Actor-Driven Workflow Execution in Distributed Environments

Frank Berretz[1], Sascha Skorupa[1], Volker Sander[1], Adam Belloum[2], and Marian Bubak[2,3]

[1] FH Aachen University of Applied Sciences, Juelich, Germany
[2] University of Amsterdam, Amsterdam, Netherlands
[3] AGH University of Science and Technology, Krakow, Poland

Abstract. Currently, most workflow management systems (WfMS) in Grid environments provide push-oriented task distribution strategies, where tasks are directly bound to suitable resources. In those scenarios the dedicated resources execute the submitted tasks according to the request of a WfMS or sometimes by support of a Meta-Scheduling service. This approach has specific problems, especially because of various conditions and constrains that have to be taken into account like local policies or the sites' autonomy. To deal with such issues, this paper takes a closer look to the task distribution strategies. The established Grid WfMSs essentially support control-flow and data perspectives. However, they neglect the resource perspective. This paper exposes the advantages to deal with this perspective and demonstrates its feasability by a prototype implementation that integrates the missing resource patterns into UNICORE.

Keywords: Grid WfMS, Resource Pattern, UNICORE, jBPM, Human Tasks.

1 Introduction

Current e-Science infrastructures provide support for complex scientific processes that consist of orchestrated resources such as pure computational devices, data repositories, scientific instruments or applications. Grid environments are the common technical approach used to build these e-Science infrastructures on a middleware platform by a proper service layer. In order to support the orchestration of scientific tasks, many Grid middleware platforms offer a WfMS either as an integrative part or as an enactment service build on top of the middleware. So far, all popular Grid WfMSs use an approach that follows push patterns [4]. Here, a software agent, e.g. the workflow engine, actively exercises control about the progress of a workflow by pushing the individual tasks to selected resources according to the dependencies, provided by the workflow description. Consequently, in either case it is the WfMS that takes the initiative and causes the distribution process of newly created workflow tasks to occur. The alternative solution would be a pull-based actor-driven approach, where the commitment to undertake a specific task is initiated by the resource itself rather than the system. But the implementation of the associated pull patterns are not considered in today's Grid WfMSs that all prefer a system-initiated resource allocation [4]. In fact, all Grid WfMSs and their corresponding description languages focus on control-flow and data issues when mapping scientific processes onto workflows. In such data-centric systems resources are just regarded as dedicated machines that execute software as instructed. Within this paper the term "actor" is used to describe the entity that is actually responsible for the execution of a task. This means, e.g. an actor can be a computing Grid resource that executes a job according to a specific task. The proposal of this paper is to take a closer look on the resource perspective and especially the work distribution strategies. The participating resources who actually are the actors of a workflow should not be treated as passive automata with little influence to the assigned tasks and the way work is distributed. The modeling of the acting resources should be considered in a proper way and the connection between the resources and WfMSs is not just unidirectional. In the following paragraphs the focus is on the pros and cons of push- and pull-based work distribution approaches. Furthermore, this paper presents a possible prototype implementation of the pull-based approach on the example of the well known jBPM WfMS [11] and the UNICORE Grid middleware [7].

2 Related Works

This paper presents a novel concept for WfMSs in Grid environments. All known Grid WfMS approaches follow the push pattern. Desktop Grids, such as BOINC, EDGeS or pilot jobs in the DIRAC Pilot framework [10][9] can be viewed as intermediate approaches that emulate a pull concept. The presented task repository can be realized by using mechanism such as UNICORE XML-spaces [6], where computational resources request for jobs from a shared job queue implemented as tuple space. However, the space is targeted towards high-throughput computing and that's why it is a thin application without any logical layer that is required for our application domain. Clearly, the work among the emerging WS-HumanTask [12] standard is related to the presented concept. There, a model is described that enables the integration of human beings in service-oriented applications like BPEL processes. In the proposed architecture the WfMS is separated from a task processor, which is a standalone component, exposed as a service to manage tasks' lifecycles. But in contrast to the task processor of WS-HumanTask the task repository should manage tasks for any kind of actors.

3 Task Distribution to Resources

As indicated in the introduction the currently used Grid WfMSs primarily focus on the control-flow and data perspectives, while neglecting the resource perspective of the well-known workflow patterns. From a resource's perspective the manner in which tasks are advertised and ultimately bound for execution is of particular importance. In Figure 1 a part of the state transition diagram for

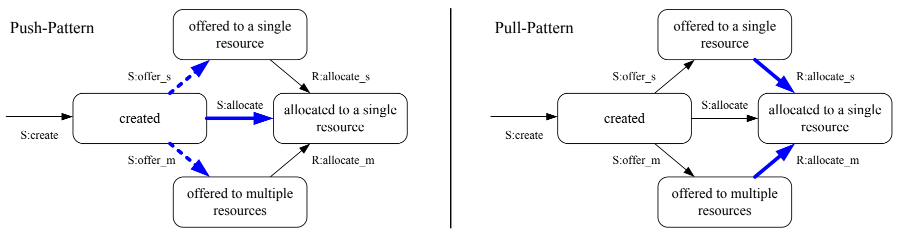
Push-Pattern
offered to a single resource
S:offer_s
R:allocate_s
created
S:allocate
allocated to a single resource
S:create
S:offer_m
R:allocate_m
offered to multiple resources
Pull-Pattern
offered to a single resource
S:offer_s
R:allocate_s
created
S:allocate
allocated to a single resource
S:create
S:offer_m
R:allocate_m
offered to multiple resources

Fig. 1. Patterns of the resource perspective to bind resources to tasks [1]. Possible task states are denoted by boxes and transitions by edges. Blue edges indicate the applicable transitions supported by the particular pattern. Dash doted edges indicate that this state transitions only make sense if there is a transition of a pull pattern.

tasks is illustrated according to Aalst et al. [1]. In this paper, we focus on the allocation process, which is performed after a task is created and before it is executed by a suitable resource.

After a workflow task comes into existence, it is set into the `created` state, indicating that it is capable of being executed. Here, the task has not been bound to a specific resource for execution. Obviously, there are multiple possible paths through these states for an individual task. The edges within this diagram are prefixed with either an `S` or an `R` indicating that the transition is initiated by the WfMS or resource respectively. Essentially, there is a distinction between push- and pull-patterns, identified by the initiator of the various transitions. The state transitions that belong to the push-patterns are typically initiated by the WfMS for `created` tasks. So, these push-patterns concentrate on the subject of making resources aware of available tasks to execute. This can be realized in three different ways leading to distinct subsequent states. In today's Grid WfMSs, a task is usually allocated to a suitable resource explicitly denoted by the `s:allocate` operation. Hence, the resources are always allocated by the system. Indeed, most Grid WfMSs use schedulers and information systems to identify the most appropriate resource for the task allocation. Nevertheless, the real binding of a task to a resource is initiated by the system. Consequently, the alternative courses of action indicated by the states `offered to a single resource` and `offered to multiple resources` are not considered in common Grid WfMSs. This means in turn that the resource-initiated operations of the pull-patterns like `R:allocate_s` and `R:allocate_m` are not implemented in current Grid WfMSs. This is exactly the gap that the concepts presented in this paper aimed to bridge. The feasibility of our concepts is discussed in section 3.

The integration of such pull-pattern might solve various existing problems with regard to the interactions between the WfMSs and the resources. For instance, pushing each job from the WfMS to the resources requires efficient working Meta-Schedulers [3] that have to be to be able to coordinate and allocate resources across multiple administrative domains. These Meta-Schedulers have to consider various conditions and constraints, like local policies and the autonomy of each site, security issues like authentication and authorization as well as the heterogeneity of the different sites and their local scheduling systems.

A pull-based approach could remove the need for a complex scheduling process to identify the right resource explicitly and enables the resource providers to enforce their local policies. This approach to distribute tasks could effectively speed throughput by eliminating the notion of complex allocation by scheduling services. With regard to the heterogeneity of the resources the pull-based allocation strategies for tasks are also benificial. The incorporation of more specific resources like the integration of human beings, telescopes or medical devices is, however, a rather difficult task that often lacks of related standards such as JSDL for job submission to computational resources. Standards like the emerging WS-HumanTask [12] that might address this issue are neither sufficiently supported by Grids nor cover all kinds of specific resources. We believe the main reasons for the troublesome incorporation of specific resources are the commonly used push-based allocation mechanism, which postulates that a WfMS or any other job distributer require a set of well-known interfaces to interact with the resources. While OGSA provides a general framework for this, more details such as provided by JSDL are needed for specific resources. Because this assumption cannot be fulfilled with reasonable expenditure and in absence of related standards, it is currently hard to join the push-oriented model to the ambition of integrating special resources. The proposed pull-based task distribution strategy makes this goal relatively easy to realize because it opens the way to integrate resources that do not have a well-defined interface. Therefore, it is necessary to offer tasks to execute to resources, e.g. by informing multiple suitable resources of the existence of a specific task. In this case the WfMS does not attempt, which resource should undertake the task. So the resources, to which the task is offered, are free to choose whether they are interested in undertaking the task or not. Generally, this procedure results in the task being placed on a specific task list of the individual resources for later execution. In the proposed pull concept actors are the driving force of the execution of a workflow. Indeed, this approach makes a centralized resource control e.g. by scheduling systems impractical. But, at the same time it opens new perspectives with respect to community approaches, where members delegate their resources to execute tasks as actors for the benefit of a community.

4 UNICORE Middleware Integration

The above discussed concept of using pull patterns to bind resources to tasks has been ported to existing Grid technologies in the scope of the ongoing project HiX4AGWS [8]. In particular, the UNICORE Grid middleware has been extended by establishing a corresponding task repository to allow an actor-driven execution of tasks. The resulting UNICORE architecture is illustrated in figure 2. A starting point for integrating the pull-based approach has been the jBPM WfMS [11]. The jBPM engine that is a suitable candidate for the realization of the pull-based approach especially because of its expandability supports the usual workflow control-flow and data patterns [1].

In the first step the jBPM engine has been integrated to UNICORE framework similarly to the actually used Chemomentum engine [7]. The fundamental

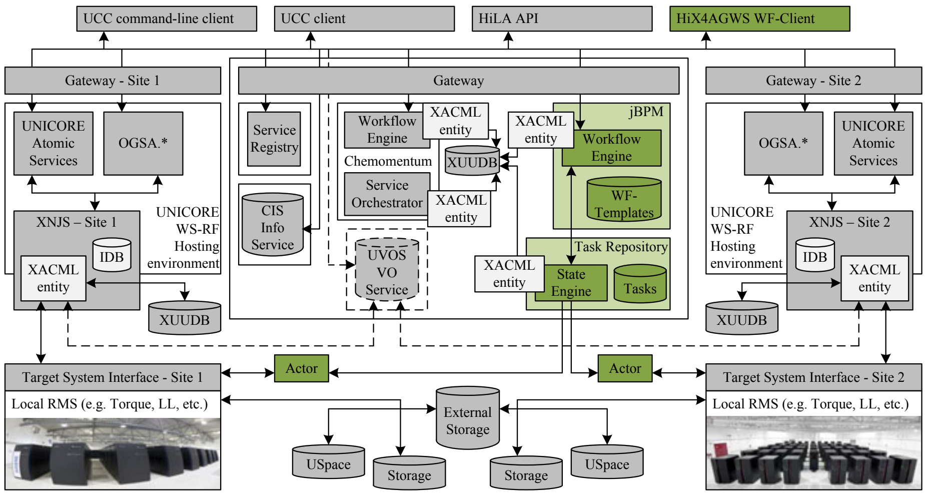

Fig. 2. Architecture of the prototype and its integration into UNICORE. The grey boxes represent the UNICORE components that have not been changed. Green highlighted boxes indicate the components implemented within the HiX4AGWS project.

difference between these two WfMSs is the concept of abstract and concrete workflows. Thereby an abstract workflow can be regarded as a generalization of a business case whose instances are referred to as concrete workflows. In jBPM an abstract workflow has to be deployed before it can be instantiated as often as desired to finally execute several concrete workflows. The Chemomentum system lacks of this concept to distinct between abstract and concrete workflows. Here, a workflow modeled by a user is executed just once immediately after the submission. Therefore, the jBPM engine offers some advantages, particularly with respect to the reusability of workflows and hence to a collaborative working.

For the purpose of the UNICORE integration of jBPM, the essential interfaces of the WfMS, e.g. to deploy or delete abstract workflows, has been abstracted through web services. Endpoint references (EPRs) are used to explicitly identify workflows. For instance, once a abstract workflow is deployed, in form of a jPDL schema conform XML document [11], the associated web service operation returns the relevant EPR to the invoking entity. Afterwards, this EPR can then be used to create concrete workflows or to delete the deployment and all associated instances. If, e.g., a wrong EPR is used to instantiate a workflow or any other internal or external failures occur, all web service operations are able to notify the invoking entity with the help of a corresponding exception handling. After successfully integrating jBPM into UNICORE by the explained web services, the next step is to focus on the actually intended goals: Autonomous resources should be able to apply for jobs in a pull-based way. Therefore, a new kind of task has been introduced to jBPM. During the processing of this new task, jBPM contacts an external repository, where the associated task description is published. That means the task is not executed by the WfMS on the local UNICORE site, but is available for interested and appropriate resources in a repository. After, a task that is published like this is executed by an appropriate resource, jBPM is notified through a callback mechanism and the corresponding workflow is resumed. The result is that the extension of the jBPM engine allows a pull-based task distribution and execution by corresponding resource clients.

4.1 Task Repository

Like mentioned above the implementation of a task repository is necessary to realize a pull-based task distribution. This repository has to be integrated into UNICORE like the jBPM engine by abstracting it through UNICORE web services. On the one hand the repository serves as storage for the tasks and jobs that need to be executed and that are received by the WfMS. On the other hand the repository is a kind of intermediary between the published tasks and the respective actors. Hence, the task repository is designed to provide two independent interfaces. One interface enables the jBPM WfMS to publish jobs that can be executed by appropriate actors. The second interface allows resources to deal with tasks, e.g. to query for tasks, waiting for them and to work on these tasks.

Once the WfMS publishes a task, a wrapper object is created and stored in the repository. This object consists of the task description, which can be a JSDL job and resource description. The resource description is used to define which role a resource must possess to claim and execute the task. This process corresponds to a role-based access control, which also allows working across various VOs. If a resource provider wants to apply for specific kinds of tasks, he just has to get the associated role. Besides this information, which is particularly necessary for the communication to the actors, the wrapper object also consists of callback information. This information allows notifying the WfMS and respectively the corresponding workflow instance about a successful execution of a task to trigger the workflow's progress. The current status of a task is additional an essential information stored in the wrapper object. For this purpose a state engine has been implemented to manage the different states of a published task and the transitions between them. The associated state diagram for the tasks is shown in figure 3. The presented actions and their resulting state transitions are abstracted by web service operations that can be invoked by the actors. Hence, the actors are able to communicate with the task repository by web service operations like claim(), start(), finish() or cancel(). The unique identification of a task is again directed through EPRs. An integrated exception framework notifies the actors about invalid state transitions or faulty accesses to EPRs.

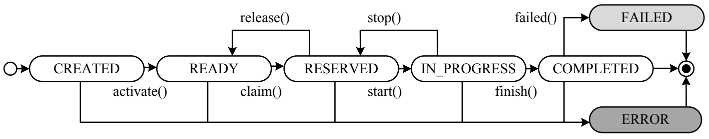

Fig. 3. The task repository's state engine including all states represented by oval boxes, transitions and its triggering actions represented by the arrows and its labels

4.2 Evaluation - Human Tasks in Unicore

UNICORE uses the Chemomentum engine to execute workflows. In this WfMS the resource allocation is done by the service orchestrator, who submits JSDL jobs to appropriate resources, supervises their execution and informs the workflow enginge of job completion. So, the service orchestrator can be identified as the resource broker or scheduler working according to the push pattern, because he determines which resource is used for a specific job. The process of job submission only works, if the service orchestrator knows the interfaces of the resources, which makes the integration of more specific resources like humans hard to realize. Because sometimes a resource may even be a person, one goal of our project was to integrate human tasks into UNICORE workflows. This is a useful feature, if workflows require a human step, such as verification by a scientist that the workflow is proceeding correctly. To integrate human resources into a workflow execution, our jBPM integration prefer a hybrid usage of pull and push patterns for task distribution. This approach also serves as an evaluation for the implemented pull patterns. In the hybrid approach, a task to be executed by a human can explicitly be denoted as "pull"-task in the workflow description. The jBPM engine sends such a task to the task repository where a human can pick up it for execution according to the pull pattern. Each task that is not denoted as pull-task is treated like a common Grid job that can be sent to the service orchestrator by using its web service interface. This solution enables the integration of human resources into UNICORE workflows, whereby a first project goal can be complied. Furthermore, the pull-based approach can be evaluated by executing a workflow that contains a human step on our system. At this time, it is not possible to make performance analysis because the task repository and the corresponding pull concept lacks a comprehensive security concept. For this reason, it is inequitable to compare workflow execution of the jBPM and the Chemomentum engine. Such analysis are planned for the future.

5 Conclusions and Future Works

This paper has shown that current Grid WfMSs neglect the resource perspective of the workflow patterns resulting in several problems that are mentioned in the previous sections. Developments as mentioned in section 4 try to integrate the resources' point of view into Grids. But these approaches continue to place importance on well known standards. The proposed actor-driven approach binds resources to tasks through an intermediate task repository presented in this paper. The prototype prepares a way to cleanly integrate pull patterns into UNICORE. Important components are the task repository and the actors. Therefore, the Grid resources as well as human resources are connected to the task repository. Thus, e.g. human beings are able to claim and execute tasks like the qualitative evaluation of interim results resulting in a clean integration in scientific workflows. The next steps consist of the integration of additional resources by proper actors into the Grid. Particularly, the expansion of the actor-based pull concept by using Cloud computing technologies e.g. to incorporate resources from community clouds may increase the elasticity of the system with respect to the amount of users as well as the amount of available resources.

Acknowledgements. This research is conducted in the context of the History tracing XML for an Actor-driven Grid-enabled Workflow System project, supported by the Federal Ministry of Education and Research and the Federation of Industrial Research Associations in Germany.

References

1. Van der Aalst, W.M.P., et al.: Workflow Resource Patterns. BETA Working Paper Series, WP 127, Eindhoven University of Technology, Eindhoven (2004)
2. Van der Aalst, W.M.P., et al.: Workflow Patterns. Distributed Parallel Databases 14(1), 5–51 (2003)
3. Wldrich, O., Wieder, P., Ziegler, W.: A Meta-Scheduling Service for Co-allocating Arbitrary Types of Resources. CoreGRID Technical Report, Nr TR-0010 (2005)
4. Buyya, R., Yu, J.: A taxonomy of scientific workflow systems for grid computing. Journal of Grid Computing 3(3-4), 171–200 (2005)
5. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organization. International Journal of Supercomputer Applications 15(3), 200–222 (2001)
6. Schuller, B., Schumacher, M.: Space-Based Approach to High-Throughput Computations in UNICORE 6 Grids. In: Euro-Par 2008 Workshops, pp. 75–83 (2009)
7. Schuller, B., et al.: Chemomentum - UNICORE 6 based infrastructure for complex applications in science and technology. In: Bougé, L., Forsell, M., Träff, J.L., Streit, A., Ziegler, W., Alexander, M., Childs, S. (eds.) Euro-Par Workshops 2007. LNCS, vol. 4854, pp. 82–93. Springer, Heidelberg (2008)
8. Berretz, F., Skorupa, S., Sander, V., Belloum, A.: Towards an Actor-Driven WfMS for Grids. In: Proceedings of CTS 2010, Chicago, pp. 611–617 (2010)
9. Casajus, A., et al.: DIRAC Pilot Framework and the DIRAC Workload Management System. Journal of Physics: Conference Series 219(6) (2010)
10. Urbah, E., et al.: EDGeS: Bridging EGEE to BOINC and XtremWeb. Journal of Grid Computing 7(3), 335–354 (2009)
11. jBPM Developers Guide (January 2010), `http://docs.jboss.com/jbpm/v4/devguide`
12. WS-HumanTask V1.0. (June 2007), `http://xml.coverpages.org/WS-HumanTask-V1-0706.pdf`