Flexibility Enhancements in BPM by applying Executable Product Models and Intelligent Agents

Markus Kress, Detlef Seese

kress@aifb.uni-karlsruhe.de, seese@aifb.uni-karlsruhe.de

Abstract: In this paper we present an alternative approach to model and execute business processes. The approach combines a compact model with an intelligent control flow mechanism. Instead of using a pre-designed process model that is executed during runtime by a workflow engine, we use a special model called executable product model (EPM) that is executed by a multi-agent system. The EPM provides a compact representation of the set of possible execution paths of a business process by defining information dependencies instead of the order of activities. In our approach the application of intelligent agents takes advantage of the flexibility provided by the EPM. Relational reinforcement learning (RRL) with a genetic algorithm (GA) is applied for managing the control flow. In experiments we show that business processes can be executed successfully with our approach and that the application of machine learning leads to significant performance gains.

1 Introduction

In practice there is a variety of business process management (BPM) approaches that model business processes on the basis of process models. These process models do not necessarily provide the required flexibility for dealing with changing environments and competitors. Process models must be completely defined in the sense that every aspect and exception must be known and modeled in advance. The main objective of the approach introduced in this paper is to increase the flexibility of the execution of business processes. It differs from other existing approaches as it is based on an executable product model (EPM) instead of an explicitly defined process model.

In [vRL01] a product data model (PDM) was introduced, which is used in a business process design methodology called product driven workflow design. The PDM was developed specifically for the service industry. It consists of information elements and dependencies between them. In the PDM the order of activities is not as rigid predefined as in common process models. In accordance to the design methodology, a process is derived from the PDM based on specific objectives during design time. The flexibility of the PDM is utilized only to create different processes. During runtime the PDM is not used anymore and therefore the flexibility lost. In this approach there is also one problematic aspect. The algorithms for deriving process models rely on data like probabilities or durations that must be estimated. In practice the availability of this kind of information is often limited and also the data quality is not as required. This makes the realistic data estimation difficult and the resulting process model questionable.

In [KMS07] we introduced the EPM approach which has the objective to increase the flexibility in the execution of service industry's business processes. We enhanced the PDM so it can be executed directly without deriving a process first. One of the enhancements was the development of an execution algorithm. Executing the product model directly keeps the flexibility and bypasses the problem with the data estimation. The EPM contains a compact representation of the set of all possible execution paths of a business process. By defining information relationships instead of task sequences as in process models, the alternative possibilities or execution paths in the product model are defined in parallel by default. In a process model, such behavior has to be modeled explicitly. Modeling alternatives using one of the common process modeling languages may lead to a complicated and complex process model.

As the EPM contains variants, a control flow mechanism is required which selects the appropriate execution path. The control flow of process models is normally defined by a fixed number of business rules. Even if the control flow logic is separated by the use of a business rule engine, the flexibility is still restricted. We take advantage of the flexibility provided by the EPM during runtime by using a multi-agent system (MAS). A MAS was chosen, as it is a promising paradigm in distributed environments and has been applied successfully in practice (e. g. in the production industry [BS00]). The MAS manages and controls the execution of EPMs by making intelligent choices regarding the execution decisions. The intelligence stems from two combined machine learning approaches, relational reinforcement learning with a genetic algorithm.

The paper is outlined as follows: Related work is listed in section 2. In section 3 it is explained in detail how a business process with its different execution paths is modeled as an EPM. In section 4 we describe how this EPM is executed using a MAS. The developed machine learning approach is introduced in section 5. It is explained how an execution path is selected using machine learning algorithms. In section 6 the conducted experiments using a simulation approach are summarized and the results discussed. A conclusion and an outlook to future work end the paper.

2 Related Work

The widely used approach for business process execution is based on explicitly defined process models. There exist a multitude of vendor-specific (e. g. FileNet, Staffare) and standardized modeling languages (e. g. BPEL). The execution of such a business process is performed with the help of a workflow engine which is either composed by regular software components, software agents (agent-based workflow [CS96, Sto00]) or both (agent enhanced workflow [JOSC98, STO99]). There are two different approaches in agent-based workflow: (1) Either, business processes are explicitly defined in a model or (2) the business process knowledge is distributed among the different agents participating in the process. Our approach is not based on an explicitly defined process model, but on an EPM.

Several approaches exist that focus on the flexibility of business processes. The approach described in [SP06] makes it possible to model and manage process variants on the basis of process family engineering techniques in an efficient way. But as the actual variant is chosen on instantiation of the business process there is no possibility to change the variant during runtime. Another approach introduced in [TAS05] uses fuzzy logic for decision making. This handles the problem when decisions have to be made based on uncertain data. The approach enables flexible decision making but does not provide more execution options.

To our knowledge there exist only two papers ([CTRB06], [TCR$^+$05]) addressing the combination of MAS and RRL. Both papers are based on the same approach and introduce a classification using properties of the learning problem and agent abilities, e.g. full observability or communication capabilities. They carried out preliminary experiments for the blocks world problem, as well as experiments for a planning task with full observability. As we describe in section 5.3, providing full observability would not be efficient for our approach.

3 The Executable Product Model

The EPM is based on the PDM introduced in [vRL01]. This model was enhanced, resulting in the EPM which can be executed directly without having derived or designed an explicit process model. The enhancement comprises extensions to the original model, relaxation of restrictions, and the development of an algorithm for the determination of executable production rules[1] (see [KMS07] for details). The EPM contains information nodes and dependencies between them. Such a dependency is related to exactly one production rule.

3.1 Information Nodes

The information nodes correspond to abstract information like application data, business objects, documents, decisions which are part of the business process. By defining information nodes as abstract information, the EPM can be modeled on different levels of abstraction. An information node has a type, (descriptive) name and a unique ID. The root node of a product model corresponds to the final decision of the related business process. In the EPM information nodes are visualized as circles.

3.2 Production Rules

The dependencies determine which information must be available before another information can be generated, e.g. a decision can be made only when the application form has been filled out correctly. These dependencies reflect the production or execution order of the

[1]Production rules are similar to activities in a business process

information elements and are modeled as directed arcs. These arcs represent production rules that describe which task has to be performed to create a new value stored in the information node the corresponding arc is pointing at. Each production rule has a unique ID. The information nodes on which the production rule depends on are called origin nodes, the created node is the destination node of the production rule. Each information node is the destination node of at least one production rule (or several in the case of variants). As leaf nodes do not depend on any other node, they have special production rules that can be executed at any time - visualized as dashed arcs. Further elements of the EPM are explained on the basis of an example which is explained in the next paragraph.

3.3 Example of an Executable Product Model

The modeling of an EPM is explained on the basis of a simplified business process of a credit application. All information that accrues during the execution of this process is listed in Table 1.

Table 1: Information elements of the credit application.

Information Element	Explanation
application data	application data of the customer
financial situation check	result of the financial situation check of the customer
customer history check	result of the check of past financial transactions of the customer
contract conditions	calculated contract conditions based on the checks and on the application data
contract approved	indicates whether the contract for the credit was approved or not
final decision	indicates whether the credit was granted or denied

For creating the EPM, these information nodes must be included in the model and the dependencies between them must be specified. The determination of the dependencies / production rules is done in a backwards directed approach. Starting point is the root information node. It must be determined what information must be available for creating the node under consideration. The required information nodes are added as origin nodes of the respective production rule(s). This is done until all information nodes and dependencies have been added to the model. The resulting EPM is depicted in Fig. 1.

Two kinds of dependencies can be used. In an AND dependency all origin nodes of the related production rule must have been created before the production rule becomes executable. In Fig. 1 the production rule with ID 3 is an AND dependency. By means of an OR dependency, variants can be specified. All production rules belonging to an OR dependency can be executed independently from each other. In the example the final decision can be generated by three different production rules. A constraint can be attached to each production rule. Constraints determine under what circumstances a production rule can be executed. They are depicted as square brackets.

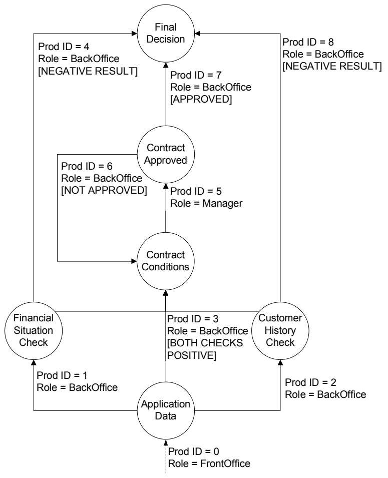


Figure 1: Simplified credit application.

3.4 Execution Algorithm

The execution algorithm was introduced in [KMS07]. It determines the production rules that can be executed based on the dependencies and conditions of the model. But the algorithm does not decide which of these executable production rules are executed. This decision is made by intelligent agents using machine learning algorithms.

4 The Multi-Agent System

The EPM provides flexibility that is utilized by the multi-agent system (MAS). The MAS comprises two different kinds of agents: resource agents (RAs) and intelligent business object agents (IBOs) that are described in the following subsections.

4.1 Resource Agent

A RA is modeled for each human resource (e. g. manager, caseworker) and technical resource (e. g. database, web service). RAs are not processing tasks by themselves. Instead, they negotiate about tasks they are able to execute with the objective to achieve an optimal workload. The actual task processing is done by the related resources. Therefore a RA can be considered as a mediator between one specific resource and IBOs.

4.2 Intelligent Business Object

A new IBO is created for each execution of exactly one product model. This is similar to a workflow engine in which a process instance is created for each process execution. The IBO determines the executable production rules by the means of the execution algorithm. Once the executable production rules are determined, it has to decide which of them should be executed and when. For each production rule that is selected for execution, the IBO negotiates about the assignment with the resource agents and observes the execution progress.

4.3 Learning Aspects

In a MAS without any intelligent approach, there are various parts and aspects that could be improved by applying machine learning. The flexibility provided by the EPM can only be utilized if the IBO decides about the execution order in an efficient way. Therefore this area is subject to the application of machine learning algorithms. The IBO has to learn individually, how the EPM is executed efficiently based on the current situation and the objectives under consideration. For simplification we consider only one objective which is the minimization of the cycle time. The applied learning approaches are explained in the following section, especially how the global optimization objectives can be reached by individual agent learning.

5 Machine Learning Approach

5.1 Introduction to Relational Reinforcement Learning

A variety of different approaches exists in the area of relational reinforcement learning (RRL). For a thorough introduction see [DdRD01], for a survey or overview see e.g.: [TGD04], [van02], or [van05]. RRL is a combination of reinforcement learning (RL) and relational learning (inductive logic programming).

Reinforcement learning is a try and error approach in which an agent selects an action based on its current state and its policy. The execution of an action puts the agent into a new state. Furthermore the agent receives a reward from the environment. The learning objective is to find an optimal policy. A commonly used objective is the maximization of the discounted sum of rewards. In RL the representation of a state is mainly propositional, e.g. a state is decomposed in a set of state features.

In RRL the state, actions and the policy are described by using objects and relations. In many domains this kind of encoding is much more natural. Also the knowledge transfer may be easier using a relational representation. The most common knowledge transfer is the generalization in which previously learned knowledge is applied on a bigger or more complex problem. Another advantage is the possibility of defining background knowledge which can improve the learning process. In relational domains policies have been used successfully as they can be defined in a compact way due to logical abstraction. One of the standard policy search techniques is the policy gradient search. According to [Mv05] logical abstractions do not possess necessarily natural gradients. Because of this they applied a GA for searching the policy space.

5.2 Introduction to Genetic Algorithm

A genetic algorithm is a search heuristic that belongs to the class of evolutionary algorithms that are derived from the process of evolution in biology. Starting point is a set of individuals called population. Each individual represents a solution to the search problem and is evaluated by a fitness function. Genetic operators are applied to evolve the population over time. For this evolution individuals are selected from the population as parents for creating new individuals using operators like crossover and mutation. Individuals are selected for the next generation based on their fitness value. For a detailed introduction see e.g. [Mic98]. Before the combined RRL-GA approach is explained, MAS specific considerations regarding the RRL approach are discussed.

5.3 Relational Reinforcement Learning Adaptations

There are several aspects that need to be considered when using a RRL approach in a MAS scenario, especially how actions of different agents interfere with each other. In a MAS, actions of other agents can interfere and influence the following status. Also depending on the design, an agent cannot be sure about the current status in a parallel scenario. This could be the case if the status changes between the observation of the status and the execution of the chosen action. In our scenario actions of other IBOs only affect the task processing duration, or more precisely the time a task waits for execution.

The MAS for executing the EPMs is designed as decentralized control system. This paradigm is not supposed to be changed for the learning mechanism. For deciding about the execution path(s) the IBO has the knowledge about the product model structure and the status of information nodes and production rules. Other aspects could be considered as well like work load, priority, service level agreements or error proneness but are neglected for simplification and are subject for future work.

In the past it was shown in experiments that individual learning can produce group behavior that influences the performance positively [DMS06]. Our learning mechanism is designed to achieve this as well. In our approach each IBO learns by itself with the common goal to minimize the average cycle time. In simulation-based experiments we have to show that our individual agent learning approach reaches this common goal.

5.4 The RRL-GA Approach

Our RRL-GA approach for learning optimal execution decisions is based on a probabilistic policy as in [IN04]. But instead of using a policy gradient method, we use a GA for reasons described above. The probabilistic policy contains a finite set of predefined rules. Each rule has a condition and action part. Furthermore one probability is assigned to each rule. To select an action based on this policy the condition part of each rule is consecutively evaluated until one evaluates to true. If the condition is fulfilled, the corresponding action is executed based on the assigned probability. If the action is not executed the next rule in the policy is tested. The IBO can perform more than one action per time slot, e.g. it can execute two executable production rules. The following manually created policy was used in the conducted experiments:

$$\begin{aligned}
executable(X),\ creates\_info\_node(X,Y), root\_info\_node(Y) &\longrightarrow execute(X) \quad (1)\\
executable(X),\ active\_production\_rules(Y), Y >= 1 &\longrightarrow do\_nothing() \quad (2)\\
executable(X),\ in\_execution(Y), creates\_info\_node(X,Z),&\\
creates\_info\_node(Y,Z), not\ created(Z) &\longrightarrow do\_nothing() \quad (3)\\
executable(X),\ not\ in\_execution(Y) &\longrightarrow execute(X) \quad (4)\\
execution\_failed(X) &\longrightarrow execute(X) \quad (5)\\
in\_execution(X),\ creates\_info\_node(X,Y), created(Y) &\longrightarrow cancel(X) \quad (6)\\
executable(X) &\longrightarrow execute(X) \quad (7)
\end{aligned}$$

The seven policy rules contain the main predicates[2]. The rules can be explained as follows: Rule (1) executes a production rule that creates the root node. Rule (2) does not execute an executable production rule if there is another active one. With the do_nothing predicate the IBO has the possibility not to perform any action. By applying rule (3), an executable production is not executed if another production rule is in execution that creates the same information node. Rule (4) executes an executable production rule if there is not another one in execution. Rule (5) assures that rescheduling is performed. In this case a failed production rule is executed again. Rule (6) cancels a production rule if an alternative has already created the destination node. This avoids unnecessary processing. Rule (7) executes a production rule that is executable.

An individual contains one chromosome. The chromosomes' gene is related to one probability assigned to one policy rule, so an individual represents a probability vector. For each individual a simulation has to be performed in order to evaluate its fitness value. The fitness is calculated based on the average processing time and the average number of product models in execution. The number of product models in execution is used as a penalty as using the policy for decision making does not necessarily lead to a successful product model execution. There is also the possibility that the agent decides not to finish the product model, which increases the number of product model instances in execution.

[2]There are some more state specific predicates. They are not listed for reasons of space restrictions

6 Experiments

6.1 Simulation Approach and Experimental Settings

In order to conduct experiments with the MAS, a simulation approach is required. For this work, the discrete event simulation approach [LK00, pp. 6-11] was chosen. An additional simulation component "creates" new product model instances entering the system using an exponentially distributed arrival process following (8). It also determines the task processing times (gamma distribution following (9)) as well as the inter-arrival time between two break downs (equation 8) and the duration of a break down of a resource (equation 9).

$$p_\beta(x) = \frac{1}{\beta} e^{-\frac{x}{\beta}} \qquad , \quad \beta > 0 \tag{8}$$

$$p_{\alpha,\beta}(x) = \frac{1}{\beta^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{-\frac{x}{\beta}} \qquad , \quad \alpha, \beta > 0 \tag{9}$$

For a simulation run, the parameters β for the exponential respectively α and β for the gamma distribution have to be chosen for each task and resource. Additionally, the number and types of resource agents available at the MAS have to be configured. As our approach depends on the data in the information nodes, a set of so called *execution instances* that assign pre-defined and consistent values to the information nodes has to be delivered to the simulation. During simulation, each product model instance is uniformly assigned to one of these execution instances.

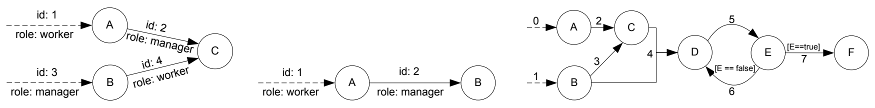
A single system configuration pertains to a specific experimental setting. Such a setting comprises the product model, the negotiation protocol, the execution mechanism, and the n simulation runs, among others.

For the analysis of a single system configuration, we compute a point estimator and a $100(1-\alpha)$ percent confidence interval ($0 < \alpha < 1$) for the expected average $E(X)$ of each interesting value X (see [LK00, pp. 505-515] for details). We used the paired-t confidence interval approach (see [LK00, pp. 557-559] for details) to check whether there is a statistically relevant difference between two system configurations. If the confidence interval for $E(\Delta)$ contains zero, no statistically relevant difference is present. Otherwise, one also gets a quantifier for the difference.

6.2 Conducted Experiments

Based on the first implementation of the MAS without any machine learning algorithm we showed successfully the proof-of-concept of our approach. In this first version we made the simplification that all executable production rules are executed immediately by the IBO. As this was a quite rigid rule we could show considerable improvements by applying our machine learning approach. We could reach the common goal of minimizing the average cycle time by applying our individual agent learning approach.

For the experiments we used different product model structures (see Fig. 2). The big structure is an abstract example containing all elements of an EPM.



(a) Parallel Structure. (b) Linear Structure. (c) Big Structure.

Figure 2: EPMs used for experiments.

The following solutions were found by the RRL-GA approach for two different product model structures:

Parallel:	[1.0, 0.9, 0.3, 0.1, 0.9, 0.7, 1.0]
Linear:	[0.9, 0.4, 0.9, 0.3, 0.3, 0.3, 1.0]

With these probability vectors, the performance experiments were carried out. The results are shown in Table 2. As you can see, the results are highly dependent on the product model structure. For strictly linear structures the execute-all approach is optimal as all production rules must be executed. An experiment with the vector [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0] showed no significant difference between the execute-all and the machine learning approach. If a product model has a parallel structure with various variants the execute-all approach is not efficient due to the unnecessary execution of production rules. Applying the machine learning approach for these product model structures leads to a significant improvement.

Table 2: Results of the performance experiment ($\alpha = 0.05$, 50 simulation runs).

	Parallel Structure			Linear Structure		
	Basic	RRL-GA	Difference	Basic	RRL-GA	Difference
# IBOs	5.76 ± 0.11	2.60 ± 0.02	3.16 ± 0.11	8.96 ± 0.21	10.62 ± 0.32	-1.66 ± 0.39
processing	575.2 ± 9.1	260.5 ± 0.7	314.8 ± 9.3	493.5 ± 10.2	500.1 ± 9.1	-6.7± 13.1

We also performed preliminary experiments for the knowledge transfer. Only for some product models the learned policy resulted in an improvement when applied to the bigger product model (Fig. 2(c)). We assume that more detailed background knowledge should be provided so the IBO is able to make more informed decisions.

The learning mechanism is able to remove inefficient policy rules. For the experiments the rule that every production rule that is in execution is cancelled was included in the hand-coded policy: $in\_execution(X) \longrightarrow cancel(X)$. The policy rules' probability determined by the RRL-GA tended towards zero.

In consideration of the results, the application of our EPM approach in practice is only appropriate for complex business processes (parallel structures with many variants). Otherwise the corresponding EPM does not provide any flexibility the MAS could take advantage of.

7 Conclusion and Future Work

We could successfully execute business processes modeled as EPMs without using process models. As the resource agents are either related to human or technical resources, a resource agent could also be related to a service, e.g. implemented as a web service. In this case information nodes correspond to the outcome of a service invocation. The orchestration of services could be done with the EPM, providing enhanced flexibility on the business process layer in a service-oriented architecture (SOA).

We were also successful in the application of the combination of relational reinforcement learning and genetic algorithm in a multi-agent system scenario in the area of business process management. By using RRL-GA we were able to increase the performance of our system significantly. The promising results of the conducted experiments motivate further study and developments.

We plan to examine several open issues of our approach. First, by providing the IBO with more detailed background knowledge, we hope to get better results in the generalization experiments and further efficiency improvements. Also we intend to apply more complex product model structures to study the generalization in more detail. Another aspect of our future work is the genetic algorithm policy search. Here, we want to enhance the rule learning. Besides the probabilities, the whole policy could be learned. This will be challenging due to the simulation effort that is required.

References

[BS00] S. Bussmann and K. Schild. Self-Organizing Manufacturing Control: An Industrial Application of Agent Technology. In *Proceedings of the 4th International Conference on Multi-Agent Systems*, pages 87–94, Boston, MA, 10-12 July 2000.

[CS96] J. W. Chang and C. T. Scott. Agent-based Workflow: TRP Support Environment (TSE). In *Proceedings of the 5th International World Wide Web Conference*, pages 1501–1511, Paris, France, 6-10 May 1996.

[CTRB06] T. Croonenborghs, K. Tuyls, J. Ramon, and M. Bruynooghe. Multi-Agent Relational Reinforcement Learning - Explorations in Multi-State Coordination Tasks. In *LAMAS*, volume 3898 of *Lecture Notes in Artificial Intelligence*, pages 198–212, 2006.

[DdRD01] S. Dzeroski, L. de Raedt, , and K. Driessens. Relational Reinforcement Learning. *Machine Learning*, 43:7–52, 2001.

[DMS06] T. Dahl, M. J. Mataric, and G. S. Sukhatme. A Machine Learning Method for Improving Task Allocation in Distributed Multi-Robot Transportation. *Complex Engineered Systems*, 2006.

[IN04] H. Itoh and K. Nakamurra. Learning to Learn and Plan by Relational Reinforcement Learning. In *Proceedings Workshop on Relational Reinforcement Learning*, July 8 2004.

[JOSC98] D. Judge, B. R. Odgers, J. W. Shepherdson, and Z. Cui. Agent Enhanced Workflow. *BT Technology Journal*, 16(3):79–85, 1998.

[KMS07] M. Kress, J. Melcher, and D. Seese. Introducing Executable Product Models for the Service Industry. In *Proceedings of the 40. Annual Hawaii International Conference on System Sciences, HICSS'07*, Waikoloa, Hawaii, 3-6 January 2007.

[LK00] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, Boston, MA, 3 edition, 2000.

[Mic98] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 3 edition, November 26 1998.

[Mv05] T. J. Muller and M. van Otterlo. Evolutionary Reinforcement Learning in Relational Domains. In *Proceedings Rich Representations for RL workshop ICML'05*, Bonn, Germany, August 7 2005.

[SP06] A. Schnieders and F. Puhlmann. Variability Mechanisms in E-Business Process Families. *9th International Conference on Business Information Systems (BIS 2006)*, P-85 of LNI:583–601, 2006.

[STO99] J. W. Stepherdson, S. G. Thompson, and B. R. Odgers. Cross Organisational Workflow Co-ordinated by Software Agents. In *Proceedings of the Workshop on Cross-Organisational Workflow Management and Co-ordination*, San Franciso, CA, 22 February 1999.

[Sto00] H. Stormer. Task Scheduling in Agent-Based Workflow. In *International ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce*, Wollongong, Australia, 11-13 December 2000.

[TAS05] O. Thomas, O. Adam, and C. Seel. Business Process Management with Vague Data. In *Proceedings of the 16th International Workshop on Database and Expert Systems Applications (DEXA05)*, 2005.

[TCR$^+$05] K. Tuyls, T. Croonenborghs, J. Ramon, R. Goetschalckx, and M. Bruynooghe. Multi-Agent Relational Reinforcement Learning. In *Proceedings of the Learning and Adaption in MAS Workshop 2005*, pages 123–132, 2005.

[TGD04] P. Tadepalli, R. Givan, and K. Driessens. Relational Reinforcement Learning: An Overview. In *Proceedings of the ICML'04 workshop on Relational Reinfocement Learning*, pages 1–9, Banff, Canada, 2004.

[van02] M. van Otterlo. Relational Expressions in Reinforcement Learning: Review and Open Problems. In E. de Jong and T. Oates, editors, *Proceedings of the ICML'02 Workshop on Development of Representations*, 2002.

[van05] M. van Otterlo. A Survey of Reinforcement Learning in Relational Domains, 2005. Technical Report, Centre for Telematics and Information Technology.

[vRL01] W. M. P. van der Aalst, H. A. Reijers, and S. Limam. Product-driven Workflow Design. In *Proceedings of the 6th International Conference on Computer Supported Cooperative Work in Design*, pages 397–402, London, Ont., Canada, 12-14 July 2001.