A Pattern-Oriented Planning Approach for Grid Workflow Generation

Shaohua Zhang, Jinlei Jiang, Meilin Shi, Xiaomin Zhao
Department of Computer Science, Tsinghua University, China
{zshua, jjlei, shi, zxmin}@csnet4.cs.tsinghua.edu.cn

Abstract

As the key enabling technology for grid perspective, grid workflow receives more and more attention of researchers and has been applied to many projects. Dynamic workflow generation is considered as one of the most important issues in grid workflow researches, because it has essential impact on system usability, flexibility and robustness. Currently, exploiting AI planning technology for workflow construction is emerging in some researches. Although it works well in specific problem domain, the lack of explicit knowledge jeopardizes its scalability and significance in grid workflow. In this work, we put forward process pattern as a vehicle for knowledge representation to capture process expertise in business level, and based on it, a planning approach is proposed for automated workflow generation; Compared with existing planning method, our pattern-oriented approach minimizes user-visible complexity and makes system more scalable and flexible by utilizing explicit knowledge support.

1. Introduction

With the ever-increasing popularity of grid computing, grid workflow, which is considered as 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, grid workflow system comprises a group of tasks or jobs and assigns them to suitable services or resources for execution. Most existing grid workflow systems build entire process specification before execution on grid environment. Process specification can be constructed based on simulation or performance prediction, or even manually. Because all information needed in 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. resource physical location, service endpoint) for defining process specification. Moreover, the full-ahead plan often raises exceptions owing to hardware failures or resource usage policy changes in runtime. The situation would get even worse when workflow duration spans several days or weeks. To deal with the issues in large-scale, complex and dynamic environment like grid, automatic workflow construction becomes necessary. The demand is growing in visibility as grid computing shifts from scientific community to business context.

Recently, utilizing artificial intelligence (AI) techniques for automated workflow generation emerges in some researches. This approach offers several advantages. Firstly, it eases users' burden and improves system usability. Some potential users who are afraid of grid complexity will be encouraged to use grid. Secondly, workflow gets more fault-tolerable and more rapid in response to exceptions, because when error occurs, system could handle it promptly by reproducing an alternative process plan.

Pegasus [3][13] is a typical workflow system that integrates AI planning techniques for workflow construction. It is used in GriphYn [14] developed by the University of Southern California. Pegasus can take user's highly specified desired results and then generates a valid workflow for execution. In more details, Pegasus takes desired data product as “goal state”, and takes the application components as “operators”. Like typical AI planning system, Pegasus receives inputs of a current state of 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 [15] to provide the target data product.

The main disadvantage of this planning approach is lack of explicit knowledge. More exactly, its planner and knowledge used in planning are mixed together. Therefore, when description becomes abstract or contains less detail, the planner is more difficult to yield a good planning result. On the contrary, when
description goes more exhaustive, the result maybe better, but the planner has to understand more complexity. In a word, the planner is tightly bound with specific domain. This drawback jeopardizes domain independence of planner and may harm system scalability. With the problem scale grows, searching space will dramatically increase to huge size and certainly overwhelms the planner.

As Yolanda Gil et al. [1] 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 process pattern as a knowledge representation structure to capture process expertise in business level. Based on process pattern, a planning approach is proposed to automatically generate workflow. User could submit the business goal in application terms, and system generates executable workflow that can achieve the goal specified.

The rest of this paper is organized as follows: section 2 details the pattern oriented planning approach. Firstly, section 2.1 explains the concept of process pattern and why we choose it as basic knowledge element; then, section 2.2 introduces the knowledge base used in system; section 2.3 discusses the steps of planning approach for automated workflow construction. A case study is explained in Section 3. After that, section 4 compares our approach with related works. Conclusions and future work are covered in Section 5.

2. Pattern oriented planning

In order to develop scalable, domain-independent mechanisms for dynamic workflow generation, knowledge should be integrated into grid workflow management system. In grid workflow, 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 researches. 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 performs MPI/PVM jobs or data-intensive analysis task, which is more related to underlying resources, such as computation, storage capacity. Nowadays, grid workflow broadens its field and begins to supports business process, i.e. procurement, supply-chain, and coordination of activities or services in different organizations. In this case, application level knowledge probably plays more significant role than environment knowledge in coordination of workflow generation, execution and exception handling.

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

2.1 Process Pattern

There are some requirements for knowledge representation for grid workflow management, especially workflow generation. Firstly, it should be suitable for representing procedural knowledge. In workflow domain, the most important knowledge is procedural knowledge; the representation approach must be adequate for describing it. Secondly, it should be efficient for use. We integrate it in application so the representation structure must be convenient to program and manipulate. Thirdly, it should be easy to understand so that user from 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 network, rule-based method etc. are not suitable. Rule-based method is not fit for describing procedural knowledge; frames and semantic network are hard to program into system; while predicate logic seems too intricate for user and business expert.

In this work, process pattern is proposed for knowledge representation in grid workflow generation process. Pattern originally comes from architecture domain, which "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 [7]". The basic pattern structure, as shown in Figure 1, includes three parts: problem, scenario and solution. Problem is a description of task to be handled; Scenario describes context information of environment that the problem lays; Solution provides a guideline to perform the task or gives answer to the problem. A practicable pattern may contain more properties such as intent, diagnosis,
Figure 2. A sample process pattern

known uses etc. [8]. In summary, pattern can be considered as a kind of expertise about problem and its solution in specific context.

Process pattern is an extension of pattern in workflow domain, it can be thought as process solution for specific business task or goal in particular situation. Problem part is the business goal or the problem need to be solved. Scenario section identifies context information about the environment characteristic. In solution portion, a relative cohesive process fragment is provided for solving pattern’s problem.

Figure 2 gives an example of process pattern in Xml format. <Problem> section uses <goal> to describe the pattern’s objective. Goal use domain attribute to denote the question domain that it belongs to. In planning, domain is used to filter patterns in different area;

<Scenario> describes a situation to decide applying this pattern or not. Context terms in scenario are divided into two groups: positive factors and negative ones. <PositiveFactors> express suitability of using this pattern, while negative factors describe the situation in which pattern should not adopted. <NegativeFactors> has an impactRatio to measure the intensity of denying influence. When current situation matched with the pattern scenario, this pattern can be applied. Each <ContextTerm> has a weight attribute to describe the importance of this term in quantitative evaluation. Contextinfo identifies the related context name, while benchmark is the reference value of context information. Evaluator is the evaluation function for this context term. Factor is the custom parameter of the term. It influence this term’s evaluating sensitivity. Designer can calibrate factor value for specific preference or fine-granularity control.

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

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

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

In short, workflowlet is the most preference process for solving pattern problem in situation conforming to the pattern scenario. When solving a sophisticated problem, business goal may be decomposed into some sub-goal through pattern matchmaking. After every sub-goal refinement, initial goal is mapping to several workflowlets. These workflowlets can be composed into a bigger logic process definition.

Pattern structure can satisfy the requirements mentioned in the beginning of this section. Firstly, pattern is adequate and suitable for representing procedural knowledge. In fact, pattern connects process with its objective naturally. Because there are no additional constraints for process specification, hence, user can define process by any way they like.

Secondly, pattern is well structured and cost-effective. It is easy to program in application. System mainly uses goal and context information in planning phase and finds appropriate process solution for execution phase. It is simpler than to take all details such as operators, preconidtion, and effects into consideration from the beginning of planning.

Grain size or granularity is another important issue concerned in knowledge representation. Pattern can describe knowledge of different abstract levels in various resolution details. Well-organized hierarchical pattern library minimizes visible complexity for user and also provides adequate details for utilization. This
feature makes it possible for existing planners to use patterns to generate workflow.

Moreover, pattern is relative independent with each other. It is convenient for building and updating pattern library incrementally. Pattern oriented system can expend its knowledge by finding implicit patterns during repeated same planning result in specific scenarios.

2.2 Knowledge base

Without adequate declarative expressive information of environment and application, making sophisticated planning and scheduling decision become very difficult or even impossible. In order to build flexible and intelligent grid workflow system, pattern alone is not enough. More ontology and metadata, which describe grid environment and business activities, are necessary as the semantic basis of matchmaking, reasoning and planning.

A suggested knowledge base structure is shown in figure 3. In the bottom, metadata and shared ontology are defined in form of OWL. Shared ontology is about basic elements of collaboration activity, including organization, task, event, goal, space, time and so on. These ontology and basic configuration are shared in the entire system. Application ontology and knowledge are built on shared ontology. 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. Policy includes rules, constraints and preferences. It is used for more elaborate process management.

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

 Obviously, how well the system performs mostly depends on how well the knowledge base is. Although ontology and patterns are created by workflow experts and business analysts, knowledge, especially patterns, still need to validate, update and improve in practice. Furthermore, knowledge management, especially process pattern conflict detection and resolution, is in the sequel to this paper.

2.3 Pattern-oriented planning

Figure 4 outlines basic procedure of pattern oriented workflow generation. Knowledge base is shown separately as ontology repository, pattern base and policy. Ontology repository contains representation of goals supported by system. Pattern oriented planning approach has multi-phases: goal transform phase, matchmaking phase, planning phase and execution phase. Business goal is mapping to suitable process pattern according to the situation at that time. Then, sub-goals in pattern solution can also refined by process

![Figure 3. Structure of knowledge base](image3)

![Figure 4. Pattern-oriented planning](image4)

**Algorithm: GenerateWorkflow (BizGoal task)**

```plaintext
<g, ctx> ← Parser.transform(task)
<g, ctx> ← getMostRelatedContext(g, ctx)
partialFlow ← φ
queue.enQueue( new node(g))

while (node = queue.dequeue())≠ φ do
  if node.type = goal then
    p ← MatchMaker.PatternMatching(node.goal, ctx)
    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](image5)
goal is incrementally refined to an executable process.

In details, figure 5 gives the basic algorithm for pattern-oriented planning.

Firstly, user submits business goal in terms of application vocabulary. System then parses user request and transforms it into system goal format and some context information attached with user request.

Secondly, System checks goal-context relationship in policy, finds ‘most common context’ related to the goal. If there are some new contexts related to the goal, system get their value by the corresponding context services. After that, system gets a declarative representation of the goal and a group of relevant context information.

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

Fourthly, system gets goal node from planning queue in sequence, performs pattern matching and chooses suited process pattern.

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

Finally, partial workflow may have several workflowlets after planning. System needs to compose and orchestrate these workflowlets into an integrated workflow after planning queue is empty. These sub-goal actions will be refined via re-planning at runtime.

Pattern matching includes domain filtering, goal matching and context matching. Every goal has a specific domain that it belongs to. System uses target goal domain as a filter in pattern library to reduce scope of patterns. Then, all domain patterns are matched with target goal to get candidate patterns, which can solve target problem. Then, context matching measures the fitness between current situation and scenario in candidate pattern to decide applying this pattern or not.

Context terms in pattern scenario are divided into two groups: positive and negative. Context terms in PositiveFactors identify the conditions to adopt this pattern, while context terms in NegativeFactors describe the situation which is not suitable for applying the pattern.

Let $S$ denotes matching score. $i, j$ denote the positive and negative context terms in pattern scenario respectively, where $0 \leq i \leq n$ and $0 \leq j \leq m$. The weight of positive context term is denoted $w_i$. We assume $c_i$ is the positive context name and $b_i$ is its reference value. $\sigma_i$ denotes the normalization factor of positive context term; $f_i()$ is the evaluator for the context term. We also use notations $w_i, c_i', b_i', \sigma_i, f_i(), r$ to denote the similar meanings for negative context terms respectively and $r$

denotes the impact ratio of negative terms. Context matching can be quantitative calculated by formula (1).

$$S = \sum_{i=1}^{n} w_i f_i(c_i, b_i, \sigma_i) - \sum_{j=1}^{m} w_j f_j(c_j, b_j, \sigma_j)$$ (1)

Additionally, the fitness formula can be defined according to different requirements.

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

In this approach, planning and execution is decoupled as two independent phases. The design of interleaving the planning and execution stages makes system more flexible for choosing suitable execution scheme for specific business domain. More important, it is not required for planner to understand technical level operators. It will help planner be domain independent.

3. A Case Study

We have introduced pattern oriented planning approach in above sections. To make it more clearly, we make a case study in this section.

3.1 Scenario

The business of fictional Insurance company SafeLife includes insurances of disease, property, and car damage. For better customer experience, company decides to introduce many personalized services for individuals. For example, company can choose different claim procedure according to information of customer and claim. Even process stages like investigation or assessment have diverse steps in different context.

Suppose company provides two claim procedures for car damage: normal procedure and express procedure. Just as its name implies, express procedure is simpler and more efficient than the normal one. Company is prone to adopt express way for prestigious client, who...
has excellent credit level, or when claim amount is small. This measurement not only reduces process time and improves customer satisfaction but also lowers running cost. Figure 6 depicts the scenario of this case.

Such consideration discussed above can be captured and expressed by process pattern. Suppose all such knowledge has been built in company knowledge base, including goal, pattern, context and their relationships. Figure 2 in section 2.1 gives an abbreviate version of express procedure pattern.

3.2 Solution

This section will illustrate the pattern-oriented planning approach to support such personalized service via automatically generating process for different situation.

```xml
<Goal id="..." name="Car Damage Claim" domain="/business.finance.insurance.carDamage"/>
</Goal>
```

```xml
<ContextInfo id="..." name="ClientId" type="xsd:string" value="2589541" />
<ContextInfo id="..." name="Amount" type="xsd:int" value="3800" />
```

**Figure 7. After business goal transformation**

Firstly, user fills a claim form and submits it to company system. The form is regarded as application goal and is transformed into goal and some ContextInfo as figure 7 shown.

```xml
<ContextInfo id="..." name="CreditLevel" type="xsd:int" value="80" />
<ContextInfo id="..." name="ClientType" type="xsd:string" value="NORMAL" />
```

**Figure 8. Get new context related to the goal**

Secondly, System checks goal-context relationship in policy and find two other ContextInfo related to the “car damage claim” goal: CreditLevel and ClientType. System obtains these contextinfo values by the related context service. The two contexts are shown in figure 8.

To reduce the searching space of pattern, system uses the “business.finance.insurance.carDamage” domain to filter the pattern library. Then, system performs goal matching to get candidate patterns. After matching the target goal with the problems of candidate patterns, system will get two patterns matched with the “Car Damage Claim” goal: Normal Procedure Pattern and Express Procedure Pattern.

Then System performs context matching for each pattern. We here only take Express Procedure Pattern context matching for example. The scenario is shown in figure 2. It explains that if customer is a VIP or user’s credit is very good, insurance company prefers to take simple procedure for user’s claim. However, if the claim amount is high, insurance will consider turning to normal procedure, which are often more strict and time-consuming.

User can specify evaluator function or choose predefined ones for each context term evaluation. We currently use membership function in fuzzy logic to define evaluator for better adaptive performance. Evaluator eGreater is defined as formula (2) and (3), where \( c \) denotes context value and \( b \) denotes the benchmark value. The custom factor is denoted as \( \sigma \). Specially, when \( \sigma = 0 \) and \( c = b \), it returns 1; when \( \sigma = 0 \) and \( c \neq b \), it always returns 0.

\[
eGreater(c,b,\sigma) = \text{gaussmf}\left(\frac{c}{b},\sigma\right);
\]

\[
\text{gaussmf}(x,\sigma) = \begin{cases} 
\exp(-(x-1)^2 / 2\sigma^2), & x \leq 1 \\
1, & x > 1
\end{cases}
\]

We can calculate matching score of Express Procedure Pattern according to formula (1).

\[
S = \sum_{i=1}^{n} w_i f_i(c_i,b_i,\sigma_i) - \sum_{j=1}^{m} w_j f_j(c_j,b_j,\sigma_j)
\]

\[
= 0.7*1 + 0.3*0.9577 - 0.3*1 = 0.6873
\]

Similarly, we calculate matching score of Normal Procedure Pattern. The score is 0.2127. Hence, system chooses Express Procedure Pattern for this claim. The solution of the pattern is also shown in figure 2.

The workflowlet of the express pattern is an executable process definition. Because the pattern is operational type, the abstract actions in workflowlet will be refined in execution. When execution reaches at "GOAL" type activity - Assessment, system sends it to planner as a sub-goal for replanning. At that time, matchmaker probably needs to collect updated context or the current state of execution for matching. The new generating workflowlet of "assessment" will process as a subflow of upper workflow.

4. Comparison with related works

WMP van der Aalst proposed "workflow pattern"[9] for systematically analyzing functionalities of workflow management system (WMS). Although process pattern seems very similar with workflow pattern, they are essentially different. Workflow pattern is mainly about workflow functions and is organized according to control flow structures, e.g. sequence, parallel split. Furthermore, workflow pattern has no context or solution part in its description. However, process pattern is a kind of business expertise which is designed to represent knowledge for dynamic workflow generation. Process pattern provides process solution for specific user task or goal in particular scenario, e.g. express claim procedure in insurance. Problem, context and solution are absolutely necessary in process pattern.

In some researches [10][12], system has a predefined process library for improving planning efficiency. Process pattern is also substantially distinguished from process library. Firstly, process
pattern is the synthesis of business expertise and workflow knowledge. It is defined in application domain, whereas process library is a collection of workflow definitions and is defined in technical level. Secondly, process pattern is a dynamic description in multi-dimension including problem, scenario and solution. The current state of environment plays an important role on choosing process pattern. On the contrary, process in library is a static description of activities and their dependencies and is used as building block. Context changes have no direct influence on which process will be used.

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 process are expressively represented in form of process pattern. As mentioned in section 1, Pegasus is lack of explicit knowledge. Its planner and knowledge used in planning can not be separated, so it is tightly bound to specific application domain. Compared with Pegasus, pattern oriented approach separates planner from knowledge. 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 more flexible and scalable.

Currently, the semantic web researches derive some services choreography methods for process management, e.g. Web Service Modeling Language (WSML) [11] or METEOR-S [12]. METEOR-S deploys QoS and preferences as constraints to turn the service composition problem to a constraint satisfaction problem. The process designers can bind Web Services to an abstract process based on constraints and generate an executable process. Compare with METEOR-S constrains satisfaction problem, our approach utilizes application knowledge to solve business task. From user perspective, it’s goal driven and is more convenient for use. In addition, METEOR-S depends on semantic web service technology, while our approach can choose any suitable execution technology because the planning and execution stages are interleaved. In this case, system can avoid being tightly coupled with underlying implementation techniques.

5. Conclusion and future work

Dynamic workflow generation is a crucial problem in grid workflow. AI planning approach has been deployed in some research project. But lack of explicit knowledge is still the main disadvantage of this method.

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

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

References