A Scheduler for a Multi-paradigm Grid Environment

Nadia Ranaldo, Giancarlo Tretola, and Eugenio Zimeo
{ranaldo, tretola, zimeo}@unisannio.it
Department of Engineering, University of Sannio
82100 Benevento - Italy

Natalia Currle-Linde, Wasseim AL Zouabi, Oliver Mangold, and Michael Resch
First.Last@hlrs.de
High Performance Computing Center Stuttgart (HLRS), University of Stuttgart

Denis Caromel, Alexandre di Costanzo, Christian Delbé,
Johann Fradj, Mario Leyton, and Jean-Luc Scheefer
First.Last@sophia.inria.fr
INRIA Sophia-Antipolis - Université de Nice - CNRS/I3S
2004, Route des Lucioles, BP 93 FR-06902 Sophia Antipolis, France

CoreGRID Technical Report
Number TR-0155
July 30, 2008

Institute on Programming Model

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

A Scheduler for a Multi-paradigm Grid Environment

Nadia Ranaldo, Giancarlo Tretola, and Eugenio Zimeo
{ranaldo, tretola, zimeo}@unisannio.it
Department of Engineering, University of Sannio
82100 Benevento - Italy

Natalia Currle-Linde, Wasseim AL Zouabi, Oliver Mangold, and Michael Resch
First.Last@hlrs.de
High Performance Computing Center Stuttgart (HLRS), University of Stuttgart

Denis Caromel, Alexandre di Costanzo, Christian Delbé,
Johann Fradj, Mario Leyton, and Jean-Luc Scheefer
First.Last@sophia.inria.fr
INRIA Sophia-Antipolis - Université de Nice - CNRS/I3S
2004, Route des Lucioles, BP 93 FR-06902 Sophia Antipolis, France

CoreGRID TR-0155
July 30, 2008

Abstract

Typical Grid environments are characterized by a two-level scheduling: super and local scheduling. While the former regards the selection of proper resources to execute high level tasks, the latter regards the scheduling of tasks to processors for optimizing performance and hardware exploitation. This paper proposes a super scheduler that provides services to domain specific high-level programming environments, while taking advantage of middleware libraries to handle the complexity of Grid computing. To prove the flexibility of the scheduler, three different programming environments with slightly different programming abstractions have been integrated with it: a problem solving environment (SEGL), a skeleton framework (Calcium) and a workflow enactment engine (SAWE). The integration above shows how it is possible to facilitate the adoption of the ProActive middleware for the development of complex Grid applications.

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

1 Introduction

Thanks to the increasing amount of resources available across the Internet and improvements of wide-area network performance, Grid computing has emerging in recent years as a viable computing paradigm to solve data and compute-intensive applications. Programming environments for building and executing applications must handle, among others, the complexity of the Grid's *deployment*, *dynamicity*, *distribution*, *fault tolerance*, *heterogeneity*, *high performance*, *interoperability*, and *scalability*. Given the complexity of these concerns, it is unreasonable to charge users for directly address them when programming their Grid applications. This has led to the development of Grid middleware delivering low-level functionalities handled in a domain independent way [19], related for example to security, low-level resource management, monitoring, data management, efficient communication, etc. On top of such low-level middleware high-level programming environments can be built. Each programming environment provides a domain specific high-level programming model, that simplifies and hides away the complexities of the Grid.

Following this direction, we aim at further factoring complex aspects in programming Grid systems into a common upper-middleware architecture. In particular we intend to factor common aspects of scheduling and resource management services, such as load balancing and fault tolerance, exploited by high-level programming environments, for the enactment of distributed and parallel applications in heterogeneous and dynamic systems.

This paper proposes a flexible super-scheduler that can be exploited by different domain specific high-level programming environments, while taking advantage of lower-middleware libraries. In order to prove its flexibility, we describe the integration of the scheduler with three different programming environments with slightly different programming abstractions: a skeleton framework (Calcium [11]), a workflow engine (SAWE [29, 28]) and a problem solving environment (SEGL [15]). The super-scheduler implementation is built on top of the ProActive middleware [10] and exploits the active object model that, by means of asynchronous method calls, allows to deal with communication latency in wide-area networks. It uses a configurable scheduling policy and the static GCM Deployment model for the dynamic acquisition and deployment of Grid computational resources through different creation and communication protocols. It delivers, moreover, services of monitoring, data dependencies among jobs and fault-tolerance.

The proposed integration shows how upper-level services can facilitate the adoption of Grid middleware, such as ProActive, to write complex Grid applications. It demonstrates, moreover, how the super-scheduler could be usefully exploited by other distributed programming models to fit heterogeneity and dynamicity, and in particular the GCM component model [14]. GCM is a component model for Grid computing based on Fractal [8]. GCM inherits hierarchy, introspection and the basic controllers from Fractal, and extends it using asynchronous method calls to deal with communication latencies. The reference implementation of the GCM is ProActive and it is also based on the active object model. Currently the GCM supports a static *GCM Deployment* model, which can acquire remote resources through deployment descriptors and deploy applications on the remote nodes. In a sense, the proposed super-scheduler could represents a dynamic extension of the *GCM Deployment* model, by allowing dynamic acquisition of Grid resources.

This paper is organized as follows. In section 2 we provide relevant background on ProActive, Calcium, SAWE and SEGL. Then, in section 3 we describe the scheduler archecture and main functionalities, and in section 4 we describe how the integration between the frameworks and the scheduler is done. Finally, we conclude and provide future perspectives in section 5.

Hence,

2 Grid Programming Environments

2.1 The ProActive Middleware

ProActive [10] is a Java library for concurrent, distributed, and mobile computing implemented on top of RMI HTTP, RMI/SSH, and Ibis as transport layers. ProActive features transparent remote active objects, asynchronous two-way communications with transparent futures, high-level synchronization mechanisms, and migration of active objects with pending calls. As ProActive is built on top of standard Java APIs, it does not require any modification of the standard Java execution environment, nor does it make use of a special compiler, preprocessor or modified Java Virtual Machine (JVM).

Distributed and concurrent applications built using ProActive are composed of a number of medium-grained entities called *active objects*. Each active object has its own thread of control and is granted the ability to decide in which order to serve the incoming method calls, which are automatically stored in a queue of pending requests. Method calls sent to active objects are asynchronous with transparent *future objects* and synchronization is handled by a mechanism known as *wait-by-necessity* [9].

2.2 Problem Solving Environment: SEGL

SEGL is a system for the automated execution of Grid experiments and for the efficient use of existing Grid resources. SEGL consists of two parts: Experiment Designer which is used for the design and debugging of experiments and ExperimentEngine for controlling the execution of the applications on the Grid resources (see Figure 2). The interaction between Experiment Engine and the Grid resources is done through a Grid Adaptor, interfacing with ProActive.

From the user's perspective, complex experiment scenarios are realized in Experiment Designer using Grid Concurrent Language (GriCoL) to represent the experiment. The technical mapping from this user perspective to the underlying infrastructure is carried out via the use of control flow, data flow and the repository layers. The control flow level is used for the description of the logical schema of the experiment. The main elements of this level are blocks: control blocks and solver blocks. A solver block is composed of nodes of data processing. It represents applications which use numerical methods of modeling. The control blocks are either nodes of data analysis or nodes for the synchronization of data computation processes. They evaluate results and then choose a path for further experiment development. The data flow level is used for the local description of interblock computation processes. The sublayer provides a common description of the database and a section for making additions to the database if necessary.

The runtime system of SEGL (ExpEngine) chooses the necessary computer resources, organizes and controls the sequence of execution according to the task flow and the condition of the experiment program. The user sets up initial conditions and parameters using GriCoL. Then SEGL monitors and steers the experiment, informing the user of the current status. SEGL employs a dynamic parametrization capability which requires an iterative, self-steering approach. The communication between the Experiment Designer and the runtime system supports the monitoring of execution of the experiment.

2.3 Skeleton Framework: Calcium

Algorithmic skeletons (*skeletons* for short) are a high level programming model for parallel and distributed computing, introduced by Cole in [13]. Skeletons take advantage of common programming patterns to hide the complexity of parallel and distributed applications. Starting from a basic set of patterns (skeletons), more complex patterns can be built by nesting the basic ones. All the parallelization and distribution aspects are implicitly defined by the composed skeletal structure.

$$\begin{aligned} \triangle ::= & farm(\triangle) \mid pipe(\triangle_1, \triangle_2) \mid seq(f_e) \mid \\ & if(f_b, \triangle_{true}, \triangle_{false}) \mid while(f_b, \triangle) \mid for(i, \triangle) \\ & map(f_d, \triangle, f_c) \mid fork(f_d, \triangle_1, ..., \triangle_n, f_c) \mid d\&c(f_d, f_b, \triangle, f_c) \end{aligned}$$

Where the task parallel skeletons are: $farm$ for task replication; $pipe$ for staged computation; seq for wrapping execution functions; if for conditional branching; and $while/for$ for iteration. The data parallel skeletons are: map for single instruction multiple data; $fork$ for multiple instruction multiple data; and $d\&c$ for divide and conquer.

As a skeleton framework we use Calcium [11], which is greatly inspired on Lithium [3, 4, 5, 17] and its successor Muskel [16]. Calcium is written in Java and is provided as a library. The Calcium framework is capable of evaluating the same skeleton program on different execution environments. Currently it supports a parallel environment using threads, and a distributed environment using ProActive's active objects. The distributed environment is suitable for short lived applications, running on a controlled environment (non-volatility, faultless, unscalable, etc..) such as a cluster environment.

For the Grid, instead of developing support for Grid concerns into the skeleton framework, Calcium can benefit by using a generic scheduling component, which can handle deployment, dynamicity, fault tolerance, security, scalability.

2.4 Workflow Engine: SAWE

Workflow management applied to Grid computing is emerging as an interesting and flexible paradigm to build and manage complex scientific applications [1]. With this approach an application is a temporal composition of independent functions provided by Grid resources, and whose data and control logic is described adopting a high-level language interpreted by a Workflow Enactment System (WES) [30].

SAWE (Semantic and Autonomic Workflow Engine) [29, 28] is a WES, compliant with the WfMC abstract reference model [22], whose main objective is to deliver advanced functionalities for the the creation, management and enactment of generic workflow processes, in business, engineering and scientific domains.

SAWE is a three-layered system (see Figure 4). The $Control\ layer$ is able to receive the process description in XPDL (Xml Process Definition Language) [12], creates and navigates the process graph and chooses the activities that could be executed according to the activation conditions. The $Binding\ layer$ is responsible for associating a concrete resource to a process activity that can be abstractly described using syntactic or semantic annotations. The $MatchMaker$ collaborates with the Binding layer and is responsible of resource discovery and selection of the most appropriate resource to assign to an activity for execution. Currently the MatchMaker is available for Web Services and uses syntactic and semantic annotations of services for performing matching functionalities. The $Interaction\ layer$ interacts with resources to enact activities using a generic $Resource\ Interface$ (RI) able to communicate with the upper layers independently from the communication middleware. Currently, specific RIs are delivered for RMI, Web Services, POJO, HTTP and ProActive. The interaction with the resources is demanded to a specific adapter for each technology.

SAWE uses asynchronous invocations and is able to anticipate the execution of sequential activities with a deferred blocking mechanism, if a proper middleware is used at interaction layer. To perform activity anticipation, the Interaction layer uses the asynchronous remote invocation provided by ProActive [27], returning placeholders to the Control layer that is so able to continue navigation in the process graph.

The dynamic binding is a crucial aspect for simplifying access to heterogeneous and distributed Grid resources and efficiently and transparently enacting Grid workflows. At this level, SAWE currently delivers a static or dynamic mapping of ProActive activities to resources without mechanism to manage load balancing and fault-tolerance at resource level. To overcome the limitation, a viable solution is the integration of SAWE with the ProActive-based Scheduler, in order to exploit its resource allocation management.

3 The Scheduler

The execution of parallel tasks on a pool of distributed resources, such as network of desktops or clusters, requires a main component for managing resources and handling task execution. This component is typically a *batch scheduler*. A batch scheduler provides an abstraction of resources to clients. Clients submit tasks and the scheduler is in charge of executing them on the available resources. Additionally, a scheduler allows several clients to share a same pool of resources, and also manages issues related with distributed environment such as faulted resources.

Several schedulers can be found in the literature. Some of the most known are LSF [31], PBS/OpenPBS [25], LoadLeveler [24], and Condor [23]. The survey [6] catalogs these major systems and reports their features. Most of them are implemented in C language and are very low-level in their implementation as well as their operating system integration. Thus, unable to handle portability and heterogeneity.

To target heterogeneous systems, Grid programming environments have developed *super-schedulers*. Super-schedulers are schedulers capable of federating several schedulers in a single one, such as Sun Grid Engine [21] and Condor-G [20]. With the complexity of Grids, super-scheduler rely on Grid middleware (*e.g.* Globus [18]) in order to handle services such as security, resource managing/monitoring, *etc.*

In this section, we present a super-scheduler component for Grid computing (*scheduler* for short), capable of federating several other schedulers. Clients can interact with the scheduler through different mechanism: command-line, API, GUI, and description files. In addition of a super-scheduler, we propose a *Resource Manager*, which is in charge of acquiring and managing resources using constrains, such as the matchmaking algorithm [26].

3.1 Architecture

The scheduler is the central entity with which clients interact using a remote Java API, or by submitting a `Job` Description. A `Job` describes the batch process to be executed. The description specifies the code, which can be in Java by extending the `Executable` interface or any native executable; required data files; and a script for validating resources.

Currently three kinds of jobs are supported: in **Task Flow** Jobs, clients describe the flow and dependencies of tasks to execute; in **Parameter Sweeping** a single task is executed in parallel with multiple data; and in **ProActive Applications** clients submit a regular ProActive distributed application.

The scheduler also supports customized allocation policies, and provides a FIFO policy by default. Basic non-functional concerns such as security and fault-tolerance are handled both at the ProActive middleware and scheduler levels.

Finally, the management, deployment, and selection of resources is handled by a second entity, named the *Resource Manager*. Figure 1 shows a global overview of the whole system.

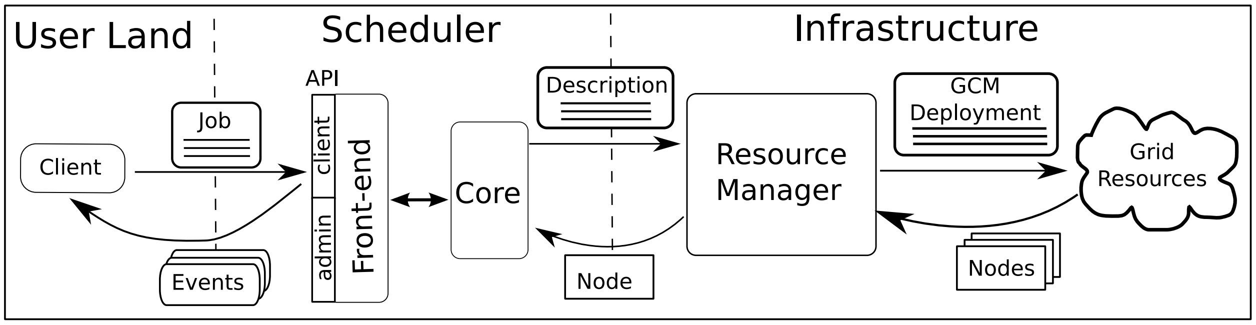

Figure 1: Scheduler Global Overview

3.2 Resource Manager

The Resource Manager (RM) is responsible for acquiring and managing resources. The RM is built on top of the GCM Deployment [2]. The GCM Deployment framework is the evolution of the previous ProActive Deployment [7] framework, which provides an abstraction of how resources can be acquired from the Grid. The deployment model allows the acquisition of resources using heterogeneous protocols, without changing the application source code. All information related with the deployment of the application is described in a deployment descriptor. Thus, all the following references are eliminated from the code: machine names, resource acquisition protocols, and communication protocols. The GCM Deployment model has been submitted for standardization to the European Telecommunications Standards Institute (ETSI), in order to provide standards for Grid deployment.

Hence, the static acquisition of resources is handled by the GCM Deployment model, and the dynamic management of these resources is done by the RM. A scheduler can ask resources from a RM, and the RM will deliver a resource through a node abstraction. Once the scheduler no longer requires a node, it is returned to RM for cleaning, pooling or releasing.

The scheduler can also request specific resources, that fulfill some requirements in order to execute a particular task. Requirements can be verified with a script attached to the task, the RM uses this script to test resources. A successful execution of this script on a given resource validates the node.

4 Integration Use Cases

4.1 Problem Solving Environment: SEGL

The application server consists of the experiment engine (ExpEngine), the container application (Task) and the controller component (MonitorSupervisor) (see Figure 2). The MonitorSupervisor controls the work of the runtime system and informs the Client about the current status of the jobs and the individual processes.

The ExperimentEngine (SEGL) consists of the TaskManager and the JobManager. The TaskManager organizes and controls the sequence of execution of the program blocks according to the task flow and calls the JobManager to start Grid jobs. The JobManager is the SEGL's interface to Grid middleware. It is informed when a job must be executed and receives the job's input and output data. After the job is finished the JobManager informs the TaskManager so that the workflow can be continued. In this work the Proactive job manager was implemented as an implementation of the generic Job Manager interface that manages SEGL jobs. It takes SEGL jobs, wraps them in a ProActive adapter object and sends them to a ProActive scheduler. It also keeps polling the scheduler for the jobs status. It manages input and output files by wrapping the file transfer tasks in the adapter and does the transfer in pulling style from the target

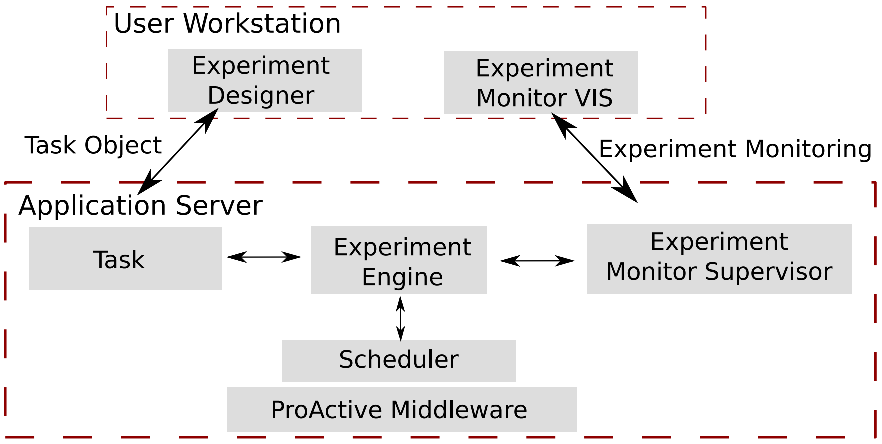

Figure 2: SEGL Integrated with ProActive based Scheduler

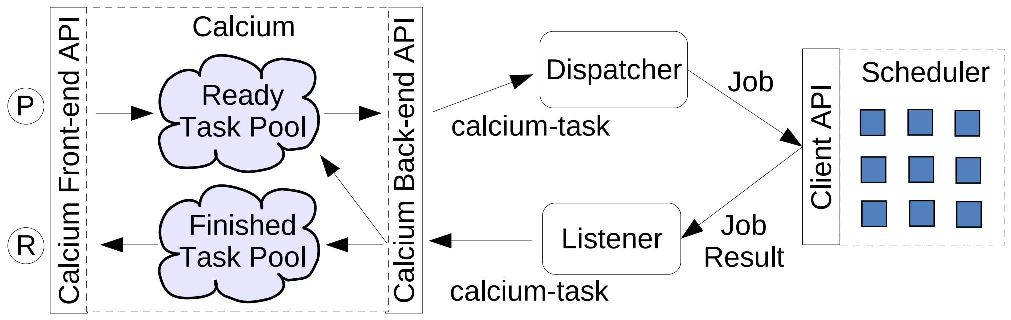

Figure 3: Calcium integration with Scheduler

machine. This allows file transfers to the job without having to know about the job's location during its submission to the scheduler. When a Grid job is finished it cleans up and then notifies its status to SEGL.

The Scheduler Component is responsible for the placement of SEGL tasks on the GRID, i.e. it is responsible for deployment. The ProActive Adapter has the SEGL job and all its information embedded in it. It is a wrapper object that runs on nodes assigned by the scheduler. It starts by preparing the environment and creating the folders, then it uses ProActive file transfer mechanisms provide input files and retrieve remote files. It then executes the SEGL task on the node. Finally, it pushes the resulting files back to the experiment engine. It finally deletes the working directories and quits.

4.2 Skeleton Framework: Calcium

A root *calcium-task* is generated for every new skeleton program instance inputted into the framework, and stored in a ready taskpool. A calcium-task is mainly composed of an instruction execution stack, which represents the program's current execution state, and a glue object that corresponds to the result/input of the previous/next instruction.

When a data parallelism instruction is encountered, such as in $fork$, map, and $d\&c$ skeletons, a calcium-task is dynamically subdivided into *calcium-subtasks* which can be computed in parallel. Calcium-subtasks are stored in the ready taskpool and will eventually be computed. Once all calcium-subtasks are finished, their results are returned to the (parent) calcium-task, which can continue with its own execution or generate new calcium-subtasks. When a root calcium-task is finished, it is placed in a finished taskpool, and the result is delivered to the user.

The execution of a skeleton program is thus reduced to the problem of scheduling calcium-tasks in accordance with the skeletons program's logic. The integration between Calcium and the ProActive based scheduler is shown in Figure 3. A *Dispatcher* module, links the ready taskpool with the scheduler interface. Calcium-tasks are wrapped in a `Job` abstraction, which can be submitted to the scheduler. A *Listener* module is in charge of waiting for finished `Job` events from the scheduler. When a `Job` is finished, the Listener modules retrieves the `Job Result` from the scheduler. A `Job Result` can contain finished calcium-tasks or new calcium-subtasks (which where dynamically generated). Using the Calcium back-end API, the Listener module stores the finished calcium-tasks into the finished taskpool, and the new calcium-subtasks into the ready taskpool. As a result, the application invoking the Calcium front-end API is independent of the lower level execution environment. The resources management, scheduling and fault tolerance are handled directly by the scheduler.

4.3 Workflow Engine: SAWE

From the programmer viewpoint, a workflow is described and submitted to SAWE adopting the Workflow definition language XPDL 2.0, an XML-based process modeling language proposed by the WfMC. In XPDL, a workflow consists of one or more activities, which can be actions performed by human actors or automated elaboration performed by computers. Control structures are delivered for basic sequential, parallel, cycle and conditional paths. XPDL is not tied to a specific platform for invoking workflow activities, so allowing the definition of an extensible multi-platform environment in which multiple concrete platforms can be adopted to enact workflow activities.

In this work, we exploited XPDL extensibility and flexibility to easily describe a workflow composed of ProActive-based activities, transparently executed on available resources of a Grid system, without requiring any modification to the workflow language. In particular such applications are defined by specifying: (1) POJO as application type (normally adopted for the invocation of a method on a local Java object); (2) POJO class and method name elements to respectively specify the class and method to execute on a remote resource; (3) extended attributes adopted to use ProActive.

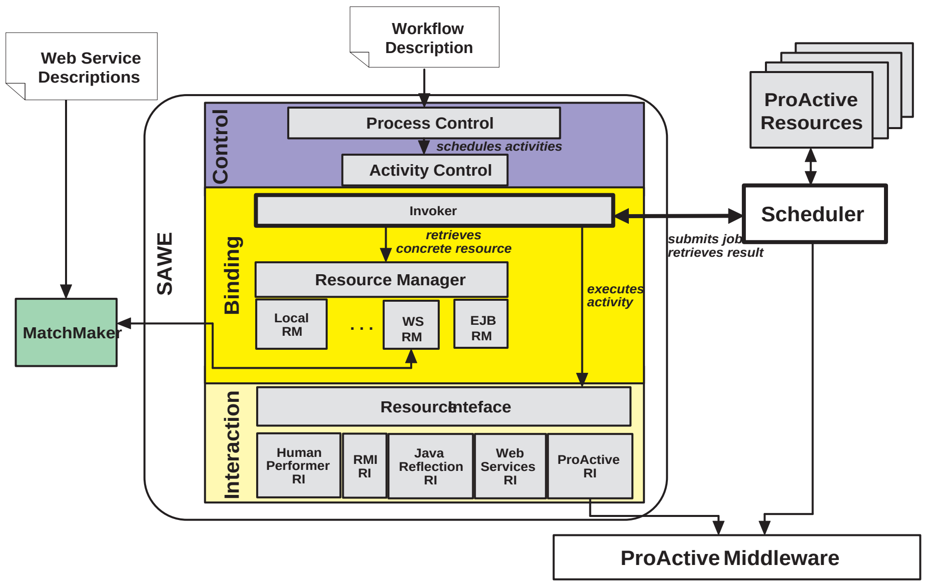

Figure 4: Architecture of SAWE

From the implementation viewpoint, when a workflow is submitted, the $Process\ Control$ component of the control layer retrieves the process definition, navigates the corresponding graph, and chooses the activities that can be enacted, on the basis of data and control dependencies specified in the workflow description. When an activity has to be enacted, the $Activity\ Control$ component passes it to the $Invoker$ component of the binding level, responsible to identify the communication middleware to enact the activity. The Invoker interacts with the scheduler, which will be responsible to deploy, schedule and monitor the execution of the activity (see bold lines and shapes in Figure 4).

Without the support of the scheduler, ProActive-based activities were statically selected by providing the Invoker with the XPDL syntactic description of the task (containing functional description and deployment information). Using such information the Invoker instantiates and configures a ProActive Resource Interface (RI) at interaction layer specialized to interact with the ProActive middleware. Such ProActive RI has the task to deploy an active object on the specified resource, and performs the remote method invocation which corresponds to the required application. By using the scheduler for the deployment and management of ProActive based activities, it is possible to deliver functionalities of dynamic binding of activities to resources. In this case SAWE is able to enact tasks described by means only of a syntactic functional description and whose deployment is dynamically managed by the system.

In particular the scheduler API and information retrieved from the XPDL description are adopted to (1) dynamically define a job, composed of a task including the execution of the specific method specified in the XDPL file; (2) asynchronously submit it to the Scheduler; (3) retrieve execution result when it is required as actual parameter for the enactment of one or more activities of the application workflow.

In the future, we intend to increase binding capabilities of SAWE, allowing the user to specify, in the XPDL workflow description, both functional and quality of service requirements for each activity. To this aim, the Invoker has to interact with a ProActive Resource Manager, which will be responsible, using matching strategies, to choose the resource which grants specified requirements for the execution of the activity.

5 Conclusion and Perspectives

This paper has proposed a flexible super scheduler based on the ProActive Middleware, which delivers scheduling services to different high level programming environments.

The scheduler's features have been validated by integrating three different domain specific high-level programming environments: a problem solving environment (SEGL), a skeleton framework (Calcium), and a workflow enactment system (SAWE). Each environment corresponds to a different CoreGRID partner: HLRS, INRIA, and University of Sannio. The basic features required by these three programming environments have guided the scheduler's design and implementation, and are thus sufficient to factor out complex Grid concerns from these high level programming environments. The global result that emerged is that the super-scheduling of Grid resources may effectively complement the features of the considered programming environments, enhancing their functionalities and increasing the efficiency of Grid resources exploitation.

Nevertheless, some advanced features are still work in progress. For example, a data management abstractions at the scheduler level is missing. This is an important aspect that is currently handled directly by the concerned programming environments. As future work, we are encapsulating the scheduler into GCM components, so that it could be composed in GCM applications and exploited for dynamic deployment of components on Grid resources. Another interesting future direction is a multi-layer integration among the programming environments experimented in this work and the Scheduler. In particular it could be useful to implement the problem solving environment SEGL exploiting the enactment functionalities of SAWE, and to exploit the Calcium skeleton framework to enact in SAWE composite tasks composed following parallel and distributed programming patterns.

Acknowledgments

This work was partially funded by CoreGRID EU Project (FP6-004265) and GridCOMP EU Project (FP6-034442).

References

[1] The Grid Workflow Forum. `http://www.gridworkflow.org`.

[2] GCM deployment standard proposal. TC Grid Meeting and OGF, May 2007. Manchester, UK.

[3] M. Aldinucci and M. Danelutto. Stream parallel skeleton optimization. In *Proc. of PDCS: Intl. Conference on Parallel and Distributed Computing and Systems*, pages 955–962, Cambridge, Massachusetts, USA, November 1999. IASTED, ACTA press.

[4] M. Aldinucci, M. Danelutto, and J. Dünnweber. Optimization techniques for implementing parallel skeletons in Grid environments. In S. Gorlatch, editor, *Proc. of CMPP: Intl. Workshop on Constructive Methods for Parallel Programming*, pages 35–47, Stirling, Scotland, UK, July 2004. Universität Münster, Germany.

[5] M. Aldinucci, M. Danelutto, and P. Teti. An advanced environment supporting structured parallel programming in Java. *Future Generation Computer Systems*, 19(5):611–626, July 2003.

[6] M. Baker, G. Fox, and H. Yau. Cluster Computing Review. *Northeast Parallel Architectures Center, Syracuse University, Nov*, 1995.

[7] Françoise Baude, Denis Caromel, Lionel Mestre, Fabrice Huet, and Julien Vayssière. Interactive and descriptor-based deployment of object-oriented Grid applications. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, pages 93–102, Edinburgh, Scotland, July 2002. IEEE Computer Society.

[8] Eric Bruneton, Thierry Coupaye, and Jean-Bernard Stefani. Recursive and dynamic software composition with sharing. In *Proceedings of the 7th ECOOP International Workshop on Component-Oriented Programming (WCOP'02)*, 2002.

[9] D. Caromel. Towards a Method of Object-Oriented Concurrent Programming. *Communications of the ACM*, 36(9):90–102, September 1993.

[10] D. Caromel, C. Delbe, A. Costanzo, and M. Leyton. Proactive: an integrated platform for programming and running applications on Grids and p2p systems. *Computational Methods in Science and Technology*, 12, 2006.

[11] D. Caromel and M. Leyton. Fine tuning algorithmic skeletons. In *13th International Euro-par Conference: Parallel Processing*, volume 4641 of *Lecture Notes in Computer Science*, pages 72–81. Springer-Verlag, 2007.

[12] Workflow Management Coalition. Xml process definition language, document number wfmc tc-1025. `http://www.wfmc.org`.

[13] M. Cole. *Algorithmic skeletons: structured management of parallel computation*. MIT Press, Cambridge, MA, USA, 1991.

[14] CoreGRID, Programming Model Institute. Basic features of the Grid component model (assessed), 2006. Deliverable D.PM.04, `http://www.coregrid.net/mambo/images/stories/Deliverables/d.pm.04.pdf`.

[15] Natalia Currle-Linde, U. Küster, Michael M. Resch, and B. Risio. Science experimental Grid laboratory (segl) dynamic parameter study in distributed systems. In *PARCO*, pages 49–56, 2005.

[16] M. Danelutto and P. Dazzi. Joint structured/unstructured parallelism exploitation in Muskel. In *Proc. of ICCS 2006 / PAPP 2006*, LNCS. Springer Verlag, May 2006. to appear.

[17] M. Danelutto and P. Teti. Lithium: A structured parallel programming enviroment in Java. In *Proc. of ICCS: International Conference on Computational Science*, volume 2330 of *LNCS*, pages 844–853. Springer Verlag, April 2002.

[18] I. Foster and C. Kesselman. Globus: a Metacomputing Infrastructure Toolkit. *International Journal of High Performance Computing Applications*, 11(2):115, 1997.

[19] Ian T. Foster. The anatomy of the Grid: Enabling scalable virtual organizations. In *CCGRID*, pages 6–7. IEEE Computer Society, 2001.

[20] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, 5(3):237–246, 2002.

[21] W. Gentzsch et al. Sun Grid Engine: Towards Creating a Compute Power Grid. *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002.

[22] D. Hollinsworth. The workflow reference model, workflow management coalition, tc00-1003, 1994. `http://www.wfmc.org`.

[23] MJ Litzkow, M. Livny, and MW Mutka. Condor-a hunter of idle workstations. *Distributed Computing Systems, 1988., 8th International Conference on*, pages 104–111, 1988.

[24] IBM Loadleveler. Using and Administering, version 2 release 1 edition, November 1998. Technical report, SA22-7311-00.

[25] OpenPBS. http://www.openpbs.org.

[26] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, 146(10), 1998.

[27] Giancarlo Tretola and Eugenio Zimeo. Workflow fine-grained concurrency with automatic continuation. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Rhodes, Greece, 2006. IEEE Computer Society.

[28] Giancarlo Tretola and Eugenio Zimeo. Activity pre-scheduling for run-time optimisation of Grid workflows. *Journal of Systems Architecture (JSA) (to appear)*, 2007.

[29] Giancarlo Tretola and Eugenio Zimeo. Activity pre-scheduling in Grid workflows. In *Proceedings of the 15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2007)*, pages 63–71, Naples (Italy), 2007. IEEE Computer Society.

[30] J. Yu and R Buyya. A taxonomy of workflow management systems for Grid computing. *Journal of Grid Computing*, 2(2):2, 2005.

[31] S. Zhou. LSF: load sharing in large-scale heterogeneous distributed systems. *Proc. Workshop on Cluster Computing*, pages 1995–1996, 1992.