Business Process Management (SABPM'99), Heidelberg, Germany, June 1999.

[Van der Aalst et al., 1999b] W.M.P. van der Aalst, T. Basten, H. Verbeek, P. Verkoulen, and M. Voorhoeve. Adaptive Workflow: On the Interplay between Flexibility and Support. In J. Filipe and J. Cordeiro (editors), Proceedings of the first International Conference on Enterprise Information Systems, pages 353-360, Setubal, Portugal, 1999.

[Agostini and De Michelis, 1998]. A. Agostini and G. De Michelis. Simple Workflow Models. In [Van der Aalst et al., 1998], pages 146-164.

[Agrawal et al., 1998] R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In Proceedings of the sixth International Conference on Extending Database Technology (EDBT), 1998.

[Basten, 1998] T. Basten. In Terms of Nets: System Design with Petri Nets and Process Algebra. PhD Thesis. Eindhoven University of Technology, Department of Computing Science, Eindhoven, the Netherlands, 1998.

[Casati et al., 1998] F. Casati, C. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. Data and Knowledge Engineering, 24(3):211-238, 1998.

[Cox and Veloso, 1997] M.T. Cox and M. Veloso. In: *Case-Based Reasoning Research and Development, Proceedings of ICCBR-97, the Second International Conference on Case-Based Reasoning*, pages 531-540. Springer Verlag, Berlin, 1997.

[Ellis et al., 1995] C.A. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In N. Comstock and C.A. Ellis, editors, Conf. on Organizational Computing Systems, pages 10 - 21. ACM SIGOIS, ACM. Milpitas, California, 1995.

[Ellis et al., 1998] C.A. Ellis, K. Keddara, and J. Wainer. Modeling Workflow Dynamic Changes Using Timed Hybrid Flow Nets. In [Van der Aalst et al., 1998], pages 109-128.

[Ellis and Nutt, 1993] C.A. Ellis and G.J. Nutt. Modelling and Enactment of Workflow Systems. In M. Ajmone Marsan, editor, Application and Theory of Petri Nets 1993, volume 691 of Lecture Notes in Computer Science, pages 1-16. Springer, Berlin, Germany, 1993.

[Han and Sheth, 1998] Y. Han and A. Sheth. On Adaptive Workflow Modeling. In Proceedings of the 4th International Conference on Information Systems Analysis and Synthesis, pages 108-116, Orlando, Florida, 1998

[Heinl et al. 1998] P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein, and M. Teschke. A comprehensive approach to flexibility in workflow management systems. Technical report TR-16-1998-6, University of Erlangen-Nuremberg, Erlangen, Germany, 1998.

[Herbst and Karagiannis, 1999] J. Herbst and D. Karagiannis. An Inductive Approach to the Acquisition and Adaptation of Workflow Models. In M. Ibrahim and B. Drabble, editors, *Proceedings of the IJCAI'99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business*, pages 52-57, 1999.

[Hirch, 1994] H. Hirsh. Generalizing Version Spaces. *Machine Learning*, Volume 17. Kluwer Academic Publishers, Boston, 1994.

[Ibrahim and Drabble, 1999] M. Ibrahim and B. Drabble, editors. Proceedings of the IJCAI'99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business, Stockholm, Sweden, August 1999.

[InConcert, 1998] InConcert. *InConcert Process Designer's Guide*. InConcert Inc, Cambridge, Massachusetts, 1998.

[Jablonski and Bussler, 1996] S. Jablonski and C. Bussler. Workflow Management: Modeling Concepts, Architecture, and Implementation International Thomson Computer Press, 1996.

[Klein et al., 1998] M. Klein, C. Dellarocas, and A. Bernstein, (editors). Proceedings of the CSCW-98 Workshop Towards Adaptive Workflow Systems, Seattle, 1998.

[Koulopoulos, 1995] T.M. Koulopoulos. The Workflow Imperative. Van Nostrand Reinhold, New York, 1995.

[Koza, 1992] Koza, J.R. *Genetic Programming: On the programming of computers by means of natural selection*. MIT Press, Cambridge MA, 1992.

[Lawrence, 1997] P. Lawrence (editor). Workflow Handbook 1997, Workflow Management Coalition. John Wiley and Sons, New York, 1997.

[Malone et al., 1999] T. Malone, W. Crowston, J. Lee, B. Pentland, and et. al. Tools for inventing organizations: Toward a handbook for organizational processes. *Management Science*, 45(3):425-443, 1999.

[Michell, 1979] T.M. Mitchell. Version Spaces: A candidate elimination approach to rule learning. *Fifth International Joint Conference on AI,* pages 305-310. MIT Press, Cambridge MA, 1979.

[Michell, 1997] T.M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[Mitchell et al., 1983] T.M. Mitchell, P.E. Utgoff and R. Banjeri. Learning by Experimentation: Acquiring and modifying problem-solving heuristics. In: Michalski, Carbonell, and Mitchell (Eds.), *Machine Learning*, Volume 1, pages 163-190. Tioga Press, 1983.

[Reichert and Dadam, 1998] M. Reichert and P. Dadam. ADEPTflex: Supporting dynamic changes of workflow without loosing control. *Journal of Intelligent Information Systems*, 10(2):93-129, 1998.

[Sheth, 1997] A. Sheth. From Contemporary Workflow Process Automation to Dynamic Work Activity Coordination and Collaboration. Siggroup Bulletin, 18(3):17-20, 1997.

[Surma and Braunschweig, 1996] J. Surma and B. Braunschweig. Case_based Retrieval in Process Engineering: Supporting Design by Reusing Flowsheets. *Engineering Applications of Artificial Intelligence,* 9(4), 1996.

[Sycara et al., 1992] K. Sycara, R. Guttal, J. Koning, S. Narasimham, and D. Navinchandra. CADET: A case-based synthesis tool for engineering design. *International Journal of Expert Systems,* 4(2):157-188, 1992.

[Veleso, 1994] M. Veleso. *Planning and Learning by Analogical Reasoning*. Lectures Notes in Artificial Intelligence No 886. Springer Verlag, Berlin, 1994.

[Voorhoeve and Van der Aalst, 1996] M. Voorhoeve and W.M.P. van der Aalst. Conservative Adaption of Workflow. In [Wolf and Reimer, 1996], pages 1-15.

[Voorhoeve and Van der Aalst, 1997] M. Voorhoeve and W.M.P. van der Aalst. Ad-hoc Workflow: Problems and Solutions. In R. Wagner, editor, Proceedings of the 8th DEXA Conference on Database and Expert Systems Applications, pages 36-41, Toulouse, France, 1997.

[Wolf and Reimer, 1996]. M. Wolf and U. Reimer (editors). Proceedings of the International Conference on Practical Aspects of Knowledge Management (PAKM'96), Workshop on Adaptive Workflow, Basel, Switzerland, 1996.