Project no. 004758

GORDA

Open Replication of Databases

Specific Targeted Research Project

Software and Services

Management Tools Set

GORDA Deliverable D5.4

Due date of deliverable: 2008/03/31
Actual submission date: 2008/04/21

Start date of project: 1 October 2004 Duration: 42 Months

INRIA

Revision 1.0

Project co-funded by the European Commission within the Sixth Framework		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Contributors

Alfrânio Correia Junior, U. Minho
Christophe Taton, INRIA
Florent Mtral, INRIA
José Pereira, U. Minho
Luís Soares, U. Minho
Miguel Matos, U. Minho
Nuno Carvalho, U. Lisboa
Ricardo Vilaça, U. Minho
Rui Oliveira, U. Minho
Sara Bouchenak, INRIA
Sylvain Sicard, INRIA

Abstract

This document provides an overall description of the GORDA Management System and details the various modules that have been developed to configure and deploy, monitor, recover and optimize a GORDA database cluster.

Contents

Chapter 1

Introduction

In this document we describe the set of management tools developed to administer replicated database systems based on the GORDA architecture. These tools are composed of different management modules and tools as follows:

- a configuration and deployment module that helps the administrator in setting up the replicated database configuration
- a monitoring module to report accurate resource usage of the various system components
- an automatic recovery module
- a dynamic optimization module
- a supervision console to reflect the dynamic state of the cluster including failure reports and alarms, as well as commands to reconfigure the replicated databases to accommodate runtime needs or configuration changes.

The software packages for the modules and tools can be found at:
http://gorda.di.uminho.pt/community/GMS

1.1 Objectives

The goal of this document is to provide an overall description of the GORDA Management System and to detail the various modules that have been developed to configure, manage and monitor a GORDA database cluster.

1.2 Relationship with other deliverables

The deliverable develops on the GORDA Architecture previously described in Deliverable D2.2 - Architecture Definition Report. The developed tools interface with

several modules described in D3.3 - Replication Modules Reference Implementation, D3.4 - Description and Configuration Guide and D4.3 to D4.5, which correspond to the GORDA compliant DBMSs descriptions.

Chapter 2

Overview of the GORDA Management Service

The GORDA Management Service (GMS) is based on Jade, a framework that provides autonomic management capabilities for distributed systems [14]. GMS tackles (i) the management of legacy database systems, and (ii) the management of clusters of replicated database systems.

Legacy systems. By a legacy system, we mean a black-box with an immutable interface, on which no assumptions may be made. Usually, legacy system management solutions are implemented as ad-hoc solutions that are tied to a particular system [17,16,15]. Unfortunately, with such an ad-hoc approach, system management solutions need to be re-implemented whenever a new legacy system is taken into account.

Clusters. The complexity of the architecture of distributed systems makes administration tasks harder. In a cluster of replicated database servers, providing, for instance, self-recovery upon a server failure does not simply imply restarting the failing server as an independent element, but also rebuilding its state, rebuilding the connections of that server with other replicated servers, etc. This requires the knowledge of the global architecture of the system in order to perform reconfigurations of several elements of the system; and such an architectural view is not provided by cluster-based systems.

GMS is based on the Jade framework which defines autonomous management actors (Autonomic Managers, or AMs), which apply an administration policy of a specific aspect to Managed Elements (e.g. self-recovery, self-optimization, self-protection, etc.). Managed Elements (or MEs) are administrable elements, which may be legacy systems. Any piece of legacy system is encapsulated in a well-defined software component, called a Jade wrapper, which provides a uniform management interface. This provides the Jade/GMS autonomic management environment a homogeneous architectural view of the underlying heterogeneous managed database legacy systems, and thus a generic way to administer these systems (see Figure 2.1).

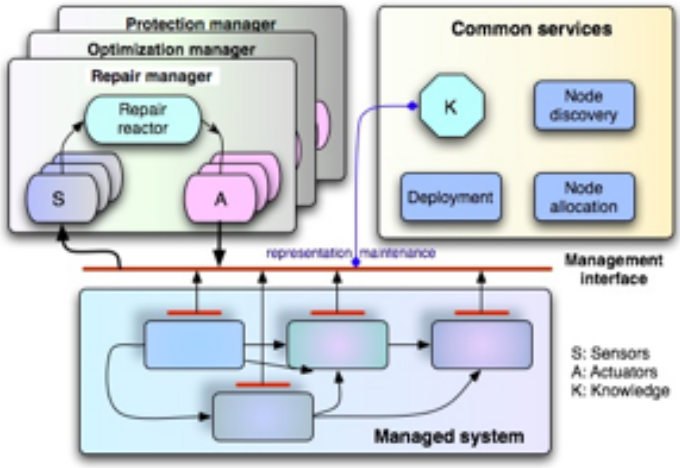

Figure 2.1: Jade autonomic management framework.

Autonomic Managers (AMs) apply an administration policy to a specific aspect (e.g. self-recovery, self-optimization, self-protection, etc.). Autonomic Managers are based on feedback control loops that first observe and monitor the managed system via *sensors*, then analyze the monitoring data and decide which reconfiguration operations need to be applied on the managed system, before actually applying those operations through *actuators*.

Autonomic Managers administer systems that are made out of Managed Elements. A Managed Element (ME) is any piece of system that needs to be self-managed; it is associated with Autonomic Managers via sensors that monitor the ME, and actuators that dynamically adapt the ME. The uniform management interface provided by the Jade wrapper that encapsulates a ME allows the following actions to be performed on the managed system (be it an individual component or the whole system):

- Configure the component, i.e., assign values to its attributes.
- Manage the component's lifecycle (e.g., start, suspend, stop).
- Inspect the component's properties.
- For a composite component (a component that includes other components such as a cluster of replicated elements), manage its contents, e.g., by adding or removing a subcomponent.

The notion of a composite component allows a complex sequence of actions to be performed by a single operation, e.g. recursively stopping all the components of an application, or configuring all or part of the nodes of a cluster.

Moreover, a set of common services in the management framework provide various functions needed for the construction of the autonomic managers. The common services provide the following functions:

- Discovering available nodes in the cluster.
- Allocating a node, and installing a set of components on that node.
- Deploying a system on a set of nodes, using the above services.

Chapter 3

Dynamic configuration and deployment

Deployment is the process of installing, configuring, and starting up the system on a given platform. In line with the general philosophy of Jade, we propose an architecture-based approach to deployment, i.e., we use an architectural model of a system as a guide for the description, the construction, and the deployment of the replicated system configurations, as shown on Figure 3.1.

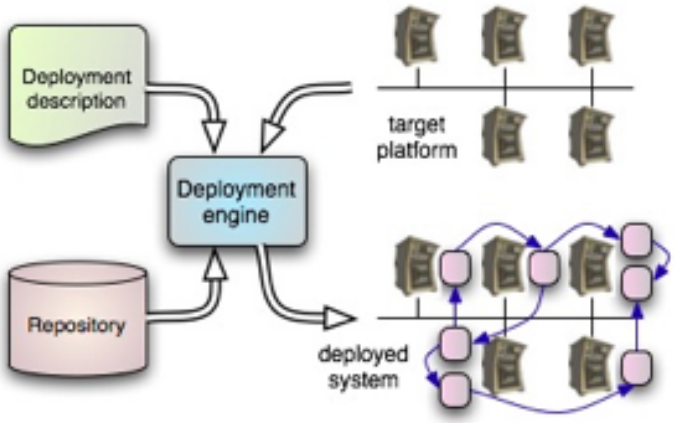

Figure 3.1: Configuration and deployment.

The elements that make up this process are organized as follows.

- The deployment description contains all the information needed by the deployment engine to instantiate a Jade-enabled application, i.e., an application made up of components (with native interfaces or equipped the wrappers) manageable by Jade. The deployment description, written in Fractal ADL,[1]

[1]Fractal is the component model underlying the Jade framework to represent Managed Ele-

consists of the structural description of the application in terms of interconnected components, enhanced by deployment-specific information, such as co-location requirements and starting order.

- The deployment engine is derived from a Fractal ADL factory. It operates as a workflow engine, in which a compiler creates tasks that are executed by backend components, specialized for the various deployment functions.
- The current repository, which contains the software components to be deployed, is based on the OSGi Bundle Repository. Thus OSGi provides a uniform packaging format for the Jade components.
- The deployment target is a set of nodes, equipped with support for component instantiation, installation, versioning, updating, and removal. Since components are packaged as OSGi bundles, these functions are provided by an OSGi layer.

3.1 Configuration tools

This provides a set of tools used in preliminary steps that the administrator must take in order to properly set up the cluster environment. For instance, he has to define the number of replicas their IP addresses and port numbers and the GAPI enabled database that each replica will run. All this information must then be specified to each one of the services provided by the GORDA architecture namely the GORDA Replication Service, the GORDA Communication

Service and the GORDA Management Service.

The configuration tool makes use of the Fractal Architecture and configuration Description Language (ADL), in the form of XML-based configuration files. The configuration files allow to specify several things such as the cluster elements (i.e. the nodes constituing the cluster), the set of replicated database servers, configuration parameters of database servers (e.g. user name and password, maximum number of connections to the database, etc.).

Configuration files also allow to associate autonomic management policies to the set of replicated databases, such as automatic recovery of failing databases (see Chapter 5), or dynamic optimization of the performance of the cluster of databases (see Chapter 6).

The configuration tool is also used to configure installation of software elements that make the GAPI enabled system. This allows to set up the repositories that host the bundles containing the core components of the GORDA architecture. The dependencies between the bundles are also configured so they can be dynamically and automatically deployed by the Deployment Module.

Finally, the Supervision Console must be configured to point to the cluster and its replicas, thus offering a global management tool of the system (see Chapter 7).

ments [11].

3.2 Deployment module

The deployment module is built around Oscar/OSGi Bundle Repository (OBR) model [6]. OBR provides means to install bundles along with its set of transitive dependencies into a running OSGi framework. By using a model of federated repositories, in the sense that each individual repository is registered with a Repository Administration Service, the concept of a central repository does not exist. Instead multiple repositories can be registered and will be seen from the client's point of view as a single repository and dependencies between bundles hosted on different repositories will be resolved transparently by OBR.

Each repository is described by a XML-like file representing the bundle meta-data, that describes the dependencies among the bundles, called resources in the OBR language. Each resource is described in detail by a set of meta-data [obr-metadata] but the most important pieces of it are related to the requirements/capabilities of each resource. The requirements express whose resources are needed for a successful deployment of this resource (dependencies) and the capabilities indicate the packages that the resource may export along with any service it may provide. In those descriptions it is also possible to state either exact or range versions of the resources required that act like version filters, thus allowing for an increased flexibility that will allow the dependency resolver to properly track down each needed resource and deploy it accordingly.

As an example, for the test scenarios presented in GORDA Deliverable D7.1, an OBR repository, hosted as a HTTP service, as been built with the following components:

- properties: creates a "Properties" service providing configuration parameters to the other bundles;
- gorda.pgsql: provides the Postgresql/G binaries;
- pgdata-schema: provides the empty schema of the TPC-C benchmark and starts Postgresql/G. Requires gorda.pgsql bundle;
- pgdata-full: provides the database of the TPC-C-light[2] benchmark populated with 20 clients and starts Postgresql/G. Requires gorda.pgsql bundle;
- scenario1: provides Replicator software and starts the replicator configured to scenario 1. Requires pgdata-full.
- scenario2: provides Replicator software and starts the replicator configured to scenario 2. Requires pgdata-schema.

[2] TPC-C-light is a *in-house* variation of the TPC-C benchmark that reduces the proportion between the number of clients and the size of the tables' entries thus reducing the overall size of the database and system load.

The OBR functionality of the deployment module has been extended to export its operations through JMX thus providing the remote deployment capabilities needed by the Automatic Recover and Dynamic Optimization modules.

Chapter 4

Monitoring module

A module to report accurate resource usage of the various system components, such as statistics about the replicated database behaviour: high level statistics (transaction abort rate), middleware level statistics (GORDA internal services),and low level statistics (cpu/memory usage).

The monitoring module is built atop of Java Management Extensions (JMX) [18] and allows to monitor the resources and services offered by the GORDA. The module collects and aggregates statistics about the provided services and resources making that data available to other modules.

Each resource is mapped to a high-level entity called a Managed Bean (Mbean for short) that provides get() and set() methods to access the resource's properties, operations to perform relevant tasks and event registration facilities to notify its listeners when a given event occurs. MBeans are registered in to a JMX server that makes them available to remote management applications.

There are two different kinds of resources currently being monitored: those that are accessed directly by inspecting the value of some variables in the running Java code, and those that require an interaction with the operating system through the Java Native Interface (JNI) which allows code running in the JVM to call native code specific to the operating system and hardware. This is needed to obtain information about the storage, network and cpu and is instrumented in such a way that is completely transparent to the clients consulting the MBeans to which those resources are mapped.

Sensors/actuators MBeans are currently provided by the ESCADA Toolkit and by the JADE OSGi-enabled framework, also known as Jade Boot.

Jade Boot has been instrumented with actuators that provide facilities to control the life-cycle of the components and with sensors that provide information about the state (uninstalled/installed/started/stop) of the components present in the platform. Furthermore the OSGi logging facility has been extended to provide remote access, via JMX , to the logs of the running components and the platform itself.

The ESCADA Toolkit provides sensors to access statistics about the database instances like the number of started/stopped/queued messages, about the state of

the global environment, like the number of clients present in the cluster and about the group membership service.

Additionally, the ESCADA Toolkit provides a custom BSD-based socket protocol that allows applications lacking JMX support to access its state. Currently it recognizes the following set of operations:

- **LIST SENSORS**: retrieves which sensors are available.
- **LIST SENSORS ATTRIBUTES sensorId**: retrieves the set of attributes passible of being monitored by sensorId.
- **GET sensorId attrId**: gets the value of a specific sensorId attribute attrId.
- **SET sensorId attrId value**: sets the value of a specific sensorId attribute attrId.
- **OPERATIONS sensorId** retrieves the set of operations that can be invoked on sendorId.
- **INVOKE sensorId operationId**: invokes operationId on sensorId.
- **CREATE REPLICA replicaId**: adds a new replica to the cluster.
- **STOP REPLICA replicaId**: removes a replica from the cluster.
- **PAUSE REPLICA replicaId**: pauses execution of a replica.
- **CONTINUE REPLICA replicaId**: resumes execution of a replica.
- **REGISTER LISTENER sensorId**: registers for notification of sensorId events.
- **UNREGISTER LISTENER sensorId**: unregister for notification events.
- **NOTIFY sensorId attrId value**: issue whenever a notification event is sent to a listener.

Chapter 5

Automatic recovery module

The goal of Automatic Recovery Module is to restore the integrity of a system in the presence of failures. We consider cluster nodes the unit of failure and by recovery we mean the full recovery of whole nodes.

The recovery module contains a representation of the system state, including the configuration of each component, the bindings between the components, and the placement of the components on the nodes of the cluster. This representation contains all the necessary information to reconstruct the system after a node failure.

The recovery process includes the following steps, according to the generic model of a control loop for autonomic computing.

- Failure detection, explicating using a heartbeat technique or through a membership service.
- Repair decision and planning, including: allocating a new available node; re-creating and re-configuring new instances of the failed components (those that were located on the failed node); restarting the restored components; updating the system representation in the knowledge base.
- Performing the actions defined above, using the services provided by the management framework: the node allocation service and the configuration and deployment service, this latter using the configuration and reconfiguration interface provided by the wrapped components.

More precisely, the automatic recovery module uses the monitor module to learn about the state of the group membership and the service provided by the deployment module to deploy a new replica when one of the existing replicas has failed.

In detail it registers a listener to watch for changes in the group membership with the monitor module provided by the ESCADA Toolkit. When a replica fails this module receives a proper notification and will try to replace the failed replica by deploying another one that is available. In this context, an available replica, is

one that is reachable (i.e. its JadeBoot service is accessible) and is in a state other than started. If such replica exists this module will instruct its deploy module to start it thus triggering the recovery process for that replica.

The recovery process starts every time a replica joins a group and makes use of the group membership service offered by the underlying group communication system (JGCS). When a replica joins the group, a blocking notification is sent by JGCS and all replicas that receive this notification stop sending messages. Meanwhile, a new notification is sent by the JGCS informing which is the new replica and indicating that messages that were sent while the new replica was joining the group were delivered. After this, the buffered messages should be processed as well as any new message. However, the new replica may not be able to process transactions as its database may be out-of-sync with the others. Considering so, the recovery process in the joined replica chooses one or more replicas as database sources (as our current implementation supports state transfer from multiple donors) and carries out a state transfer.

Depending on the amount of divergence between the donors an the new replica's state, state transfer can encompass the transfer of the whole database or just the staled or inexistent data.

Chapter 6

Dynamic optimization module

The goal of the dynamic optimization module is to optimize system performance in spite of variations on system load or of available resources. Performance may be measured by various criteria, such as average database server response time, average throughput, transaction abort rate, etc.

In our first experiments, we have selected an entire node as the resource allocation unit, and we have implemented a simple control loop based on thresholds heuristics [12]. When the load (measured by cpu or memory usage) goes over a preset threshold, a new node is allocated. When the load falls under another threshold, a node is released. An instance of this control loop is associated with each group of replicated databases. As a result, the system keeps the response time fairly stable under wide variations of the load, and attempts to maintain cluster performance near-optimal.

We also conducted other experiments where the dynamic optimization module calculates the optimal configuration of the cluster of replicated databases in spite of load variation [20]. This approach is based, on the one hand, on modeling of servers to characterize their behavior, and on the other hand, on capacity planning algorithms that implement multi-criteria utility functions that maximize several performance criteria (e.g. minimize transaction response time, maximize throughput, minimize transaction abort rate, etc.).

This module is closely tied with the monitoring and deployment modules, by gathering information about the cluster from the first and then performing the adequate operations on the second, based on policies that should specify the desired behaviour of the system accordingly to the environmental and service demand changes.

The policies are defined in independent and modular scripts that can be plugged dynamically into the module. The module would then take the necessary steps to initialize the proper environment for the script to run. These steps involve establishing a connection with the monitoring module, usually provided by the ESCADA Toolkit to access the desired MBeans and its provided attributes and additionally contacting the deployment module to invoke the operations that would control the

life-cycle of the replicas. After this initialization process is done each policy script will be run at regular time intervals, defined by the administrator, in order to enforce the desired policies.

Normally the execution of each script consists in the following steps:

- Gather data from all the required MBeans;
- Build overview of the environment by applying proper metrics and/or operations, for example by calculating the global average load;
- Choose the adequate action(s) to take in face of the actual environment that will enforce the desired policy;
- Apply those action(s) via the deployment module instructing it, for example, to start or stop a given replica.

The above steps describe the general *recipe* to use to dynamically optimize the cluster as a whole by macro-managing it. However due to the flexibility of the architecture developed it is also possible to take the level of dynamic optimization to a micro-management approach. In this approach is possible to continually optimize a replica in accordingly to its requirements, for example by fine tuning the number of clients allowed in face of the current load. In this approach this module will interact with the actuators provided by the monitor module of that replica instead of its deployment module.

By combining these two approaches it is possible to dynamically adapt the cluster to many different situations and demands ensuring that it will continue to behave as expected and providing the service as efficiently as possible.

Chapter 7

Supervision console

The Supervision Console is the frontend to the management tool set and provides a comfortable, easy to use, web-based interface to monitor and manage the cluster and is based on a highly customized version of the jManage 2.0 platform.

jManage is an open source management platform that provides a centralized console to manage application clusters and distributed applications. It provides a JMX interface to access other components, a framework to draw graphs and dashboards that are refreshed automatically to show the latest information available, support for notifications and email-alerts, guarantees production level security according to its authors, supports cluster applications such as changing attribute values and performing of operations and has a connector framework to support non-JMX based applications.

Our customized version of jManage can provide a global overview of the cluster (Figure 7.1) as well as a detailed view of each replica (Figures 7.2, 7.3 and 7.4). It also offers an interface to perform cluster-wide operations such as starting clients (for testing purposes), starting and stopping replicas, perform consistency checks, etc. (Figure 7.2)

In the global view of the cluster the following information is displayed as graphs:

- Incoming transactions per second;
- Latency;
- ClientPerformance;
- AbortRate;
- CommitRate.

and the following operations are available:

- Populate cluster;

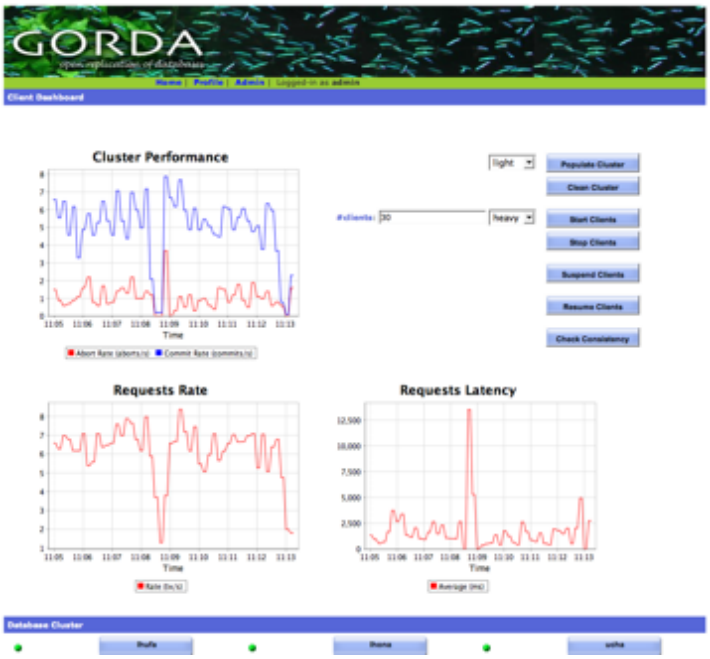

Figure 7.1: Supervision console – Global view of the cluster.

Figure 7.2: Supervision console – Detailed view of a replica (1/3).

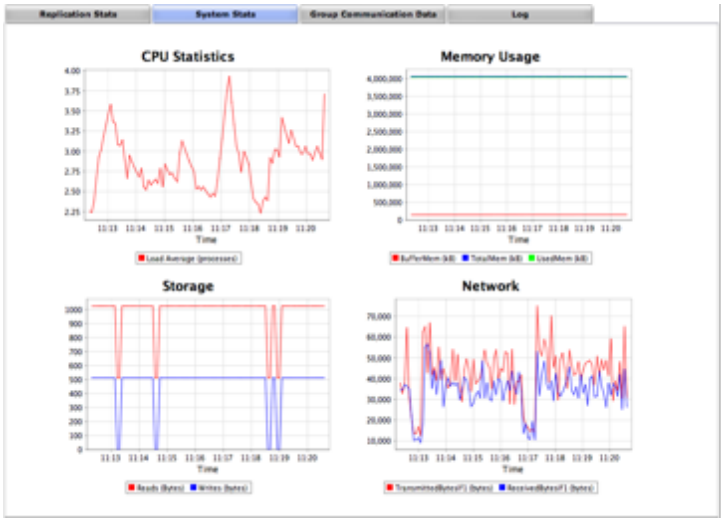

Figure 7.3: Supervision console – Detailed view of a replica (2/3).

Figure 7.4: Supervision console – Detailed view of a replica (3/3).

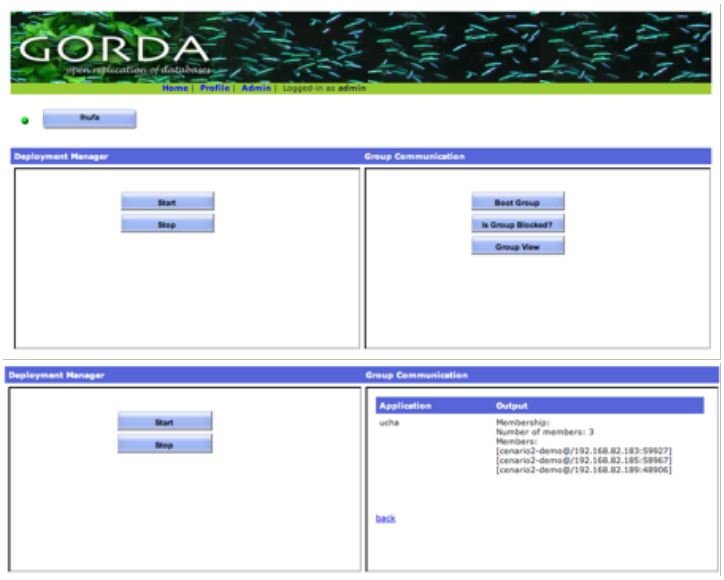

Figure 7.5: Supervision console – Cluster management operations.

- Clean cluster;
- Start Clients;
- Stop Clients;
- Suspend Clients;
- Resume Clients;
- Check Consistency.

The options related to the clients (start/stop/resume) are available for testing purposes only as they don't make sense in a production environment.

For each replica along the start and stop replica the following operations are available:

- BootGroup: which shows the constitution of the boot group;
- View: which shows the members in the current view;
- IsViewBlocked: which indicates either the view is blocked or not.

and the following statistics are displayed, grouped in four tabs that aggregate related information:

- Replication Stats;

- QueuedMessages
- Incoming Messages
- LocallySubmitedMessages (aborted vs committed)
- System Stats;
- CpuAverageLoad
- MemoryUsage
- StorageUsage
- NetworkUsage
- Group Communication Stats;
- Messages per Second
- Bytes per Second
- Log - displays the messages logged by the replica.

References

[1] S. Elnikety, S. Dropsho and F. Pedone. Tashkent: Uniting Durability with Transaction Ordering for High-Performance Scalable Database Replication. ACM Eurosys, Lisbon, Portugal, April 2006.

[2] Ruiz-Fuertes, M.I. Pla-Civera, J. Armendariz-Inigo, J.E. de Mendivil, J.R.G. Munoz-Escoi, F.D. Revisiting Certification-Based Replicated Database Recovery. The 9th International Symposium on Distributed Objects, Middleware, and Applications (DOA), Vilamoura, Algarve, Portugal, Nov. 2007.

[3] E. McManus. Java Management Extensions. Java Community Process (JSR'3), 2006.

[4] GORDA Consortium. GORDA: Open Replication of Databases. Oct. 2004, http://gorda.di.uminho.pt/

[5] JManage: Open Source Application Management. http://jmanage.org/

[6] Oscar Bundle Repository. http://oscar-osgi.sourceforge.net/repo/bundlerepository/

[7] N. Carvalho, A. Correia Jr. and J. Pereira, L. Rodrigues, R. Oliveira and S. Guedes. On the use of a reflective architecture to augment Database Management Systems. Journal of Universal Computer Science, 2007.

[8] ESCADA Replicator. 2007. http://sourceforge.net/projects/escada/

[9] H. Miranda and A. Pinto and L. Rodrigues. Appia: A Flexible Protocol Kernel Supporting Multiple Coordinated Channels. 21st International Conference on Distributed Computing Systems (ICDCS 2001), Poster, April 16-19, 2001, Phoenix, Arizona, USA.

[10] F. Pedone and R. Guerraoui and A. Schiper. The Database State Machine Approach. Journal of Distributed and Parallel Databases and Technology, 2003.

[11] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma and J. B. Stefani. The FRACTAL component model and its support in Java. Journal of Software Practice & Experience, Vol. 36, 2006.

[12] S. Bouchenak, N. De Palma, D. Hagimont, S. Krakowiak and C. Taton. Autonomic Management of Internet Services: Experience with Self-Optimization. Third International Conference on Autonomic Computing (ICAC 2006), 2006.

[13] The OSGi Alliance. OSGi Service Platform - Core Specification. 2005. http://osgi.org/osgi_technology/download_specs.asp

[14] Jade. *Jade: A Framework for Autonomic Management*. http://sardes.inrialpes.fr/jade/jade/jade-a-framework-for-autonomic-management.html

[15] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, A. Fox. A Microrebootable System: Design, Implementation, and Evaluation. *6th Symposium on Operating Systems Design and Implementation (OSDI-2004)*, San Francisco, CA, Dec. 2004.

[16] M. Y. Luo, C. S. Yang. Constructing Zero-Loss Web Services. *IEEE Conference on Computer Communications (INFOCOM-2001)*, Anchorage, Alaska, Apr. 2001

[17] Y. Saito, B. N. Bershad, H. M. Levy. Manageability, Availability and Performance in Porcupine: A Highly Scalable, Cluster-Based Mail Service. *ACM Transactions on Computer Systems*, 18(3), pp. 298-332, Aug. 2000.

[18] Sun Microsystems. *Java Management Extensions (JMX)*. http://java.sun.com/jmx/

[19] T. Abdellatif, J. Kornas, and J. B. Stefani. J2EE Packaging, Deployment and Reconfiguration Using a General Component Model. In *3rd International Working Conference on Component Deployment (CD 2005)*, Grenoble, France, November 2005.

[20] J. Arnaud and S. Bouchenak. Resource Management in Internet Services. *Conférence Française en Systèmes d'Exploitation (CFSE), ACM-SIGOPS France*, Fribourg, Switzerland, Feb. 2008.