Federated Enactment of Workflow Patterns

Gagarine Yaikhom[1], Chee Sun Liew[1], Liangxiu Han[1], Jano van Hemert[1], Malcolm Atkinson[1], and Amy Krause[2]

[1] School of Informatics, University of Edinburgh, Edinburgh, United Kingdom
[2] EPCC, University of Edinburgh, Edinburgh, United Kingdom
gyaikhom@inf.ed.ac.uk

Abstract. In this paper we address two research questions concerning workflows: 1) how do we abstract and catalogue recurring workflow patterns?; and 2) how do we facilitate optimisation of the mapping from workflow patterns to actual resources at runtime? Our aim here is to explore techniques that are applicable to large-scale workflow compositions, where the resources could change dynamically during the lifetime of an application. We achieve this by introducing a registry-based mechanism where pattern abstractions are catalogued and stored. In conjunction with an enactment engine, which communicates with this registry, concrete computational implementations and resources are assigned to these patterns, conditional to the execution parameters. Using a data mining application from the life sciences, we demonstrate this new approach.

1 Introduction

In large-scale service-oriented frameworks, a single application uses several distributed services and computational resources. Take, for instance, the distributed data mining and integration for flood predictions, where data from various meteorological stations are analysed using a wide variety of distributed computational resources and services. In order to express the components of such applications, and the interactions between these components, application programmers use high-level workflow languages. Over the past few years, several workflow expression languages and workflow management infrastructures have been developed to simplify application development, while also improving the application performance. See [5], [1], [13], and [8].

Riehle and Züllighoven [16] argue that software patterns permeate the entire process of software development, and that they are the key to providing multiple levels of programming abstraction. Using the concept of algorithmic skeletons, Cole [6] has similarly highlighted the importance of algorithmic abstraction within the context of parallel computing. In this paper, we address two research questions that are crucial to the higher-order abstraction of functionality from implementation. In particular, we enhance *workflow patterns* by addressing the following questions: 1) how do we abstract and catalogue recurring workflow patterns?; and 2) how do we facilitate the runtime optimisation of mapping from workflow patterns to actual services and resources during enactment?

P. D'Ambra, M. Guarracino, and D. Talia (Eds.): Euro-Par 2010, Part I, LNCS 6271, pp. 317–328, 2010.

The novelty of our approach lies in our ability to separate *abstract semantics* (the functional aspects of a pattern) from the range of *concrete semantics* (the actual implementation using various middleware services) that are possible; our ability to assign an appropriate concrete meaning to a given abstract semantics depending on the nature of the execution environment; and the *federated execution* model, which allows execution platforms to transparently partition and concurrently execute a workflow using a federation of distributed resources.

In section 2, we describe our proposed pattern-oriented workflow abstraction model, and discuss the mechanisms by which concrete semantics are assigned to workflow patterns at runtime. This approach is illustrated experimentally in section 3, using a data mining case study. In section 4, we briefly discuss our approach in relation to previous work; and conclude this paper in section 5.

2 The ADMIRE Architecture

The ADMIRE (Advanced Data Mining and Integration Research for Europe) project aims to provide domain specialists and data mining experts with a clear abstraction for deploying abstract workflows over distributed computational frameworks. Multiple levels of abstraction are the key to the ADMIRE architecture. The design of the architecture is user-centric, and is driven by three types of users [4]: 1) *domain experts* who are users with an acute understanding of the concepts and logic underlying an application domain (for instance, financial analysts and investors); 2) *algorithm experts* who specialise in the algorithms and techniques for carrying out computational tasks identified by the domain experts (for instance, data mining and integration algorithm specialists); and 3) *data-aware computing engineers* who specialise in various computational and data-storage frameworks.

2.1 Separation of Design and Execution

The novelty of the ADMIRE architecture lies in the manner of separation of concerns. In traditional workflow frameworks, the separation of concerns is restricted to the step-wise refinement of design. Domain experts draw the tasks and inter-task dependencies that are required to realise an application, and this is translated into the appropriate algorithms by the algorithm experts. The algorithms are then processed by the computing engineers and the resulting optimised workflow is sent to the resources for execution. Although this development process is generally acceptable for applications and environments which are practically static, where the resources and the requirements seldom change, it is not suitable for dynamic environments. The fact that users at the lower-end of the development process are tied to the specification supplied by the higher-level users makes it extremely difficult to choose the best design and implementation at the lower-end (during enactment). Furthermore, later improvements in the lower-end of the development process can only be accommodated by revision of the design, starting with the domain experts.

In the ADMIRE architecture, these restrictions are circumvented by choosing a development process where the design of the application is completely decoupled from its execution. This is achieved by using patterns as the means for expressing components and their coordination, while the meaning of the pattern is dynamically assigned at runtime depending on the requirements of the application and the resources available to the execution environment.

The two main phases for running a workflow in the ADMIRE architecture are: 1) *process engineering*, which is the design phase; and 2) *enactment*, which is the execution phase. During process engineering, a workflow is composed using well-defined patterns. This is done using DISPEL (a Data-Intensive Systems Process Engineering Language [4]), which abstracts workflow patterns as functions. During enactment, the patterns are interpreted and expanded, transforming an abstract workflow composition into actual services and inter-service interactions.

2.2 Abstract and Concrete Semantics

The *abstract semantics* of a workflow pattern defines its expected external behaviour, its black-box meaning independent of the implementation details. The concrete semantics, on the other hand, defines the internal behaviour of a workflow pattern as it should be executed, which depends on the specific implementation of the workflow pattern and the execution environment available to the enactment platform when the workflow composition is submitted.

```
use uk.org.ogsadai.SQLQuery;
function PE1(String query, String resource)
PE(<> => <Connection data>)
{
    SQLQuery query = new SQLQuery;
    |- query -| => query.expression;
    |- resource -| => query.resource;
    return PE(<> =>
        <Connection data = query.data>);
}
```

```
function PE5()
PE(<Connection input> => <Connection output>)
{
    ImageToMatrixActivity itma = new ImageToMatrixActivity;
    MedianFilterActivity mfa = new MedianFilterActivity;
    itma.output => mfa.input;
    return PE(<Connection input = itma.input> =>
        <Connection output = mfa.output>);
}
```

Fig. 1. Example DISPEL functions

In DISPEL, processing element types represent abstract semantics; whereas, the concrete semantics is expressed as a directed graph of components, where the edges represent the flow of data between components. Each component of this directed graph is either a composite processing element (in which case, we have a higher-order hierarchical abstraction), or a concrete processing element with an implementation of a computational object. In the ADMIRE context, we use OGSA-DAI activities for the concrete implementations. Each of these composite and concrete processing elements are catalogued using wrapper functions. Thus, every abstraction is written as a DISPEL function, and every function returns either a processing element instance, or a composition of components. Figure 1 lists two example DISPEL functions: PE1 returns an OGSA-DAI SQL query activity; whereas, PE5 returns a composition with abstract components `ImageToMatrixActivity` and `MedianFilterActivity`.

2.3 Semantic Registry

Semantic registries are service providers with facilities to import, export, and search workflow patterns. All abstract semantics are defined and registered with a semantic registry. In addition to this, possible concrete semantics for each of these are also defined and registered as DISPEL functions.

On the user front, semantic registries expose workflow patterns by exposing the processing element types. This allows users to search and re-use patterns during process engineering. In the prototype implementation, relationships between processing element types are captured using ontology; and the searches are carried out using the SPARQL query language[1].

Semantic registries also play a crucial role during workflow execution. They provide various mechanisms for realising the abstract semantics of a workflow pattern by mapping them to suitable *concrete semantic* objects (either compositions or activities). This mapping is carried out by exporting from the registry the function definition that corresponds to the chosen concrete semantics, where the choice of a particular mapping is dictated by the properties associated with the pattern, and the execution parameters (e.g., environment features).

2.4 Enactment Gateways

In the ADMIRE architecture, the execution environments are not exposed for immediate access; instead, they are concealed behind a service provider called the *enactment gateway*. The enactment gateway has been assigned the task of interpreting and executing workflow compositions written in DISPEL.

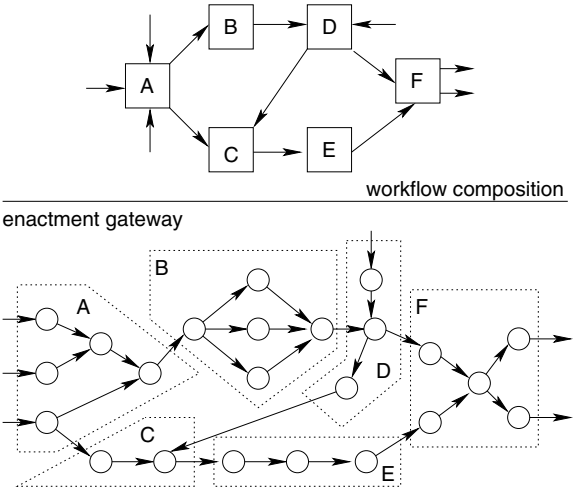

Fig. 2. Expansion of workflow patterns

[1] SPARQL query language, `http://www.w3.org/TR/rdf-sparql-query/`

In figure 2, we show how an enactment gateway expands patterns (shown in rectangles) into concrete semantic objects (shown in circles). The workflow composition received by this enactment gateway uses six different patterns: A, B, C, D, E, and F. The data dependencies between these patterns are given by the corresponding directed edges. Before interpreting this workflow composition, the enactment gateway gathers information about the execution environment that is currently available to it (resource availability, load, communication latency, etc.). This information is then used to query the semantic registry. Eventually, depending on the execution parameters received from the enactment gateway, the semantic registry returns the most appropriate pattern expansions as DISPEL functions. By executing each of these functions, recursively when higher-order abstractions are involved, the workflow composition is expanded to a form where all of the components are computational objects (e.g., OGSA-DAI activities).

2.5 Federated Execution

Higher-order abstraction leads to *federated execution*. A workflow composition submitted to an enactment gateway could be partitioned into sub-workflows, which are then delegated to a federation of enactment gateways. This facilitates execution of a complex workflow composition using various computational resources in parallel, thus increasing the throughput. Since we are interested in large-scale data intensive computations, the ability to partition a workflow composition presents an opportunity for significant performance improvement. Furthermore, since the *primary enactment gateway* (the enactment gateway which received the workflow composition) handles this internally, the entire federated execution is concealed from the domain experts.

Let us consider the workflow expansion shown in figure 2. This expansion has six partitions, each of which can be submitted to a federation of enactment gateways associated with the primary gateway. This process of partition and submission continues until an enactment gateway is ready to begin the execution: instantiation of the services and establishment of communication links between these service instances.

Optimisation. Since each enactment gateway receiving a partition can treat its sub-workflow independently of the remaining partitions, it has the freedom to interpret and execute the sub-workflow using a form which best suits the resources available to it. We can describe this by taking two commonly occurring patterns as examples: pattern B as a *Map-Reduce*; and pattern E as a *Pipeline*.

The enactment gateway which receives partition B can decide the best mapping (multiplicity of the concurrent processes in the Map-Reduce) based on the number of parallel processors available to it. Similarly, the enactment gateway which receives partition E can decide whether the three stages of the pipeline should be mapped on the same processor using multi-threading (in cases of large data transfers between stages), or to map each stage to a separate processor (in cases where the data transfers are low but the stages are computation intensive).

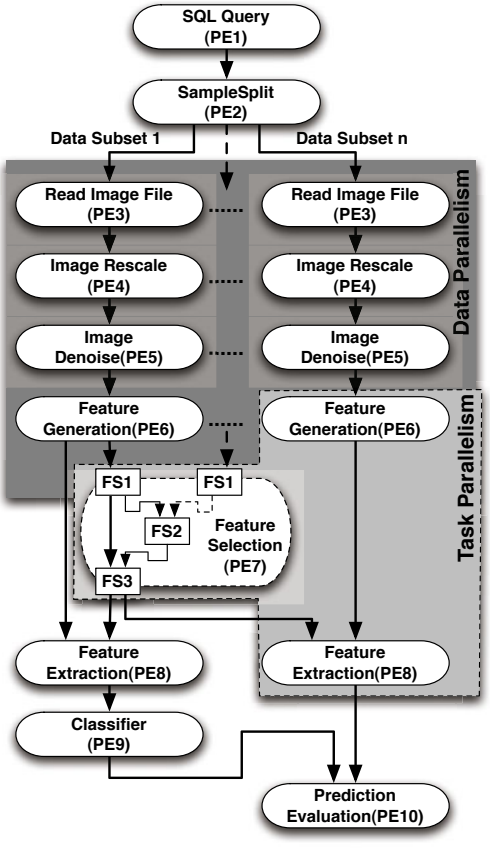

Fig. 3. Workflow composition for EURExpress-II experiment

Concrete example. To illustrate this with a concrete example, consider the workflow composition shown in figure 3. This is representative of the workflow used in our EURExpress-II experiment (discussed in section 3). The workflow uses several components, and we saw in figure 1 possible concrete semantics for two such components.

Let us focus on the section identified as data parallel. In this section, the image processing part is composed of PE3, PE4, PE5, and PE6. In the actual DISPEL implementation, we define this as a composition (not shown in the figure). As part of the optimisation carried out by the enactment engine, we take advantage of this data parallelism by replicating the image processing composition. This is done when PE2 and PE7 are expanded. Based on the execution environment available, the enactment engine parametrises calls to function PE2 and PE7 (DISPEL functions returned by the registry) by supplying the number of times the image processing composition should be replicated. When further expansions have reached the computational activities, resources are assigned.

If we consider the replicated composition involving PE3, PE4, PE5, and PE6, we can see that it is in fact a pipeline. Hence, the enactment engine exploits the task parallelism by assigning each of these components to different resources if the pipeline components expand to computational activities. On the other hand, if these components are composite, the enactment engine might send them to different enactment gateways that are part of the federation. Thus, each of the enactment gateways determines how each of the computational activities in the pipeline stages are assigned to the available resources.

In short, federated execution helps us achieve the full potential of a truly distributed computational platform. In the above example, the workflow partitioning is limited to top-down expansion, where each enactment gateway takes advantage of the available environment by parametrising the functional representations received from the semantic registry. We are currently investigating arbitrary partitioning and transformation (e.g., component sequence re-ordering) of the workflow using flow and cost-model analysis.

3 Case-Study: Data Mining and Integration

To demonstrate the ideas described in the previous sections, we shall now discuss a case-study from the life sciences. We have carried out a data mining and integration operation using the EURExpress-II [11] project data set. For this experiment, we have also implemented the required concrete semantic objects using the OGSA-DAI[3] framework.

The EURExpress-II project[2] builds an atlas database for a developing mouse embryo. It uses automated processes for *in situ* hybridisation on all of the genes from one stage of the developing embryo. The outcome of this automated process is a collection of images corresponding to the various sections of the embryo, with unique stains marking specific gene expressions. These images are then annotated by human curators, which involves tagging the images with anatomical terms from the ontology of a developing mouse embryo according to the gene expressions. The project has, so far, tagged 80% of the images manually, which amounts to 4 terabytes of data in total. The aim of our data mining and integration case-study is to automate the tagging process for the remaining 20%. This involves classifying 85,824 images using a vocabulary of 1,500 anatomical terms.

3.1 Workflow Composition

The automated image annotation through data mining requires three main stages. In the *training* stage, we train the automated classifier using training data sets taken from the human annotated collection. In the *testing* stage, we test the classifier against data sets from the human annotated collection that was not used during the training. Finally, in the *deployment* stage, we use the automated classifier to annotate the unannotated images.

[2] EURExpress-II Project, `http://www.eurexpress.org/ee/`

The sub-tasks in each of the three stages are described as follows:

- Image integration: Before the data mining process begins, image data located at various sources (as files) are integrated with annotation data from various databases. This produces images with annotations.
- Image processing: Since the sizes of the images could vary, they are normalised to a standard size using image rescaling. The noise in these images is then reduced using median filtering.
- Feature generation: From the stains in the embryo sections, which represent gene expression patterns, we generate features using wavelet transformations. For the images in the EURExpress-II data set, the features generated can be quite large; for instance, for an image with 300x200 pixels, the number of features could reach up to 60,000.
- Feature selection and extraction: In order to build an automated classifier, we have to reduce the number of features that are taken into account. This is done by selecting the most significant feature subset. This subset is then applied during the feature extraction process, which transforms the original features into feature vectors that can be used by the classifier as input.
- Classifier construction: The feature vectors are finally used to classify the unannotated images.

All of the components, which correspond to these sub-tasks, are shown in figure 3. Due to the streaming parallelism inherent in the ADMIRE architecture, we observe two patterns of parallelism at the highest-level. For the image integration and image processing tasks, we exploit the data parallelism where image scaling and noise reduction are independent for each of the images. On the other hand, for the feature generation, selection, and extraction phases, we exploit task parallelism. For instance, feature generation in the testing stage and feature selection in the training stage are independent tasks. As discussed in section 2.5 for the embedded pipeline, we recursively exploit lower-level parallelism.

Table 1. The total execution time (in seconds), plotted using the number of computational nodes against the number of images

	Number of images						
Node	800	1600	3200	4800	6400	12800	19200
1	70.680	141.162	283.399	426.015	564.927	1139.069	1721.489
2	36.183	69.846	141.614	214.035	284.753	573.158	865.271
3	23.675	47.160	93.944	142.857	191.171	382.037	581.403
4	18.364	35.990	70.548	107.783	144.252	289.562	437.669
5	15.313	29.483	56.894	85.494	115.467	232.597	350.499
6	12.925	24.357	47.705	71.808	96.17	195.802	295.397
7	11.588	21.430	41.016	61.45	82.657	168.478	253.524
8	10.220	19.154	36.436	54.157	72.274	148.068	225.335

3.2 Experimental Results

The total workflow execution times (in seconds) and the corresponding speedups of the automated data mining experiments are shown in tables 1 and 2. Their graphical representations are shown in figures 4 and 5 respectively.

These results are based on an experimental setup using nine dedicated machines that are connected over a local 100mbps network. A distributed computational framework (enactment platform) was created over this system using the OGSA-DAI framework 3.1, Java 1.6, Globus Toolkit 4.2 (web services core), and Jakarta Tomcat 5.5. Of the nine machines, a MacBook Pro with Intel 2GHz Core 2 Duo and 2GB RAM was configured as the enactment gateway, where the workflow composition was submitted. The remaining eight machines were workstations with Intel 2.4 GHz Core Duo with 2GB RAM each.

Table 2. The speedup, plotted using the number of computational nodes against the number of images

	Number of images						
Node	800	1600	3200	4800	6400	12800	19200
1	1.000	1.000	1.0000	1.0000	1.0000	1.0000	1.0000
2	1.953	2.021	2.0012	1.9904	1.9839	1.9874	1.9895
3	2.985	2.993	2.9763	2.9706	2.9678	2.9637	2.9623
4	3.849	3.922	3.9639	3.9490	3.9416	3.9305	3.9268
5	4.616	4.788	4.9586	4.9276	4.9123	4.8894	4.8818
6	5.469	5.796	5.9662	5.9102	5.8827	5.8418	5.8283
7	6.099	6.587	6.9924	6.9005	6.8555	6.7891	6.7672
8	6.916	7.370	8.0434	7.9022	7.8335	7.7326	7.6995

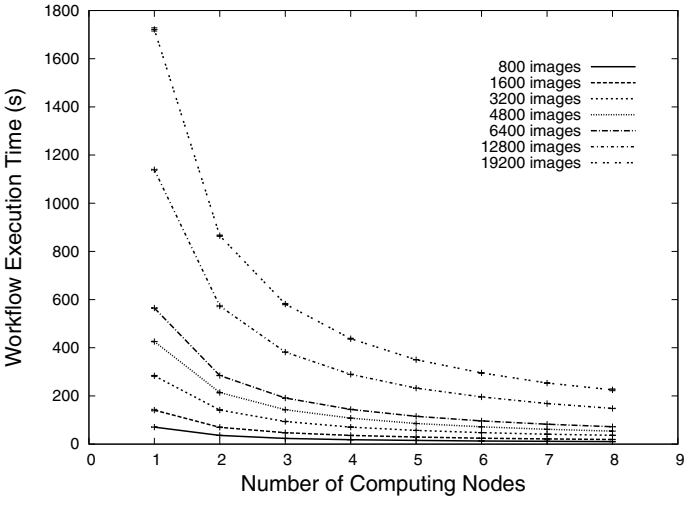

Fig. 4. Workflow execution times after exploiting streaming parallelism

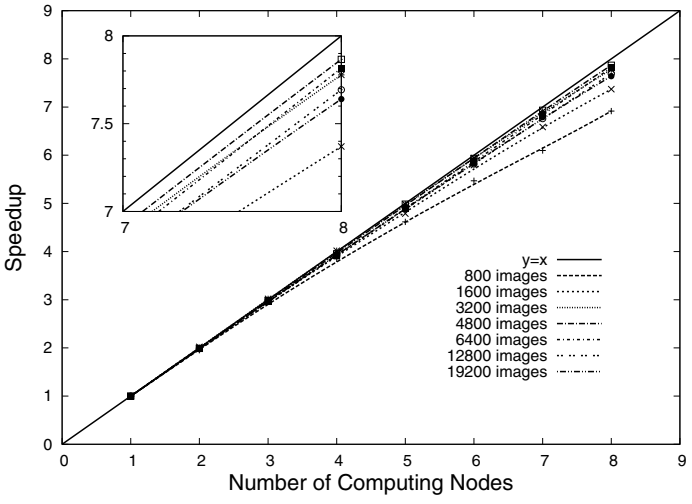

Fig. 5. Speedup when using multiple computational nodes with streaming parallelism

The graphs in figures 4 and 5 show that the streaming parallelism inherent in the ADMIRE architecture can be exploited fruitfully to yield close-to-linear speedup; except for the case of 3200 images run on eight computational nodes, where we see super-linear speedup. This anomaly is possibly caused by caching in the nodes from previous experiments.

4 Related Work

Patterns, as a notion for the abstraction of complexity, permeate various fields of investigation involving structure and coordination. The idea of using patterns as a multi-purpose tool for software development is discussed in detail in [16]. Riehle and Züllighoven defined software patterns, their forms and context, and emphasised the need for cataloguing and referencing these patterns during different phases of software development. One such area of research, which is currently very active, is the usage of patterns to capture and abstract parallelism in both distributed and multi-core systems (see [14] for a survey of this field). With regard to workflow management systems, abstraction of parallelism is an important area of research since service-oriented architectures are primarily concurrent. Some of the most prominent systems in this category are Taverna [15], Kepler [2], and Pegasus [7].

Investigations into the structure of workflows, and coordination between primitive components, have previously been done using various formal tools such as Petri Nets and UML. For instance, Ellis and Nutt [10] investigated the modelling and enactment of workflows using Petri Nets; whereas, Dumas and Hofstede [8] used UML activity diagrams to express workflows. Evolution of workflows, and mechanisms to capture and specify those changes, was reported in [9] by Ellis *et al.* and in [12] by Joeris and Herzog. Following the survey [17] due to Rinderle *et*

al. on the correctness criteria of evolving workflow systems, Sun and Jiang [18] analysed dynamic changes to workflows using Petri Nets. The idea of using patterns as a tool to encapsulate these analyses began with [19], when Aalst *et al.* further advanced the level-of-abstraction by using workflow patterns to capture recurring structures of coordination between primitive components.

In this paper, we have addressed an important aspect of workflow patterns which is lacking in these investigations—the high-order abstraction of patterns and the dynamic runtime assignment of concrete semantics. By introducing semantic registries, it is now possible to catalogue well-defined patterns based on their abstract semantics; while facilitating algorithm experts and computing engineers in providing various implementations of the concrete semantics. This freedom automatically facilitates parallelisation and resource-specific optimisations, since the expansion of patterns, inside an abstract workflow composition, takes place inside the enactment gateway, concealed from the domain experts.

5 Conclusions

We have addressed two research questions concerning workflows: 1) how do we abstract and catalogue recurring workflow patterns?; and 2) how do we facilitate optimisation of the mapping from workflow patterns to actual resources at runtime? To adapt to the practical reality of service provider multiplicity and infrastructure evolution, our approach relies on using well-defined patterns that can be mapped to concrete computational resources depending on the execution environment available. The novelty of our approach lies in our ability to separate abstract semantics (the functional aspects of a pattern) from the range of concrete semantics (the actual implementation using various middleware services); and the federated execution model, which allows execution platforms to share the workload transparently using a federation of distributed resources. Using a data mining case-study drawn from the life sciences, we have demonstrated experimentally the feasibility of this architecture.

Acknowledgments. The work presented in this paper is funded by the European Commission under Framework 7 ICT 215024.

References

1. Agostini, A., Michelis, G.D.: Improving Flexibility of Workflow Management Systems. In: Business Process Management, Models, Techniques, and Empirical Studies, London, UK, pp. 218–234. Springer, Heidelberg (2000)
2. Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., Mock, S.: Kepler: an extensible system for design and execution of scientific workflows. In: Proc. of 16th Intl. Conf. on Scientific and Statistical Database Management, pp. 423–424 (June 2004)
3. Antonioletti, M., Atkinson, M.P., Baxter, R.M., Borley, A., Chue Hong, N.P., Collins, B., Hardman, N., Hume, A.C., Knox, A., Jackson, M., Krause, A., Laws, S., Magowan, J., Paton, N.W., Pearson, D., Sugden, T., Watson, P., Westhead, M.: The design and implementation of grid database services in ogsa-dai. Concurrency - Practice and Experience 17(2-4), 357–376 (2005)

4. Atkinson, M.P., van Hemert, J.I., Han, L., Hume, A., Liew, C.S.: A distributed architecture for data mining and integration. In: Proc. of the Second Intl. Workshop on Data-Aware Distributed Computing, pp. 11–20. ACM, New York (2009), doi:10.1145/1552280.1552282
5. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Conceptual Modelling of WorkFlows. In: Proc. of the 14th Intl. Conf. on Object-Oriented and Entity-Relationship Modelling, London, UK, pp. 341–354. Springer, Heidelberg (1995), doi:10.1007/BFb0020545
6. Cole, M.: Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming. Parallel Comput. 30(3), 389–406 (2004)
7. Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A.C., Jacob, J.C., Katz, D.S.: Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Scientific Programming 13, 219–237 (2005)
8. Dumas, M., ter Hofstede, A.H.M.: UML Activity Diagrams as a Workflow Specification Language. In: Proc. of the 4th Intl. Conf. on The Unified Modeling Language, Modeling Languages, Concepts, and Tools, London, UK, pp. 76–90. Springer, Heidelberg (2001)
9. Ellis, C., Keddara, K., Rozenberg, G.: Dynamic change within workflow systems. In: Proc. of Conf. on Organizational Computing Systems, pp. 10–21. ACM, New York (1995)
10. Ellis, C.A., Nutt, G.J.: Modeling and enactment of workflow systems. In: Proc. of the 14th Intl. Conf. on Application and Theory of Petri Nets, London, UK, pp. 1–16. Springer, Heidelberg (1993)
11. Han, L., van Hemert, J.I., Baldock, R., Atkinson, M.: Automating gene expression annotation for mouse embryo. In: Huang, R., Yang, Q., Pei, J., Gama, J., Meng, X., Li, X. (eds.) Advanced Data Mining and Applications. LNCS, vol. 5678, pp. 469–478. Springer, Heidelberg (2009)
12. Joeris, G., Herzog, O.: Managing Evolving Workflow Specifications. In: Proc. of the 3rd IFCIS Intl. Conf. on Cooperative Information Systems, Washington, DC, USA, pp. 310–321. IEEE Computer Society, Los Alamitos (1998)
13. Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.J.: On Structured Workflow Modelling. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 431–445. Springer, Heidelberg (2000)
14. Mattson, T.G., Sanders, B.A., Massingill, B.L.: Patterns for Parallel Programming. Addison-Wesley, Reading (2005)
15. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A., Li, P.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20, 3045–3054 (2004)
16. Riehle, D., Züllighoven, H.: Understanding and using patterns in software development. Theor. Pract. Object Syst. 2(1), 3–13 (1996)
17. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems: a survey. Data Knowl. Eng. 50(1), 9–34 (2004)
18. Sun, P., Jiang, C.: Analysis of workflow dynamic changes based on Petri net. Inf. Softw. Technol. 51(2), 284–292 (2009)
19. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow Patterns. Distributed and Parallel Databases 14(1), 5–51 (2003), doi:10.1023/A:1022883727209