Dependable Execution of Workflow Activities on a Virtual Private Grid Middleware

A. Machì, F. Collura, and S. Lombardo

ICAR/CNR Department of Palermo
{machi, s.lombardo, f.collura}@pa.icar.cnr.it

Abstract. In this paper we relate QoS management to Workflow management and discuss dependable execution on a middleware layer, named Virtual Private Grid (VPG), of sets of processes performing a workflow activity. We propose two patterns of interaction between a Workflow Management System and the distributed workflow management support. The patterns monitor resource availability & connectivity, and, in case of fault of any resource, ensure job completion by re-mapping a process graph and restarting it. We also describe current implementation of the patterns and of the run-time support.

1 Introduction

Quality of Service (QoS) can be defined as "the set of those quantitative and qualitative characteristics which are necessary in order to achieve the required functionality of an application" [1]. We can suppose that QoS could be specified as a set of *non-functional* attributes some of which are not expressible numerically.

One category of QoS qualitative characteristics is *dependability* that characterizes the degree of certainty that an activity is performed [2].

Dependability is of main concern in grid environments where applications are ran by coordinating processes mapped on a set of computational resources co-allocated but just virtually co-reserved. In fact, according to the Virtual Organization paradigm, control of grid shareable resources is maintained by Local Organizations and no insurance is given to clients on persistence of their availability.

Even if a number of tools and high level collective services have been developed in the grid community for supporting co-reservation [3], up to now other aspects of grid-awareness as completion insurance, fault-tolerance and resources management are still in charge of user control code. The most popular grid technology for e-science, namely the Globus Toolkit 2 (GT2), and most projects based on it [4][5], offer adequate APIs for *staging* of processes and data, and for process *enactment*, but offer very limited support for process *monitoring* and just a kill mechanism to support workflow *enforcement*. Process monitoring and control are in charge of user who interleaves management code to business code to implement grid-awareness or self-adaptiveness.

The Web Services Resource Framework, emerging from the OGSA (GT3) experience and based on Web Services technology, defines a few models and patterns specifying non-functional requirements of Web Services life cycle and composition [6]. Namely, WS-ResourceLifetime [7] defines mechanisms for managing WS-Resources,

V.S. Sunderam et al. (Eds.): ICCS 2005, LNCS 3516, pp. 267 – 274, 2005.

WS-ServiceGroups [8] describes how collections of services can be represented and managed. WSBaseFaults [9] defines a standard format for reporting exceptions.

But WSRF specifics are limited and focused to the implementation of the Service Oriented Architecture (SOA) paradigm [10] and are scarcely useful to implement QoS control policies and patterns in e-science grid-enabled applications still using legacy code structured as a set of cooperative processes. In this paper we describe a workflow management support service and propose two patterns for supporting dependability of execution.

In section 2 we state equivalence between execution of a process graph and execution of a workflow activity. In section 3 we describe two patterns for management of dependable execution on the grid of sets of concurrent processes described by direct graphs.

Finally, in section 4, we describe a component implementation over GT2 of a grid middleware layer supporting non-functional aspects of workflow management through services for process life cycle monitoring and an asynchronous event bus service.

2 Process Graphs and Workflow Activities

The Workflow Management Coalition organization (WFMC) [11] defines a "*business process*" as "a set of one or more linked procedures or activities which collectively realise a business objective or policy goal" [12]. It defines an *activity* as "a description of a piece of work that forms one logical step within a process. An activity is typically the smallest unit of work that is scheduled by a workflow engine during process enactment. An activity typically generates one or more work items. A work item is a representation of the work to be processed (by a workflow participant) in the context of an activity within a process instance."

We recognize in these definitions a basic model useful for partitioning legacy applications whose business logic can be expressed as a workflow in a set of activities some of which can be separately enacted and managed on the grid. An activity may be carried out by a single workflow participant or by a set of participants organized in a workflow sub-graph. It can be hosted in a SOA service and managed honoring a Service Level Agreement (SLA) [13].

In the following we restrict our considerations to activities that may be described by a direct graph of virtual processes where one virtual node operates as an activity front-end. It coordinates the routing of work-items to other virtual processes participants to activity expressed by workflow sub-graph.

Controlled execution of an activity consists then in an atomic execution on the grid of an instance of a process graph mapped onto grid resources.

Adhering to the terminology commonly used by the WFMC community we can recognize two main control activities:

- *Routing*: the direction of work-items processing flow which is implicitly defined by the process graph and expresses part of the business logic;
- *Management*, which consists in *deployment* of virtual processes onto grid resources, staging of application data, *enactment* (initialization) of activity procedures, *monitoring* of QoS respect, *enforcement* or rerouting of processing flow in case of fault or not compliance with expected QoS.

In the WFMC scenario, management activities are assigned to a WorkFlow Management (WFM) engine holding mechanisms able to enforce workflow rerouting. Present grid toolboxes offer tools for deployment [14], or enactment and monitoring of resource status [15] but do not support collective activity monitoring and re-routing. This is mainly due to their relying on a synchronous communication paradigm (rsh-like) used both for functional and non-functional control.

The Grid Application Development System (GrADS) helps building natively self-consistent applications in which configurable objects, named COPs (Configurable Object Programs), embody at compile time intimate knowledge of application while objects drivers (application manager and executor) embody grid-awareness [16]. GrADS relies on a proprietary run-time system to support QoS monitoring and reconfiguration of COPs through application specific sensors and actuators.

In the framework of the Grid.it project, a new grid-aware programming paradigm has been proposed [17] and the ASSIST environment is being implemented for assisting native development of components configurable in respect to performance attributes. The key point for enabling flexible control is wrapping executive code in components including drivers of standard routing patterns (parmods) and a local proactive executor serving external enforcement through a non-functional interface.

Neither grid-aware programming paradigm defines patterns for integrating control of external activities in theirs application *Managers*.

We propose a grid management support engine that allows application or workflow *Managers* to extend their control on activities on the grid.

In the following we describe a grid middleware offering services for management of the graph of processes on the grid and two patterns implementing dependable execution of workflow actions.

3 Patterns for Dependable Execution of Process Graphs

Fig.1 shows a workflow scenario where two steps of a pipe (activities) execute on resources obtained on-demand from the grid (grid-enabled). Activities are performed by graphs of cooperating processes.

Dependable activity execution is required to avoid overrun of data-item flow-control. If any resource faults, activities need to be remapped and restarted.

Monitoring of stable execution of process activities is committed to a grid middleware. A *grid-front-end* process coordinates a network of sensor/actuator *demons* installed on grid nodes. It offers to the (activity or workflow) *Manager* a high level synchronous RPC functional interface for mapping process graphs, starting them, polling status of their processes, aborting them.

Two interaction patterns allow demons to monitor the regular execution of any process activated on grid resources and *Manager* to restart the graph in case of fault of a grid node or connection. While executing these patterns the *grid-front-end* process and the *demon*s interact synchronously via RPC calls and asynchronously via signals, the *Manager* and the grid front-end just via RPC calls.

The ***resource availability & connectivity check*** pattern verifies, at regular time interval, activity of graph processes and functionality of significant connections among them. Exchanging of *ping messages* between demons at each connection end

tests connectivity. A *demon* notifies to the *grid-front-end* a *missing-connection signal* when unable to reach its peer at end of any significant connection.

Furthermore, each demon emits regularly a *heartbeat* signal to notify the *grid-front-end* that node it monitors is operative. In case of extended absence of the *heartbeat* signal, the *grid-front-end* infers a fault condition, due to the death of the demon, fault of related connection or host shutdown. Successful result of connection test leads to infer a demon fault and register a *process fault* status, unsuccessful result to register a *node fault* status.

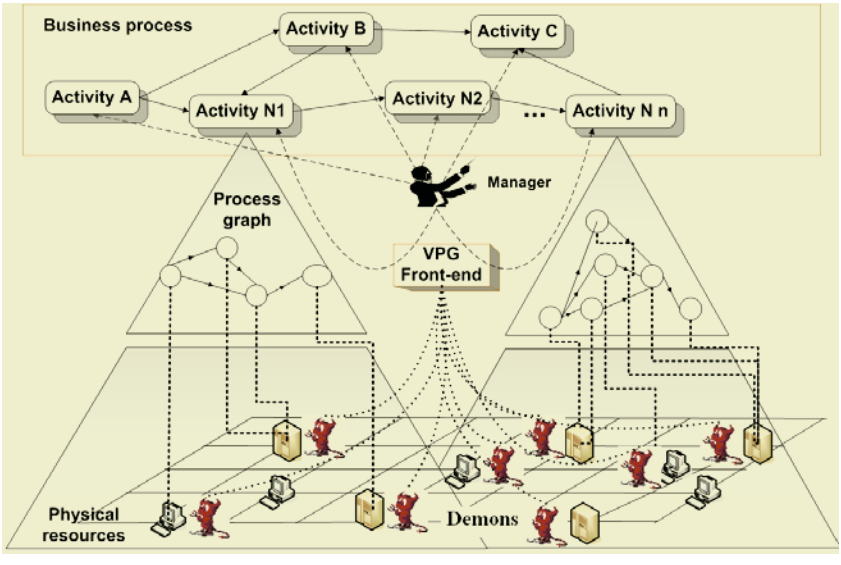

Fig. 1. Actors playing management of workflow activities on the grid

The ***insured completion*** pattern guarantees that a process graph comes to a known end (completed or terminated). *Manager* enacts graph through *grid-front-end* commands and polls the collective status of its process. In case of any node failure it restarts graph execution. Each *demon* monitors processes on its own node. It catches signals emitted by the local operating system and notifies them asynchronously to the *grid-front-end,* that updates graph status on behalf of *Manager*.

The following mechanisms ensure valid knowledge of process vitality and ensured completion:

- *Local operating system signals* inform the *demon* about normal process termination or abort. These signals are notified to the *grid-front-end*.
- *Keep-alive signal*: it is notified regularly from *grid-front-end* to each slave *demon* to maintain its service alive.
- *Automatic shutdown*: a *demon* not receiving for a while the *keep-alive signal* infers a fault of connection with *grid-front-end* node or failure of front-end itself. It kills

any process activated on the node, performs a garbage collection procedure that cleans-up any software element deployed and terminates.

Fig. 2 shows an UML sequence diagram that details an occurrence of the ***insured completion*** pattern after a process fault. Before graph restart, processes on live nodes are explicitly killed. Release of resources allocated on the fault node by the automatic shutdown procedure is omitted.

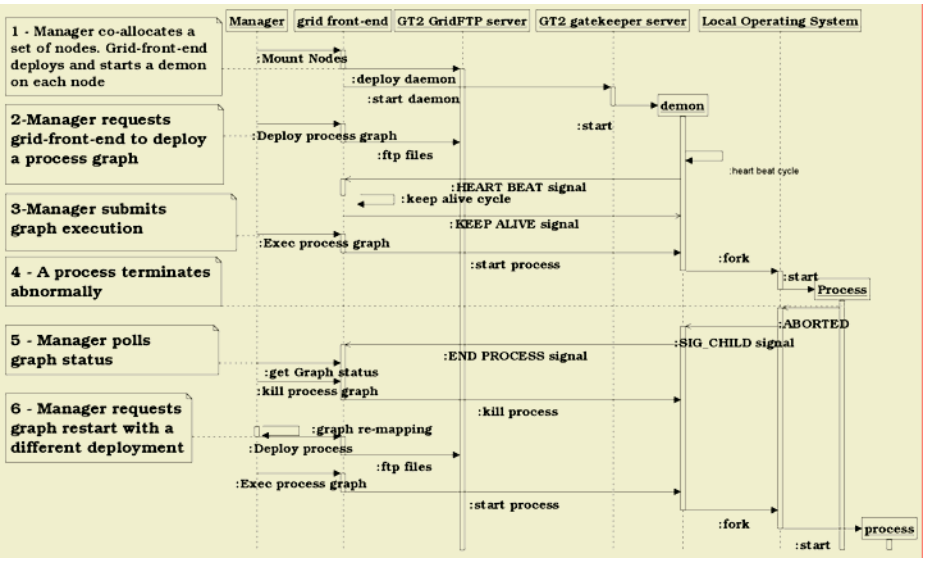

Fig. 2. UML sequence diagram illustrating events and actions in a sample occurrence of the insured completion pattern

4 The Virtual Private Grid Middleware

In the framework of the Grid.it Project we have developed a grid support engine, named Virtual Private Grid (VPG), implementing the roles of *grid-front-end* and *demon* organized as a middleware layer over GT2. VPG manages and monitors the life cycle of process graphs over a pool of grid resources. It offers to a client *Manager* a distributed operating system façade, making grid resources to appear virtually as belonging to a heterogeneous private cluster of processors.

VPG manages multiple process graphs described by lists of processes and arcs connecting them. Attributes of a process are the name of the node on which it is mapped and the list of the software elements required for its execution (executables, libraries and data files). Attributes of the link are the identifiers of processes connected.

Current middleware implementation (VPG 1.2) [18] provides the following functionalities:

- Grid nodes management:
  - ***Mount/Unmount*** of a grid node;
  - ***Retrieve status:*** return activity status of a grid node (disconnected, connected, activated, active).
- Process graph lifecycle primitives:
  - ***Staging*** of a set of files (executable, libraries, etc) on grid nodes. For each file the (redundant) deployment node list is specified. An identifier is assigned to the file set.
  - ***Garbage collection***: removal of any files belonging to a file set from grid nodes where they have been staged.
  - ***Enactment*** of the process graph. An identifier is assigned to each process according to its rank in the process list.
  - ***Kill*** of graph processes included in a list of identifiers.
  - ***Restart*** of graph processes included in a list of identifiers.
  - ***Retrieve status***: return activity status of processes belonging to a process graph (inited, aborted, running, done). Activity progress status is unsupported.
- Event services: an asynchronous event bus is provided for exchanging signals among VPG components according to the event subscribe/notify pattern.

Fig 3 shows, through an UML deployment diagram, VPG components and actors interacting with them. The VPG Master acts as a grid front-end for the *Manager* while slave demons, called Remote Engine run on each node of the grid.

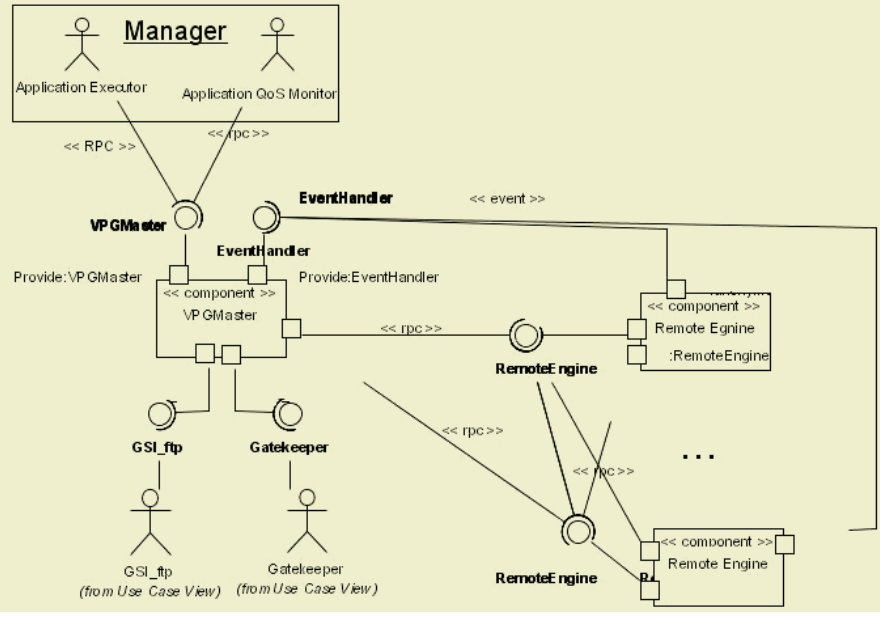

Fig. 3. Interaction among VPG components and actors shown through an UML deployment diagram

VPG Master: provides a functional interface that exposes every of the over mentioned functionalities. It maintains and updates in a register the status of each process controlled. It provides the asynchronous event bus and registers itself for any events related to remote process status changes. The Remote Engine provides local processes lifecycle primitives (run, kill, status, clean), and sends to master events about processes status changing or fault of connections among meaningful neighbor nodes. This entity is automatically deployed during node mount and it is removed during unmount.

Present VPG implementation is based on the usage of several design patterns: acceptor-connector, reactor, proxy, wrapper and adapter provided by the open-source object-oriented framework ACE [19].

Grid nodes management and file transfer are implemented over Globus Toolkit2 tools and services: GRAM for start-up of the Remote Engine, GridFTP for file staging.

APIs for calling VPG Master through synchronous Remote Procedure Calls (RPC) formatted according to the XML-RPC specifications or for direct integration in the *Manager* code, are provided in the VPG 1.2 SDK.

5 Conclusions and Future Work

In the paper we have introduced a methodology for considering QoS management of grid activities in the context of workflow management in SOA architectures. In particular we have discussed the simple case of dependable execution of graphs of processes performing a self-consistent activity.

We have shown that, in this case, two control patterns can be implemented without intervention over the application code, by exploiting an external run-time support offering collective control services to the WFM system.

Control rights over graph processes are obtained by spawning them through remote engines. Restart of the entire process graph has been indicated as a minimal solution for insured execution in case of fault of any grid resources.

A more interesting pattern for insured execution should support assisted recover from a node fault, or at least from a slave node fault.

Design of a pattern for joining or disjoining slaves from a master-slave graph routed by a parallel skeleton is a topic of current research.

Acknowledgements

Work supported by Italian Ministry of Scientific Research: , Project FIRB ***Grid.it.***

References

[1] Vogel A., Kerhervé B., Von Bochman G. Ad Gecsei J. (1195). Distributed Multimedia and QoS: A Survey, IEEE Multimedia Vol. 2, No. 2, p10-19

[2] ITU-T Rec. I.350: General aspects of Network Performance and Quality of Service in Digital Networks, including ISDN.

[3] I. I. Foster, M. Fidler, A. Roy, V, Sander, L. Winkler. End-to-End Quality of Service for High-end Applications. Computer Communications, 27(14):1375-1388, 2004.
[4] The Condor® Project , http://www.cs.wisc.edu/condor/
[5] The DataGrid Project, http://eu-datagrid.web.cern.ch/eu-datagrid/
[6] From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution. K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, S. Tuecke, March 5, 2004.
[7] Frey, J., Graham, S., Czajkowski, C., Ferguson, D., Foster, I., Leymann, F., Maguire, T., Nagaratnam, N., Nally, M., Storey, T., Sedukhin, I., Snelling, D., Tuecke, S., Vambenepe, W., and Weerawarana, S. 2004. WS-ResourceLifetime. http://www-106.ibm.com/developerworks/library/ws-resource/wsresourcelifetime.pdf.
[8] Graham, S., Maguire, T., Frey, J., Nagaratnam, N., Sedukhin, I., Snelling, D., Czajkowski, K., Tuecke, S., and Vambenepe, W. 2004. WS-ServiceGroups. http://www-ibm.com/developerworks/library/ws-resource/ws-servicegroup.pdf.
[9] Teucke, S., Czajkowski, K., Frey, J., Foster, I., Graham, S., Maguire, T., Sedukhin, I., Snelling, D., Vambenepe, W. 2004. WS-BaseFaults. http://www-106.ibm.com/developerworks/library/wsresource/ws-basefaults.pdf.
[10] K.Channabasavaiah, K.Holley, E.M.Tuggle, "Migrating to a service-oriented architecture", http://www-06.ibm.com/developerworks/webservices/library/ws-migratesoa/
[11] The Workflow Management Coalition, www.wfmc.org
[12] Reference Model - The Workflow Reference Model (WFMC-TC-1003, 19-Jan-95, 1.1)
[13] Web Service Level Agreements (WSLA) Project - SLA Compliance Monitoring for e-Business on demand http://www.research.ibm.com/wsla/
[14] R. Baraglia, M. Danelutto, D. Laforenza, S. Orlando, P. Palmerini, P. Pesciullesi, R. Perego, M. Vanneschi “AssistConf: a Grid configuration tool for the ASSIST parallel programming environment”, Eleventh Euromicro Conference on Parallel, Distributed and Network-Based Processing February 05 - 07, 2003 Genova, Italy
[15] WS GRAM Docs, http://www-unix.globus.org/toolkit/docs/3.2/gram/ws/ index.html
[16] The Grid Application Development Software Project (GrADS), http://hipersoft.cs.rice.edu/grads/index.htm
[17] M. Aldinucci, S. Campa, M. Coppola, M. Danelutto, D. Laforenza, D. Puppin, L. Scarponi, M. Vanneschi, C. Zoccolo "Components for high performance Grid programming in the Grid.it Project". In Proc. Of Intl. Workshop on Component Models and Systems for Grid Applications.
[18] S. Lombardo, A. Machì. “Virtual Private Grid (VPG 1.2) Un middleware a supporto del ciclo di vita e del monitoraggio dell’esecuzione grafi di processi su griglia computazionale”, Technical report RT-ICAR-PA-11 -2004, October 2004
[19] The ADAPTIVE Communication Environment (ACE) http://www.cs.wustl.edu/ ~schmidt/ACE.ht