*Journal of Universal Computer Science, vol. 15, no. 9 (2009), 1924-1944*
*submitted: 15/8/08, accepted: 25/4/09, appeared: 1/5/09 © J.UCS*

# Pattern-Oriented Workflow Generation and Optimization

**Yong Xiang**
(Dept. of Computer Science and Technology, Tsinghua University, Beijing, P.R.China
xyong@csnet4.cs.tsinghua.edu.cn)

**Shaohua Zhang**
(Dept. of Computer Science and Technology, Tsinghua University, Beijing, P.R.China
zshua@csnet4.cs.tsinghua.edu.cn)

**Yuzhu Shen**
(Dept. of Computer Science and Technology, Tsinghua University, Beijing, P.R.China
syzhu@csnet4.cs.tsinghua.edu.cn)

**Meilin Shi**
(Dept. of Computer Science and Technology, Tsinghua University, Beijing, P.R.China
shi@csnet4.cs.tsinghua.edu.cn)

**Abstract:** Automatic workflow generation is becoming an active research area for dealing with the dynamics of grid infrastructure, because it has a pervasive impact on system usability, flexibility and robustness. Artificial intelligence technology and explicit knowledge have been exploited in some research for workflow construction or composition. With the increasing use of knowledge, its quality has growing impact on system performance. In this report, we present the process pattern as a vehicle for knowledge representation to capture process expertise at the business level. A pattern-based planning approach is proposed for automated workflow generation. Our pattern-oriented approach decreases user-visible complexity and makes systems more scalable and flexible by utilizing explicit knowledge support. Then we propose a hybrid method of pattern knowledge optimization for pattern-based workflow generation planning; experts define the primary model, and subsequent classifier training adjusts and improves the pattern knowledge settings. Experiments with a prototype application demonstrated that this approach can substantially reduce modelling difficulties and effectively improve pattern knowledge quality.

**Keywords:** workflow pattern, workflow generation, knowledge management, business knowledge optimization, classifier training
**Categories:** C.2.4, H.1.0, H.4.3, I.2.6

## 1 Introduction

With the ever-increasing popularity of grid computing, grid workflow, which is an important enabling technology for complex grid applications, has become an active research area. Nowadays, grid workflow offers an attractive basis for supporting processes ranging from *in silico* experiment analysis in bioinformatics to global business activities spanning different organizations.

Generally, a grid workflow system accepts a group of tasks or jobs and assigns them to suitable services or resources for execution. Most existing grid workflow systems build an entire process specification before execution in the grid environment. A process specification can be constructed based on simulation [Cao, 03] or performance prediction [Berman, 01], or even manually [Oinn, 04] [Montoto, 08]. Because all information needed in the execution stage is specified, this method is also called full-ahead plan. In this case, users are often required to know many technical details of the grid environment (e.g. resources' physical locations, service endpoints) for defining the process specification. Moreover, the full-ahead plan often raises exceptions owing to hardware failures or resource usage policy changes at runtime. The situation gets even worse when workflow duration spans several days or weeks. To deal with the issues in large-scale, complex and dynamic environment like grids, automatic workflow construction is becoming necessary. The demand is growing in visibility as grid computing shifts from the scientific community to the business context.

Recently, techniques utilizing artificial intelligence (AI) for automated workflow generation have emerged in some research. This approach offers several advantages. First, it eases the users' burden and improves system usability. Some potential users who are afraid of grid complexity will be encouraged to use the grid [Gil, 04]. Second, the workflow becomes more fault-tolerant and responds more rapidly to exceptions [Cheatham, 05], because the system can promptly handle an error by producing an alternative process plan.

Pegasus [Pegasus, 04] [Deelman,03] is a typical workflow system that integrates AI planning techniques for workflow construction. It is used in GriphyN [GriPhyN, 00] developed by the University of Southern California. Pegasus can take a user's highly specified desired results and then generate a valid workflow for execution. In more detail, Pegasus takes the desired data product as a "goal state", and takes the application components as "operators". Like a typical AI planning system, Pegasus receives inputs of the current state of the environment, a declarative representation of a goal state, and a library of operators that can change states and then searches for a valid, partially ordered set of operators that will transform the current state into goal state with heuristics. The planning result is an executable workflow. It can be transformed into a directed acyclic graph for execution by Condor DAGMan [Frey, 02] to provide the target data product.

The main disadvantage of this planning approach is its lack of explicit knowledge. More exactly, its planner and the knowledge used in planning are mixed together. Therefore, when the description becomes abstract or contains less detail, the planner has more difficulty to yield a good planning result. On the contrary, when the description becomes more exhaustive, the result may be better, but the planner has to understand more complexity. In other words, the planner is tightly bound to a specific domain. This drawback jeopardizes the domain independence of the planner and may harm system scalability. As the problem scale grows, the search space will expand to a huge size and certainly overwhelm the planner.

WMP van der Aalst proposed the “workflow pattern” [Aalst, 03] for systematically analyzing the functionalities of workflow management systems (WfMS). These workflow patterns have been utilized to analyze the workflow behaviour expressiveness of some description languages for the implementation of web service interfaces [Musicante, 06]. Here we propose the use of process patterns to carry the knowledge necessary for workflow planning. Although a process pattern seems very similar to a workflow pattern, they are essentially different. A workflow pattern is mainly about workflow functions and is organized according to control flow structures, e.g. sequence, parallel split. Furthermore, a workflow pattern has no context or solution part in its description. In contrast, a process pattern is a kind of business expertise which is designed to represent knowledge for dynamic workflow generation. A process pattern provides a process solution for a specific user task or goal in a particular scenario, e.g. an express claim procedure in insurance. Problem, context and solution are absolutely necessary in a process pattern.

In some research [Chung, 03] [Rohit, 04], the system has a pre-defined process library for improving planning efficiency. A process pattern is also substantially distinct from a process library. First, a process pattern is a synthesis of business expertise and workflow knowledge. It is defined in an application domain, whereas a process library is a collection of workflow definitions and is defined at the technical level. Second, a process pattern is a dynamic description in multiple dimensions including problem, scenario and solution. The current state of the environment plays an important role in choosing a process pattern. On the contrary, a process in a library is a static description of activities and their dependencies and is used as a building block. Context changes have no direct influence on which process will be used.

As Yolanda Gil et al. [Gil, 04] conclude: “to address more aspects of the grid environment’s workflow management problem … we find that, as mentioned, a more distributed and knowledge-rich approach is required.” In this work, we put forward the process pattern as a knowledge representation structure to capture process expertise at the business level. Based on process patterns, a knowledge-rich, goal-driven planning approach is proposed to automatically generate grid workflow. Besides declarative representation of grid resource entities, it utilizes process patterns to capture business expertise and knowledge. Using pattern-oriented workflow generation planning, the user can submit the business goal in application terms, and the system will generate an executable workflow that can achieve the specified goal.

With the increasing growth in popularity of knowledge, the quality of the knowledge has a significant impact on workflow system performance [Dustdar, 05]. Therefore, a new question has arisen: How can we ensure the correctness and efficiency of knowledge in workflow? This paper also focuses on that problem. To help with this effort, we propose a hybrid mechanism that combines ‘expert fuzzy modelling’ and ‘machine learning’ to construct and refine knowledge in pattern-oriented planning. Experts first build the fundamental pattern models based on their business experience. For some elaborate parameters in a process pattern, the experts assign them estimated values. Then, after necessary tagging of workflow history data, we can calibrate these parameters in more precisely via machine learning.

## 2 Pattern-oriented planning

In order to develop scalable, domain-independent mechanisms for dynamic workflow generation, knowledge should be integrated into the grid workflow management system. In the grid workflow context, we categorize related knowledge into two basic types: grid environment knowledge and application level knowledge. Grid environment knowledge is the declarative representation of grid resource entities and their relationships, capabilities and usage policies. Application level knowledge consists of business process expertise, user preferences, policy constraints, and other intelligence related to business procedures.

Compared with grid environment knowledge, it seems that the value of application-level knowledge is underestimated; it is not addressed adequately in recent research. However, high-level knowledge is even more important when we broaden the range of applications outside the scientific community. In the past, grid workflow often performed MPI/PVM jobs or data-intensive analysis tasks, which are more related to underlying resources, such as computation and storage capacity. Nowadays, grid workflow has broadened its field and begun to supports business processes, i.e. procurement, supply-chain, and coordination of activities or services in different organizations. In these cases, application level knowledge probably plays a more significant role than environment knowledge in the coordination of workflow generation, execution and exception handling.

This section first introduces the process pattern as the knowledge structure that captures implicit knowledge. Then, we introduce the knowledge base used in the system. Finally, pattern-oriented planning is described.

### 2.1 Process Pattern

There are some requirements for a knowledge representation for grid workflow management, especially workflow generation. First, it should be suitable for representing procedural knowledge. In the workflow domain, the most important knowledge is procedural knowledge; the representation approach must be adequate for describing it. Second, it should be efficient for use. We integrate it in applications, so the representation structure must be convenient to program and manipulate. Third, it should be easy to understand, so that a user from a business field could smoothly accept it, use it and examine it.

According to these criteria, some traditional knowledge representation techniques, like predicate logic, frames, semantic networks, rule-based methods etc. are not suitable. The rule-based method is not fit for describing procedural knowledge; frames and semantic network are hard to program into systems; and predicate logic seems too intricate for users and business experts.

In this work, the process pattern is proposed for knowledge representation in the grid workflow generation process.

Pattern originally comes from the architecture domain, and “describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over [Alexsander, 77]”. The basic pattern structure, as shown in Figure 1, includes three parts: problem, scenario and solution. Problem is a description of the task to be handled; Scenario describes the contextual information of the environment that the problem inhabits; Solution provides a guideline to perform the task or gives answer to the problem. A practical pattern may contain more properties such as intent, diagnosis, known uses etc. [Lukosch, 04]. In summary, a pattern can be considered as a kind of expertise about a problem and its solution in a specific context.

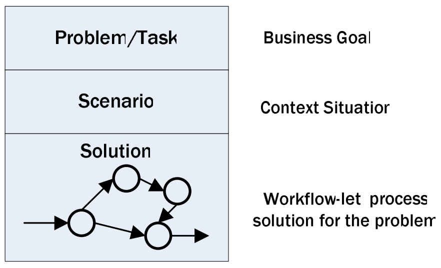

*Figure 1: Process pattern simple form*

The process pattern is the fundamental business knowledge representation structure. It is an extension of the pattern concept to the workflow domain; it can be thought of as a process solution for a specific business task or goal in a particular situation. It is a kind of empirical knowledge that describes how to solve a problem in specific situation.

$$pattern = \langle task, scenario, solution \rangle \qquad (1)$$

As illustrated in (1), *pattern.task* describes the business goal or the problem that needs to be solved. It is often a reference to a task instance. *pattern.scenario* portrays the most suitable environment for applying this pattern by some context terms. The scenario section identifies context information about the characteristic environment. *pattern.solution* provides a guideline to achieve the task. It is often a relatively cohesive process fragment provided for solving the pattern's problem, and consists of a series of *activities*, *transitions* and *data* dependencies.

Figure 2 gives an example of a process pattern in XML format. The <*Problem*> section uses <*goal*> to describe the pattern's objective. The goal uses the *domain* attribute to denote the question domain that it belongs to. In planning, the domain is used to filter patterns in different areas.

<*Scenario*> describes the situation in order to decide whether or not to apply this pattern. The context terms in the scenario are divided into two groups: positive factors and negative ones. <*PositiveFactors*> describe situations suitable for using this pattern, while negative factors describe situations in which the pattern should not be adopted. <*NegativeFactors*> has an *impactRatio* to measure the intensity of the dissuasive influence. When the current situation matches the pattern scenario, this pattern can be applied. Each <*ContextTerm*> has a *weight* attribute to describe the importance of this term in quantitative evaluation. C*ontextinfo* identifies the related context name, while *benchmark* is the reference value of the context information. *Evaluator* is the evaluation function for this context term. *Factor* is the custom

parameter of the term, which influences this term's evaluation sensitivity. The designer can calibrate factor values for specific preferences or fine-grained control.

*<Solution>* is a functionally independent process fragment, called a *<workflowlet>*. It consists of a set of related, cohesive actions for achieving the pattern objective. A workflowlet includes *<actions>*, *<Transitions>* and *<RelevantData>*. An action is an abstract activity that is a placeholder for a set of services matching the action description. In some cases the cardinality of the set is more than one. If an action has the *Goal* type, it means that the action is a sub-goal that needs replanning and refinement by other appropriate patterns. Replanning can be done at planning time or at runtime according to the pattern's *category* attribute.

*<Pattern>* has two categories: *Operational* and *Strategic*. When a *strategic* pattern is applied, all the *goal* type actions in the workflowlet must be expanded by replanning before execution. *Operational* patterns may also contain *goal* type actions, but their refinements are left to runtime.

*<Transitions>* describe the control and data dependencies between actions. *<RelevantData>* is a collection of parameters or variables used in the workflowlet.

```
 <Pattern     id="     4bec39c6-b727-4a2f-b83f-6f00669c2706"     category="operational"
name="Express procedure">
 <Problem>
  <Goal id="…" name="Car Damage Claim" domain ="business.finance.insurance.carDamage">
   …</Goal>
 </Problem>

 <Scenario>
   <PositiveFactors>
    <ContextTerm       weight="0.7"       contextinfo="ClientType"       benchmark="VIP"
    evaluator="eGreater" factor="0" />
    <ContextTerm       weight="0.3"       contextinfo="CreditLevel"       benchmark="85"
    evaluator="eGreater" factor="0.2" />
   </PositiveFactors>

   <NegativeFactors impactRatio="0.3">
    <ContextTerm weight="1" contextinfo="Amount" benchmark="3000" evaluator="eGreater"
    factor="0.2" />
   </NegativeFactors>
 </Scenario>

 <Solution>
   <Workflowlet id="…">
    <Actions>
      <Action id="…" type="BEGIN" name="" … />
     <Action id="…" type="NORMAL" name="Record" … />
     <Action id="…" type="GOAL" name="Assessment"…/>
     <Action id="…" type="NORMAL" name="Payment" …/>…
    </Actions>
    <Transitions/>
    <RelevantData/>
  </Workflowlet>
 </Solution>
</Pattern>
```

*Figure 2: A sample process pattern*

In brief, a workflowlet is the preferred process for solving the pattern problem in a situation conforming to the pattern scenario. When solving a sophisticated problem, the business goal may be decomposed into sub-goals through pattern matchmaking. After all sub-goals have been refined, the initial goal is mapped to several workflowlets. These workflowlets can be composed into a larger logical process definition.

The pattern structure satisfies the requirements mentioned at the beginning of this section. First, a pattern is adequate and suitable for representing procedural knowledge. In fact, a pattern naturally connects a process with its objective. Because there are no additional constraints for process specification, users can define a process in any way they like.

Second, a pattern is well structured and cost-effective. It is easy to program in applications. The system mainly uses goal and context information in the planning phase and finds appropriate process solutions for the execution phase. This is simpler than taking all details such as operators, preconditions, and effects into consideration from the beginning of planning.

Grain size or granularity is another important issue in knowledge representation. A pattern can describe knowledge at different levels of abstraction resolved to various levels of detail. A well-organized hierarchical pattern library minimizes visible complexity for the user and also provides adequate details for utilization. This feature makes it possible for existing planners to use patterns to generate workflows.

Moreover, patterns are relatively independent of one another. This is convenient for building and updating a pattern library incrementally. A pattern-oriented system can expand its knowledge by finding implicit patterns during repeated planning for the same result in specific scenarios.

### 2.2 Knowledge Base

Without adequate declarative and expressive information about the environment and the application, making sophisticated planning and scheduling decision become very difficult or even impossible. In order to build a flexible and intelligent grid workflow system, patterns alone are not enough. More ontologies and metadata that describe the grid environment and business activities are needed as the semantic basis for matchmaking, reasoning and planning.

A suggested knowledge base structure is shown in Figure 3. At the bottom, metadata and a shared ontology are defined in the form of OWL. The shared ontology describes basic elements of collaborative activity, including organization, task, event, goal, space, time and so on. This ontology and basic configuration are shared by the entire system. The application ontology and knowledge are built on top of the shared ontology. The application ontology describes the business entities (e.g., meta-context, goal, evaluator) and their relationships. Process patterns and supported goals are usually established by business experts. A policy includes rules, constraints and preferences, and is used for more elaborate process management.

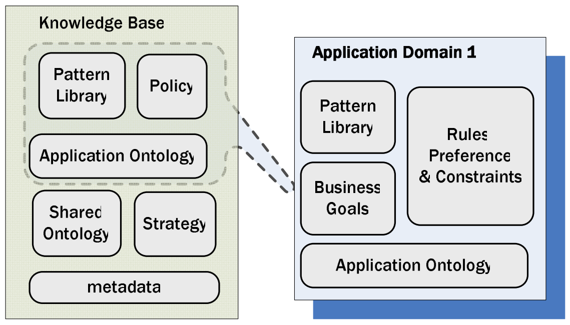

*Figure 3: Structure of knowledge base*

If the system needs to support a new application, the knowledge of this application domain should be added to the knowledge base, as shown on the right side of Figure 3. Theoretically, a system can support new business domains incrementally by adding new sets of application-specific knowledge and doing some configuration and administrative work.

Obviously, how well the system performs mostly depends on the quality of the knowledge base. Although the ontologies and patterns are created by workflow experts and business analysts, knowledge, especially patterns, still needs to be validated, updated and improved in practice. Further discussion of knowledge management, especially process pattern conflict detection and resolution, will appear in a sequel to this paper.

## 2.3 Pattern-oriented planning

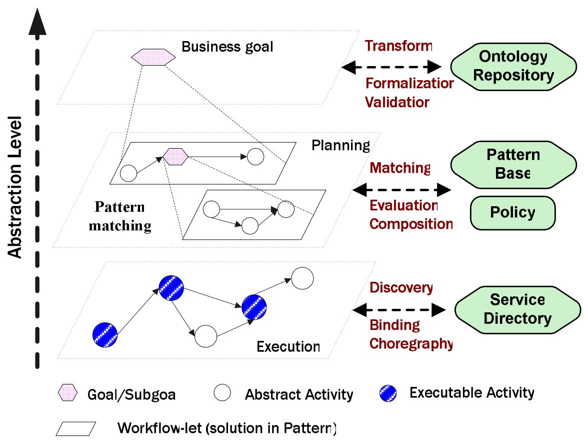

*Figure 4: Pattern-oriented planning*

Figure 4 outlines the basic procedure of pattern-oriented workflow generation. The knowledge base is shown decomposed into an ontology repository, a pattern base and a policy. The ontology repository contains representations of goals supported by system. The pattern-oriented planning approach has multiple phases: a goal transform phase, a matchmaking phase, a planning phase and an execution phase. A business goal is mapped to a suitable process pattern according to the current situation. Then, sub-goals in the pattern solution can also refined by process matching. In this way, the goal is incrementally refined to an executable process.

In detail, Figure 5 gives the basic algorithm for pattern-oriented planning.

First, the user submits a business goal in terms of application vocabulary. The system then parses the user request and transforms it into the system goal format and attaches some context information to the user request.

Second, the system checks the goal-context relationship in the policy, and finds the 'most common context' related to the goal. If there are some new contexts related to the goal, the system get their values from the corresponding context services. After that, system gets a declarative representation of the goal and a group of relevant context information.

Third, a partial workflow, which is the interim result of the planning process, is initialized and the goal node is sent to the planning queue.

Fourth, the system retrieves goal nodes from the planning queue in sequence, performs pattern matching and chooses suitable process patterns.

```
Algorithm: GenerateWorkflow ( BizGoal task)

Input: bizTask
Output: proc

<g, cxt>  Parser.transform(task)
<g, cxt>  getMostRelatedContext(g, cxt)
partialFlow   φ
queue.enQueue( new node(g))

while (node = queue.deQueue()) ≠ φ do
   if node.type = goal then
      p  MatchMaker.PatternMatching(node.goal, cxt)
      partialFlow.add(p.solution)
      if p.category = strategic then
         foreach (action in p.solution) then
            queue.enQueue(new node(action))
         end foreach
      end if
   end if
end while

proc  WorkflowComposer.compose(partialFlow)
return proc
```

*Figure 5: Planning algorithm*

Fifth, the solution of the applied pattern is added to the partial workflow. If the pattern is strategic, the system converts goal type actions to goal nodes and adds them to the queue. The system repeats these steps until the planning queue is empty.

Finally, the partial workflow may have several workflowlets after planning. The system needs to compose and orchestrate these workflowlets into an integrated process according to the inter-dependencies of the corresponding patterns. After that, the process result probably still contains unexpanded portions. These sub-goal actions will be refined by replanning at runtime.

Pattern matching includes domain filtering, goal matching and context matching. Every goal belongs to a specific domain. The system uses the target goal domain as a filter in the pattern library to reduce the scope of patterns. Then, all domain patterns are matched with the target goal to get candidate patterns that could solve the target problem. Finally, context matching measures the fit between the current situation and the scenarios in candidate patterns to decide whether or not to apply a pattern.

The context terms in pattern scenarios are organized into two categories: positive factors and negative factors. The former depict suitable circumstances for application of this pattern, while the latter characterize situations in which this pattern should not be adopted. A score can be calculated to measure the concordance between the current situation and the pattern scenario. This context-scenario matching evaluation is shown as (2):

$$s = r_P \sum_{i=1}^{n} w_P^i f_P^i(\vec{x}, \vec{p}) - r_N \sum_{j=1}^{m} w_N^j f_N^j(\vec{x}, \vec{p}) + b \qquad (2)$$

where $\vec{x}$ denotes current context data. $f_P^i()$ and $f_N^j()$ are specified as context evaluation functions. $\vec{p}$ contains additional parameters for the context evaluators. Each evaluator returns a decimal score in [0, 1] assessing the context suitability. $r_P \sum_{i=1}^{n} w_P^i f_P^i(\vec{x}, \vec{p})$ is the quantitative metric of the pattern's degree of suitability. $w_P^i$ and $w_N^j$ are the weights of the context evaluators, and $w_P^i \geq 0$ , $w_N^j \geq 0$ , $\sum_{i=1}^{n} w_P^i = 1,\ \sum_{j=1}^{m} w_N^j = 1$. The weights reflect the different importance of context terms in the pattern scenario. $r_P$ measures the intensity of the positive factors' influence. For convenience, we often let $r_P$=1. Similarly, $r_N \sum_{j=1}^{m} w_N^j f_N^j(\vec{x})$ assesses the degree of unsuitability of the pattern in the current environment. $b$ is a bias constant; we often let $b$=0.

The *activities* in *pattern.solution* are categorized into two types: atomic activity or abstract activity. An atomic activity can be directly bound to a resource for execution, whereas an abstract activity describes a business goal/sub-goal in the process.

Pattern-oriented hierarchy planning can be summarized in five steps as follows:

1) Parsing the business goal and collecting relevant context information.
2) Pattern matching, including domain filtering, goal matching and context matching. Choosing the winning pattern for the goal by evaluation scores.

3) Taking abstract activities in the pattern solution as new goals for pattern matching and planning.
4) Repeating matching until all pending abstracts node have been expanded.
5) Finally, creating a new process in compliance with the plan tree.

After planning, the workflow generated is assigned to appropriate services or resources by a scheduler for execution. The enactment engine can be a distributed job scheduler like DAGMan, a service choreography engine such as BPEL4WS, or another, user-developed enactment engine. The partially-specified portions of the process, i.e. goal type actions, will be refined at runtime on the basis of the context and the current state of the execution.

In this approach, planning and execution is decoupled into two independent phases. The design of interleaved planning and execution stages makes the system more flexible for choosing suitable execution schemes for specific business domains. More importantly, the planner does not need to understand technical-level operators. This will help the planner to be domain-independent.

The key issue that differentiates our approach from Pegasus is that we exploit high-level business knowledge to assist workflow generation. The knowledge and expertise of business processes are expressively represented as process patterns. As mentioned before, Pegasus lacks explicit knowledge. Its planner and the knowledge used in planning cannot be separated, so it is tightly bound to a specific application domain. In contrast to Pegasus, the pattern-oriented approach separates the planner from the knowledge. The process pattern is proposed as a knowledge representation structure and is used in workflow generation. It is domain-independent and these features enable the system to be more flexible and scalable.

Currently, semantic web research has provided some service choreography methods for process management, e.g. Web Service Modelling Language (WSML) [wsml, 06] and METEOR-S [Rohit, 04]. METEOR-S deploys QoS and preferences as constraints to turn the service composition problem into a constraint satisfaction problem. The process designers can bind Web Services to an abstract process based on constraints and generate an executable process. In contrast to the METEOR-S constraint satisfaction problem, our approach utilizes application knowledge to solve business tasks. From the user perspective, it is goal driven and more convenient to use. In addition, METEOR-S depends on semantic web service technology, whereas our approach can choose any suitable execution technology because the planning and execution stages are interleaved. In this way, the system can avoid being tightly coupled to underlying implementation techniques.

## 3 Pattern Modelling and Optimization

As the semantic basis of matchmaking, reasoning and planning in workflow generation, knowledge, such as the goal ontology, context and especially the process patterns, plays an essential role in the system. It is not only the foundation for workflow generation planning, but also promotes understanding between developers and workflow users.

The scenario is the most complex and essential part of a pattern. Although every variable in (2) has a clear meaning, it is still a big challenge for experts to define pattern scenarios. First, a complicated pattern scenario may involve many variables. It is very difficult for experts to accurately specify them depending only on empirical experience. Second, experts have to define appropriate context evaluators for each context term in a scenario, which requires considerable insight into business as well as mathematical capability. Finally, long-term, delicate manual modelling is costly.

In order to reduce the difficulties in pattern modelling and make knowledge more precise, we provide a three-step method: predefined context evaluators, expert fuzzy modelling and machine learning.

1) A set of evaluation functions $F = \left\{ f_i \left( \vec{x}, \vec{p} \right) \right\}$ are provided by the system. The modeller can directly choose proper functions rather than define them. These evaluators cover the most common numerical comparison and set operators. In addition to the context input $\vec{x}$, the evaluation function can also accept additional parameters $\vec{p}$ for fine-grained adjustment of the evaluation calculation.
2) Specialists build patterns using domain expertise, including choosing evaluators, determining parameters and weights, etc. For some abstruse parameters, experts can provide estimates.
3) After a period of running, the pattern knowledge can be calibrated by the results of machine learning. We use tagged history data as the training sample, and then update the parameters of scenarios by classifier training.

There are several advantages to this approach: First of all, the prior knowledge is utilized in pattern modelling. Although some parameters are estimates, the values given by experts are still approximately correct. This makes the subsequent training converge more rapidly. Second, because every variable in the context evaluation formula has a clear definition, the effect of changing the parameters is easier to understand for the user. This helps experts gain deeper insight into the business. For example, if a weight value grows substantially after training, it indicates that the experts probably underestimated the importance of the relevant context. Finally, this approach reduces difficulties and the workload of process pattern knowledge modelling.

### 3.1 Pattern scenario classifier

We first introduce the concepts of pattern cluster and scenario classifier.

**Definition 1**: A pattern cluster is a set of patterns which have the same task or goal. *PatternCluster(t)* denotes the pattern cluster of task *t*. The count of patterns in a pattern cluster is called the cluster cardinality.

**Definition 2**: A scenario classifier is a two-layer feed forward network for pattern clusters. The input units of layer 1 consist of the evaluation functions of pattern scenarios in *PatternCluster(t)* and a bias unit. Layer 2 contains sum units. The number of sum units is the cluster cardinality. There are weighted connections between the units in the two layers. The classifier takes a context vector as input and outputs scenario evaluation scores for patterns in the cluster. The scenario classifier of *PatternCluster(t)* is named *Classifier(t).*

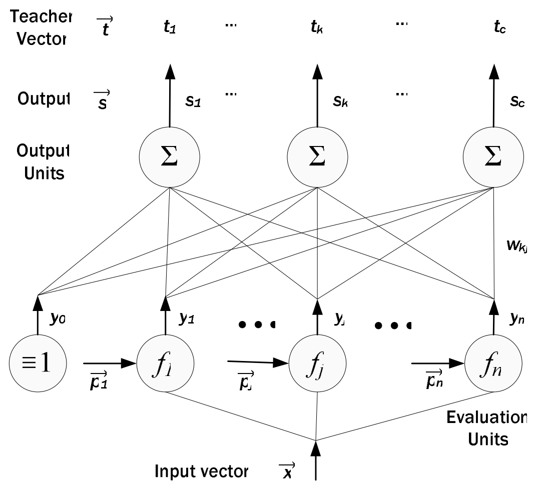

*Figure 6: Pattern scenario classifier*

As shown in Figure 6, the context $\vec{x}$ is the input vector of evaluation units $f_1$, …, $f_n$. In addition to $\vec{x}$, evaluator $f_j$ also accepts an additional parameter vector $\vec{p}_j$, and returns $y_j = f_j(\vec{x}, \vec{p}_j)$. The connection between the *k-th* output unit and *j-th* evaluation unit has the weight $w_{kj}$. The sum output unit calculates the dot product of the evaluators' outputs and the weights of relevant connections. Hence, the output of scenario classifier $s_k$ is by definition:

$$s_k = \sum_{j=1}^{n} w_{kj} f_j(\vec{x}, \vec{p}_j) + w_{k0} \qquad (3)$$

Obviously, (3) can be easily transformed to the previous scenario evaluation equation (2), so $s_k$ is the context-scenario matching score of the *k-th* pattern in the cluster.

## 3.2 Sample acquisition

In pattern matching, the system evaluates pattern suitability by matching the context with the pattern scenario. Candidate patterns are sorted in descending order of evaluation score in a queue called CP. The main purpose of machine learning is to generate more appropriate pattern sequences from context-scenario evaluation.

The sample data provides an ideal *CP* for specific context information. It determines the priority of applying patterns for achieving the cluster's task.

First, context data is generated by a simulation program. The simulation can be specified with different probability distributions on sampling intervals. Then, the system takes the context input, evaluates matching scores on corresponding pattern clusters and returns relevant *CP*s. Finally, business analysts review the data, and modify some *CP*s based on their experience.

Therefore, the sample data *TS* is composed of two parts: automatic data $TS_A$ and revised data $TS_M$. They have different significances; the automatic data calculated by the system confirms the correct aspects of prior knowledge, while the revised data modified by users reflects the unsatisfactory aspects.

## 3.3 Classifier training

In Figure 6, the outputs of the scenario classifier are the context matching scores for $p_1$, $p_2$, …, $p_c$. In training, the output of the feed-forward calculation $\vec{s}$ is compared with a target vector $\vec{t}$ . The training algorithm will adjust the classifier according to the comparison error to reduce the difference between $\vec{s}$ and $\vec{t}$ .

The back propagation (BP) algorithm is adopted in classifier training. It is a supervised learning technique based on gradient descent. BP is widely used in training neural networks and is successful in many fields. Based on back propagation, the training algorithm is listed in Table 1.

The classifiers are saved in hash table *CM*. The training sample *DS* is divided into two sets: training set *TS* and validation set *VS*.

The algorithm adopts a stochastic training protocol. At the beginning of each epoch, a training instance $td = < tk, \vec{x}, \vec{i} >$ is randomly selected. Then, the system performs a feed forward calculation and gets score vector $\vec{s}$ . After that, the components of $\vec{s}$ are sorted in descending order, returning a new score vector $\vec{s}\,'$ and subscript vector $\vec{i}\,'$. $\vec{i}\,'$ records the original positions of the components in $\vec{s}$ . If $\vec{i}\,'$ is not equal to $\vec{i}$ in the training data, the system would train *classifier(tk)* with this error.

```
ClassifierTraining (DS, tkList)
Input:   DS: sample data set
         tkList: task list for training.
Output:  null

 foreach tk in tkList            // tk is the goal/task
   $CM[tk] \leftarrow classifier(tk)$   // construct classifiers
 end foreach
 $\langle TS, VS \rangle \leftarrow partition(DS)$
 do
   $E_r \leftarrow 0$            // error ratio
   foreach td in TS
     $< tk, \vec{x}, \vec{i} > \leftarrow td$
     $\vec{s} \leftarrow forwardCalculation(CM[tk], \vec{x})$
     $< \vec{s}\,', \vec{i}\,' > \leftarrow sort(\vec{s}, DESC)$
     if $\vec{i} \neq \vec{i}\,'$ then
     $\vec{t} \leftarrow resortByIndex(\vec{s}\,', \vec{i})$
```

```
            $CM[tk] \leftarrow backpropagation\left(CM[tk], \vec{x}, \vec{t}\right)$
          **end if**
        **end foreach**
        **foreach** $td$ **in** $VS$
          $< tk, \vec{x}, \vec{i} > \leftarrow td$
          $\vec{s} \leftarrow forwardCalculation\left(CM[tk], \vec{x}\right)$
          $< \vec{s}\,', \vec{i}\,' > \leftarrow sort\left(\vec{s}, DESC\right)$
          **if** $\vec{i} \neq \vec{i}\,'$ **then** $E_r \leftarrow E_r + 1$
          **end if**
        **end foreach**
        $E_r \leftarrow E_r / |VS|$
      **until** $|\Delta E_r| \leq \varepsilon$ **or** $epoch > EPOCH\_MAX$
    $updatePatterns\left(CM\right)$     **// update process patterns**
```

*Table 1: Classifier training algorithm*

Because there are no teacher scores in the training data, the algorithm obtains a teacher vector $\vec{t}$ by resorting $\vec{s}\,'$ based on $\vec{i}$ . For example, supposing that $\vec{s} = \langle 0.56, 0.71, 0.33 \rangle$ ; after resorting we have $\vec{s}\,' = \langle 0.71, 0.56, 0.33 \rangle$ , $\vec{i}\,' = \{2, 1, 3\}$ ; if in the sample data $\vec{i} = \langle 3, 1, 2 \rangle$ , then the teacher vector is $\vec{t} = \langle 0.56, 0.33, 0.71 \rangle$ .

## 4 Case study

### 4.1 Overview

S-Power is a pilot application of a cooperative research project in the logistics field that combines process management and a variety of wireless technologies. Pattern-oriented workflow generation is the essential component of S-Power. This section uses the sub-task “determining transportation mode” in a logistics process to illustrate how to model and optimize the related process pattern knowledge.

First of all, we specified the most related contexts of the goal “determine transportation”. As shown in Table 2, the contexts are the most important factors for choosing the most suitable transport mode. The value ranges for each context were assigned empirically. *SpecialLevel* denotes the degree of difficulty of the transport. For example, special freight, such as oversize, overweight, fragile or liquid, will make the transportation more arduous and costly.

| | Sym | Range | Notes |
|---|---|---|---|
| ***ClientLevel*** | ***c*** | **0 ~ 100** | **Customer importance** |
| ***Distance*** | ***d*** | **2~3000 km** | **Comm. distance** |
| ***Urgent*** | ***u*** | **0 ~ 10** | **Urgency degree** |
| ***TimeLeft*** | ***t*** | **1 ~ 200** | **Time left for delivery** |
| ***InsuranceAmt*** | ***i*** | **100~ 100000** | **Value** |
| ***DangerLevel*** | ***a*** | **0 ~ 10** | **Hazardous material level** |
| ***SpecialLevel*** | ***s*** | **0 ~ 100** | **Laboriousness** |

*Table 2: Contexts in transportation*

After that, an expert constructed the process patterns for different transportation modes. According to practice, the *PatternCluster(lgs:transportation)* had 5 patterns, including special plane pattern $p_{sa}$ , air transport pattern $p_a$ , railway pattern $p_r$ , express truck pattern $p_{st}$ and road transport $p_t$ . Thus, the transportation problem involved 7 contexts and 5 patterns with dozens of parameters. It was very difficult for experts to specify every parameter accurately. Therefore, fuzzy evaluation and estimation were used in pattern construction.

As demonstrated in Table 3, experts gave fuzzy partitions of each context and estimated each evaluator's weight. Literals such as 'High' or 'Average' denote different context evaluation functions. Negative values are the weight values of negative factors in pattern scenarios, while a value in row '-' is the negative impact ratio.

| | SA. $p_{sa}$ | Air $p_a$ | Rail $p_r$ | Exp. $p_{st}$ | Truck $p_t$ |
|---|---|---|---|---|---|
| ***c*** | **Highest** | **High** | **Average** | **High** | **Average** |
| | **0.2** | **0.2** | **0.2** | **0.2** | **0.14** |
| ***d*** | **Far** | **Far** | **Average** | **Far** | **Near** |
| | **0.2** | **0.2** | **0.2** | **-0.5** | **0.15** |
| ***u*** | **Highest** | **High** | **Average** | **Higher** | **Low** |
| | **0.2** | **0.2** | **-0.5** | **0.2** | **0.14** |
| ***t*** | **Less** | **Little** | **Much** | **More** | **Average** |
| | **0.2** | **0.2** | **0.2** | **-0.5** | **0.15** |
| ***i*** | **High** | **Average** | **Average** | **High** | **Low** |
| | **0.2** | **0.2** | **-0.5** | **0.2** | **0.14** |
| **-** | **0.3** | **0.3** | **0.3** | **0.3** | **0** |
| ***a*** | **Low** | **High** | **Low** | **High** | **Low** |
| | **-0.5** | **-0.5** | **0.2** | **0.2** | **0.14** |
| ***s*** | **Average** | **High** | **High** | **Low** | **Average** |
| | **-0.5** | **-0.5** | **0.2** | **0.2** | **0.14** |

*Table 3: Primary pattern modelling*

The fuzzy concept gives experts a more convenient way to describe business expertise. When a problem is complicated, precise definitions will be very difficult or impossible. Consequently, using fuzzy and qualitative estimation is a feasible approach that brings stronger fault-tolerance to knowledge representation. For evaluating the context *ClientLevel,* the expert adopted three predefined evaluation functions: *FuzzyGreater* ( $f_>$ ), *FuzzyEqual* ( $f_=$ ) and *FuzzyLess* ( $f_<$ ) for describing highest, high and average rankings of client level. The fuzzy evaluators were implemented by Gaussian-like or Sigmoid-like functions. Fuzzy evaluators give the system more flexibility and smoother performance.

The fuzzy partition and qualitative estimations seem imprecise, but they were still approximately correct because they were based on prior knowledge. These parameters would be optimized in subsequent classifier training. Therefore, this approach provided a good balance between correctness and modelling workload.

| | ***Air transport*** $p_a$ | ***Rail transport*** $p_r$ |
|---|---|---|
| ***c*** | **0.2×$f_=$(6, 85)** | **0.2×$f_<$(0.6, 80)** |
| ***d*** | **0.2×$f_>$(0.01, 2100)** | **0.2×$f_=$(300, 1500)** |
| ***u*** | **0.2×$f_=$(1, 7)** | **(-0.5)×$f_=$(1, 6)** |
| ***t*** | **0.2×$f_=$(16, 60)** | **0.2×$f_=$(16, 140)** |
| ***i*** | **0.2×$f_=$(1400, 3000)** | **(-0.5)×$f_=$(1400, 3000)** |
| **-** | **0.3** | **0.3** |
| ***a*** | **(-0.5)×$f_>$(2, 6)** | **0.2×$f_<$(2, 6)** |
| ***s*** | **(-0.5)×$f_>$(0.3, 75)** | **0.2×$f_>$(0.3, 75)** |

*Table 4: Pattern evaluator parameters*

Because many contexts were involved in each pattern, the weights of the contexts were initialized uniformly, as shown in Table 3. Table 4 shows two pattern scenario's evaluators and parameters in detail. Only patterns | $p_a$ and | $p_r$ are listed due to space limitations.

Supposing context $\vec{x} = < c, d, u, t, i, a, s >$ , according to (2) and Table 4, the scenario matching score of railway transport pattern $p_r$ could be calculated as follows:

$$\begin{aligned} s(p_r, \vec{x}) &= 0.2 \cdot f_<(0.6, 80, c) + 0.2 \cdot f_=(300, 1500, d) \\ &+ 0.2 \cdot f_=(16, 140, t) + 0.2 \cdot f_<(2, 6, a) + 0.2 \cdot f_>(0.3, 75, s) \\ &- 0.3 \cdot [0.5 \cdot f_=(1, 6, u) + 0.5 \cdot f_=(1400, 3000, i)] \end{aligned}$$

| $\vec{x}$ | ***<84.7, 2132, 8, 106, 88902, 2.85, 87.5>*** |
|---|---|
| ***CP*** | ***4 3 5 2 1*** |

*Table 5: A training sample*

The sample data could be obtained as described in Section 3.2. Table 5 shows a sample training instance, where *CP=(4 3 5 2 1)* represents the pattern sequence $<P_{st}$ $P_r$ $P_t$ $P_a$ $P_{sa}>$. The scenario classifier training was offline learning implemented in Matlab 7.0.

## 4.2 Result analysis

First, the *classifier(lgs:transportation)* was built. It had 7 input components and 25 evaluation units. There were 5 output components in layer 2 representing the matching scores of $p_{sa}$, $p_a$, $p_r$, $p_{st}$, $p_t$ respectively. Training set *DS* contained 500 sample data. The first 350 samples comprised the training set *TS* and the remaining 150 the validation set *VS*. The result of training is shown in Figure 7.

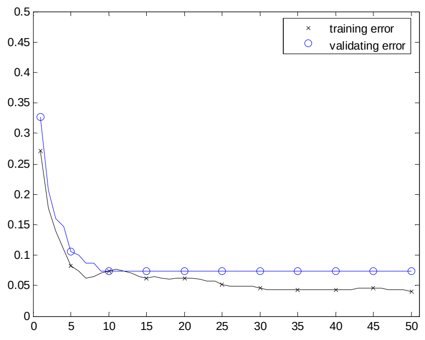

*Figure 7: Classifier training result*

The X-axis scale represents training epochs, and the Y-axis scale represents error ratio. The training error curve indicates the efficiency of the training algorithm, and the validation error curve reflects the generalization ability of the network. As shown in Figure 7, the training error ratio and validation error ratio decreased dramatically during the training. The initial errors of training and validation were both approximately 30%. After 5 epochs, the error ratios declined to about 10%. After 50 training epochs, the error ratios were both under 10%, reaching 4% and 7.3% respectively. A small amount of overfitting after the 10th epoch occurred because some constraints had been applied to the parameters and weights according to their attributes in the evaluation functions.

The initial error shows that the system performance couldn't meet the users' expectations with the primary knowledge modelling. However, the classifier training can could effectively optimize pattern knowledge and improve system performance. Furthermore, the training result could help the expert to better understand the business. For instance, in the primary modelling, the weights of positive factors in $p_a$ and $p_r$ were set to 0.2, and the negative factor weight was -0.5. After training, the weight values were changed as shown in Table 6.

| | ***Air transport*** $p_a$ | ***Rail transport*** $p_r$ |
|---|---|---|
| ***c*** | 0.17302 | 0.1784 |
| ***d*** | 0.22332 | 0.20448 |
| ***u*** | 0.21809 | -0.5361 |
| ***t*** | 0.21287 | 0.16371 |
| ***i*** | 0.18644 | -0.4531 |
| ***a*** | -0.5458 | 0.19612 |
| ***s*** | -0.4336 | 0.23947 |

*Table 6: Context weights after training*

The changes of weights indicated that different contexts had different importance in scenario matching. In the air transport pattern, the importance of the client level *c* was only 77% of the importance of the communication distance *d*, but the difference of the two weight values was just 0.05. Such a subtle difference is difficult for an expert to clearly specify. In the rail transport pattern, the weight of *SpecialLevel s* was 0.23947, apparently higher than the *TimeLeft t* weight 0.16371, which implies that 'transport difficulties' had a larger influence on railway transport decision than *TimeLeft*.

The weights in Table 6 are raw data from training. Before use for updating knowledge, these values needed post-processing, including necessary round-off, weight normalization and other adjustments. Finally, the parameters of the process patterns could be updated according to the training result.

## 5 Conclusions and future work

Dynamic workflow generation is a crucial problem in grid workflow. AI planning approaches have been deployed in some research projects, but a lack of explicit knowledge is still the main disadvantage of this method.

In this paper, the process pattern is proposed as a knowledge representation structure for business process knowledge. Based on the process pattern, we proposed a planning approach to automatically generate workflow by utilizing application-level knowledge. This approach provides a lightweight, efficient and cost-effective way to introduce knowledge into workflow generation. Working with an appropriate knowledge base, this approach can streamline the workflow optimization process and significantly improve system scalability.

Knowledge plays an increasingly important role in workflow generation and has a growing influence on system performance. The correctness and efficiency of pattern knowledge have become crucial. To that end, this paper has proposed a hybrid approach for pattern knowledge building and optimization. Experts construct a primary model using their domain knowledge. For a complex pattern scenario, they can estimate parameters, even using fuzzy partitions in context evaluation. Then classifier training on tagged history data can adjust the pattern scenario settings. Finally, we can update the pattern knowledge according to the training result.

This approach both reduces the difficulties of manual knowledge modelling and ensures the correctness and efficiency of the pattern knowledge. Compared with traditional BP neural networks, classifier training has the advantages of lower initial errors, rapid convergence, better training outcome and knowledge update.

We have deployed this approach in a prototype system. Our future work will include semantic-rich multi-modalities of process description, pattern conflict detection and resolution, more robust semantic reasoning for matching and some further implementation work. The pattern-oriented approach itself will also be refined and improved.

**Acknowledgements**

This work was supported by the National Natural Science Foundation of China under Grant No. 60736020 and the National High-Tech Research and Development Plan of China under Grant No. 2007AA010305.

## References

[Aalst, 03] Aalst, W., Hofstede, A.,Kiepuszewski, B., et al.: Workflow Patterns, Distributed and Parallel Databases, 14(3), July 2003, pp.5-51.

[Alexsander, 77] Alexander, C., Ishikawa, S., Silverstein, M.: A pattern language: towns, buildings, construction, Oxford University Press, 1977.

[Berman, 01] Berman, F., Chien, A., Cooper, K., et al.: The GrADS Project: Software Support for High-Level Grid Application Development. International Journal of High Performance Computing Applications (JHPCA), 15(4):327-344, 2001.

[Cao, 03] Cao, J., Jarvis, S., Saini, S., et al.: GridFlow: workflow management for grid computing, In Proc. 3rd Int. Symposium on Cluster Computing and the Grid, 12-15 May 2003, pp.198-205.

[Cheatham, 05] Cheatham, M., Cox, M.: AI planning in portal-based workflow management systems, In Proc. of the 2005 Conference onOptical Network Design and Modelling: Towards the broadband-for-all era, ONDM 2005, Febrary 07-09,2005, pp.47-52.

[Chung, 03] Chung, P., Cheung, L., Stader, J., et al.: Knowledge-based process management-an approach to handling the adaptive workflow, Knowledge-Based Systems, 16, 2003, pp. 149-160

[Deelman,03] Deelman, E., Blythe, J., Gil, Y., et al.: Mapping abstract complex workflows onto grid environments, Journal of Grid Computing, 2003, 4(1): 25-39.

[Dustdar, 05] Dustdar, S.: Reconciling Knowledge Management and Workflow Management Systems: The Activity-Based Knowledge Management Approach. Journal of Universal Computer Science, vol. 11, no. 4 (2005), 589-604

[Frey, 02] Frey, J., Tannenbaum, T., Livny, M., et al.: Condor-G: A Computation Management Agent for Multi-institutional Grids, Cluster Computing, vol. 5, 2002, pp. 237–246.

[Gil, 04] Gil, Y., Deelman, E., Blythe, J., et al.: Artificial Intelligence and Grids: Workflow Planning and Beyond, IEEE Intelligent Systems, 2004 Jan/Feb, pp.26-33

[GriPhyN, 00] The GriPhyN project, 2000, http://www.griphyn.org/

[Lukosch, 04] Lukosch, S., Schümmer, T.: Patterns for Managing Shared Objects in Groupware Systems, In Proc. 9th European Conference on Pattern Languages and Programs, Irsee, Germany, 2004.

[Montoto, 08] Montoto, P., Pan, A., Raposo, J., et al.: A Workflow Language for Web Automation. Journal of Universal Computer Science, vol. 14, no. 11 (2008), 1838-1856

[Musicante, 06] Musicante, M., Potrich, E.: Expressing Workflow Patterns for Web Services: The Case of PEWS. Journal of Universal Computer Science, vol. 12, no. 7 (2006), 903-921

[Oinn, 04] Oinn, T., Addis, M., Ferris, J., et al.: Taverna: A tool for the composition and enactment of bioinformatics workflows, Bioinformatics Journal, vol. 20(17) pp.3045-3054, 2004.

[Pegasus, 04] The Pegasus project, 2004, http://pegasus.isi.edu/

[Rohit, 04] Aggarwal, R., Verma, K., Miller, J., et al.: Constraint Driven Web Service Composition in METEOR-S, In Proc. IEEE Int. Conf. on Services Computing, September 15-18, 2004, pp.23-30.

[wsml, 06] Web Services Modeling Language, 2006, http://www.wsmo.org/wsml/