NLE-GRID T1 – Architectural Model

Natural Language Engineering on a Computational Grid
POSC/PLP/60663/2004

— INESC-ID Lisboa Tech. Rep. 30/2008 —

David Martins de Matos, Tiago Luís, Ricardo Ribeiro
L^2F – Spoken Language Systems Laboratory
INESC ID Lisboa, Rua Alves Redol 9, 1000-029 Lisboa, Portugal
{david,tiago,rdmr}@l2f.inesc-id.pt

This report describes the architecture of the grid infrastructure for Natural Language Engineering applications used in the project. This infrastructure uses a web-based framework for interaction, at the upper level, and specialized connectors to provide interaction with the lower levels. Although web services play an important role, they are not the only possibility for providing services. Services can also be provided by other backends, as long as the appropriate gateway exists. The emphasis is in simplicity and use of concepts familiar to the end users (both final users and service providers).

This revision: January 30, 2008

NLE-GRID T1: Architectural Model
Natural Language Engineering on a Computational Grid
POSC/PLP/60663/2004

David Martins de Matos, Tiago Luís, Ricardo Daniel Ribeiro

L^{2}F – Spoken Language Systems Laboratory
INESC ID Lisboa, Rua Alves Redol 9, 1000-029 Lisboa, Portugal
{david,tiago,rdmr}@l2f.inesc-id.pt

Abstract. This report describes the architecture of the grid infrastructure for Natural Language Engineering applications used in the project. This infrastructure uses a web-based framework for interaction, at the upper level, and specialized connectors to provide interaction with the lower levels. Although web services play an important role, they are not the only possibility for providing services. Services can also be provided by other backends, as long as the appropriate gateway exists. The emphasis is in simplicity and use of concepts familiar to the end users (both final users and service providers).

The Galinha system [5] was originally developed by the Spoken Language Laboratory (L^{2}F) of INESC-ID in an effort to simplify the creation of natural language engineering (NLE) applications. The original Galinha system consisted of a web interface and a data repository. Applications were built through the web interface by creating service chains from a pool of re-usable components. Various tools were integrated in Galinha: these tools performed tasks such as morphological analysis, part-of-speech disambiguation, and syntactic analysis.

The main objective of the NLE-GRID project is to create a framework for high performance NLE computing on a computational grid by extending the Galinha system. The concrete objective of the project is the extension of the Galinha architecture (the G8 core) to include an interface to computational grid services so that the components and data can be geographically and organizationally distributed. Figure 1 illustrates the overall architecture: the G8 core and the execution daemon ("Tick") are the core [9], while the Web, Condor [11], and Hadoop [12] show possible service backends.

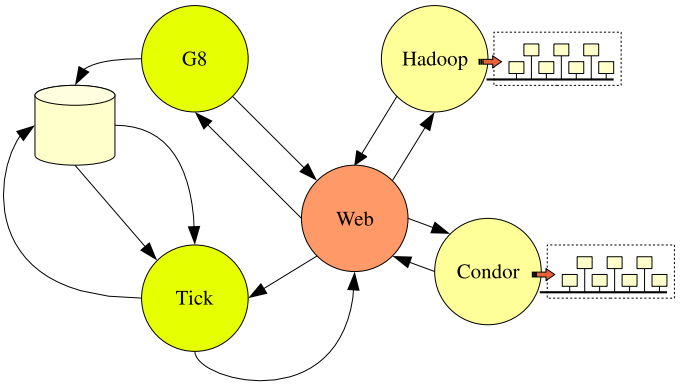

Fig. 1. Block diagram of the overall architecture.

Figure 2 shows the overall functional architecture. The upper layer (G8 API) handles requests from client applications (which include the web interface – see below) and communicates through selected middleware with the underlying application back-ends. The rightmost column represents the fact that the G8 core layer can also re-export any imported services, thus appearing as any other application back-end. Client applications invoke G8 via web services or AJAX calls [8].

The objective of this report is to define the computational architecture of a grid node in the context of the NLE-GRID project. The following sections present the upper layer interface (the G8 API); the service connectors, allowing multiple backend types to be connected to the G8 core; and, in selected cases, how to integrate particular modules in contexts that depend on a specific connector.

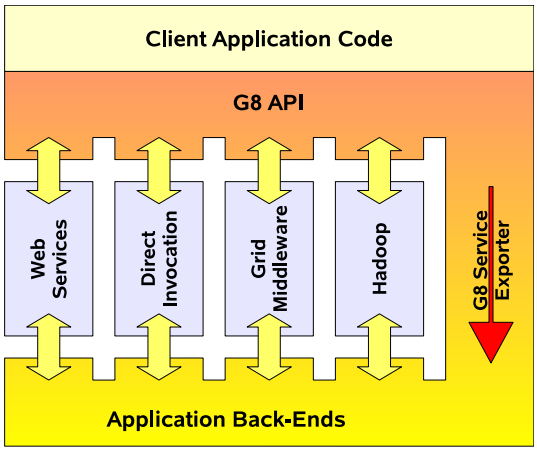

Fig. 2. Functional overview of the architecture.

1 Upper Level Interface

The upper level interface may be thought of as a service interface. This interface exports all the underlying services known to the system. Even though the underlying services may be provided by different backends, this is not apparent in the exported interface, regardless of the actual export protocol.

In addition to the client code interface, the upper layer also contains a interface to access and control the G8 core itself. These calls are used to allow the core to be extended. An example is the G8 interface itself [9].

The chosen protocol, in the current prototype, is the Simple Object Access Protocol (SOAP) [16], commonly known as web services. This choice is motivated by the wide availability of web service implementations. In addition, since their interfaces are self contained, web services are easily used as building blocks in application contexts.

2 Service Connectors

Service connectors abstract services provided at lower levels and unify their interfaces in terms of what is needed by the upper levels. In this way, base services may be successfully combined without regard for their differences.

2.1 Galaxy Communicator

The Galaxy Communicator system [10] was initially selected to provide messaging support for the Galinha infrastructure's message exchanges. Galaxy has a distributed hub-and-spoke message-based architecture intended to facilitate building spoken dialogue systems. The initial reason for selecting Galaxy was it being already in use at the Spoken Language Systems Laboratory for building domotic interactions and because the new purpose did not in any way negatively affect existing uses. This conjunction of factors allowed us to easily migrate/integrate existing modules in the new framework with minimal repercussions.

Since Galaxy did not have any special protocol for communicating with non-Galaxy applications, it was necessary to create, on the one hand, such a protocol [4], and, on the other hand, a gateway server,

to account both for Galaxy requirements and the requirements of the outside world, i.e., the protocol for communicating with the higher levels of the Galinha interface for translating Galaxy frames into Galinha messages and vice-versa. The protocol and the gateway server enable the higher levels to directly control fine aspects of the Galaxy environment (e.g. execution of single calls on a given Galaxy server). Figure 3 illustrates the interaction with the Galaxy infrastructure.

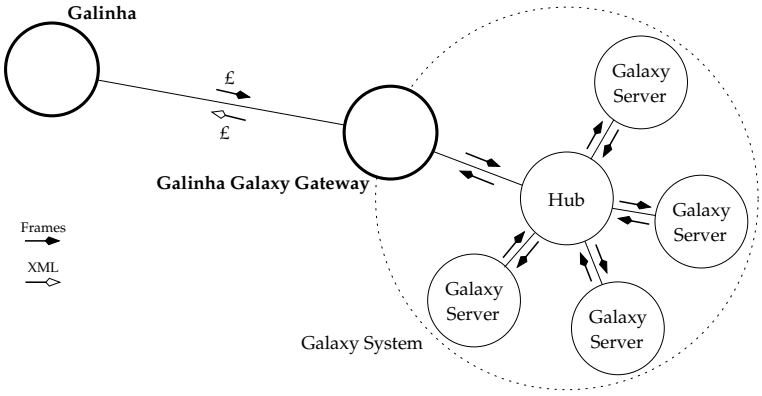

Fig. 3. G8 connection to the Galaxy infrastructure.

The Galaxy infrastructure remains stable and no changes were performed during the project, regarding its support.

2.2 Web services

Using Galaxy had the advantage of allowing a computing infrastructure already in our laboratory to be used for providing general support for invoking encapsulated services. On the down side, since Galaxy used a closed protocol, it was not simple to provide interoperability with other non-Galaxy environments. Our protocol was devised for abstract communication (§2.1), and even though it allowed simple integration with the upper Galinha layers, had the disadvantage of being a proprietary protocol. Proprietary protocols have to be adapted when needed in environments other than the ones they were intended to be used in. The adaptation process may be expensive, since it requires additional side work.

The use of web-based technologies enabled the Galinha infrastructure to be used in general environments, since web-based protocols, such as XML-RPC [15] or SOAP are either standards or recommendations for standardization, implemented in almost every relevant programming environment.

Web services were first integrated by using the approach used for Galaxy-based services. This provided a first test bench for the web platform. Support for web services was redefined in the G8 core and is also used in the G8 API for exporting services imported from other backends to client applications [9].

2.3 Hadoop platform

Hadoop [12] is a framework for running applications on large clusters built of commodity hardware. The Hadoop framework transparently provides applications both reliability, by automatically handling node failures, and data motion. Hadoop implements the map/reduce computing paradigm [6,7]. In addition, it provides a distributed file system that stores data on the compute nodes, providing very high aggregate bandwidth across the cluster.

Support for the Hadoop framework was included in our architecture to provide a platform for large clusters for data intensive operations. Hadoop was chosen due to its intrinsic support for node failures: since these failures are handled by the platform, applications do not have to increase in complexity to account for them. We provide a service-based interface to the Hadoop world as shown in figure 4. Although Hadoop provides large-scale distribution capabilities and built-in failure support, we also support other "traditional" infrastructures, such as the Condor distributed scheduler (§2.4).

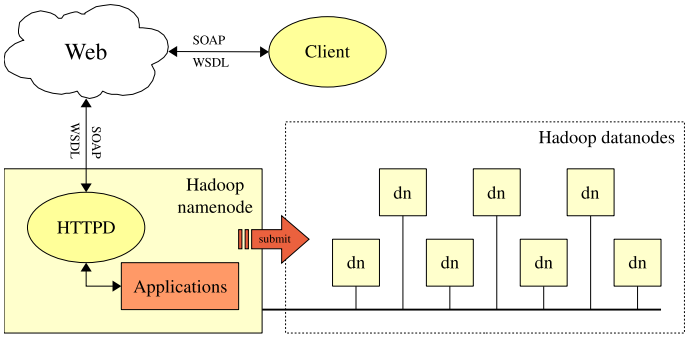

Fig. 4. Block diagram of the Hadoop service connector.

The deployment of Hadoop applications is currently not available through the interface: only previously deployed applications (as JAR files) can be invoked. The interface is supported through the architecture present in figure 4: deployed applications are invoked through web services that translate their semantics for use by prospective clients. Clients never need to know that the service is specifically supported by the Hadoop platform. This arrangement contributes to simplicity but limits other actions that could be performed over Hadoop (such as the previously mentioned deployment). Nevertheless, this limitation of the current version of the architecture can be easily handled and, since it is not a general user concern, should not negatively impact the overall architecture.

2.4 Condor

Although the Condor platform [11] is different in approach and deployment from Hadoop, the adaptation to the architecture is similar. Specifically, Condor-based application components are pre-loaded in the coordinator machine and a web service is used to invoke them.

A Condor-based component in the current architecture specification is limited to the Condor "vanilla" universe, although, in principle there should not be any limitation to other universes operating in the same way: the client is mostly independent from the differences. It is in the components themselves that differences lie (these differences are at the level of how the software needed by the deployed applications is handled). In our current test system, component implementations are mostly shell scripts running applications based on the tools described in the project's Task 2 [3,2].

Condor is a batch-oriented system, in which jobs are described by submission files. It is in these files that the inputs and outputs are specified, as well as the number of workers per job and the specific program to run. It should be noted that, as with Hadoop deployment, Condor submit files (and the programs themselves) are not controllable or modifiable by the code on the client side. Rather, services are built on top of the submission processes.

An advantage of using Condor as a backend, in addition to its relative simplicity, is its capability to interact with multiple universes, in particular, with parallel universes, such as MPI [1], and other grid environments, such as Globus [13].

2.5 Other backends

Other backends are either being tested for integration or being considered. The top candidate for integration in the short-term is the GridWay metascheduler [14]. The GridWay metascheduler (part of the Globus project) enables large-scale, reliable and efficient sharing of computing resources managed by different systems, such as Condor, within a single organization or scattered across several administrative domains (partner or supply-chain grid).

Interfacing with GridWay enables the architecture to abstract the various backend targets, further promoting its backend-independence.

3 Specific Conditions

This section deals with the problems of module encapsulation in specific contexts. This is a potential problem for service providers, but should not concern client applications.

We have been looking at modules as stand-alone tools working in a service providing context, such as a web service implementation. This is a simple and effective way of hiding complexity and make an interface widely available. Nevertheless, a service will only go so far in providing efficient access: if a tool is not already executing in a high performance environment, then a single access point becomes an execution bottleneck.

To address this problem, specific tools may have to be rewritten to account for different environments. From the point of view of the final user, and assuming that the interface does not change, there may not be any difference. For the service provider or implementer, the difference is much more in evidence, since the new enclosing environment will have to be accommodated in some way.

An example of this situation corresponds to the reimplementation of MARv's original C++ code in Java (crating jMARv), for using with the Hadoop platform. A Java implementation may be naturally included in a Hadoop program, while a C++ library or a stand-alone application will have to be adapted in ways that present other challenges (such as mechanisms depending on the operating system, as is the case of launching external applications and communicating through pipes) [3].

4 Final Remarks

The architecture as presented in this report is sufficiently general for supporting a wide range of backends and sufficiently modular that support for new backends, and drivers can be added without conflicting with existing infrastructure components, an important aspect when considering the stability offered to clients, both human and computational entities.

References

1. MPI Specification. http://www.mpi-forum.org/docs// (visited Januray 2008).
2. David Martins de Matos and Ricardo Ribeiro. NLE-GRID T2h: Lexicon Repository and Server (POSC/PLP/60663/2004). Technical Report 32, L^{2}F – Spoken Language Systems Laboratory, INESC ID Lisboa, Lisboa, Portugal, January 2008.
3. David Martins de Matos and Ricardo Ribeiro and Sérgio Paulo and Fernando Batista and Luísa Coheur and Joana Paulo Pardal. NLE-GRID T2: Encapsulation of Re-usable Components (POSC/PLP/60663/2004). Technical Report 31, L^{2}F – Spoken Language Systems Laboratory, INESC ID Lisboa, Lisboa, Portugal, January 2008.
4. David Martins de Matos and Nuno J. Mamede. The £ protocol. Technical Report 26, INESC-ID, September 2004.
5. David Martins de Matos, Joana L. Paulo, and Nuno J. Mamede. Managing Linguistic Resources and Tools. In N. J. Mamede, J. Baptista, I. Trancoso, and M. das Gracas Volpe Nunes, editors, *Computational Processing of the Portuguese Language – Proc. of the 6th Intl. Workshop, PROPOR 2003*, Lecture Notes in Artificial Inteligence, pages 135–142, Faro, Portugal, June 26–27 2003. Springer-Verlag, Heidelberg. LNAI 2721 – ISBN 3-540-40436-8.
6. Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Sixth Symposium on Operating System Design and Implementation (OSDI'04)*, San Francisco, California, December 2004.
7. Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
8. Jesse James Garrett. *Ajax: A New Approach to Web Applications*. Adaptive Path, LLC, San Francisco, California, February 2005. http://www.adaptivepath.com/publications/essays/archives/000385.php.
9. Luís Marujo, Wang Lin, David Martins de Matos. NLE-GRID T3: Multi-Component Application Builder (POSC/PLP/60663/2004). Technical Report 33, L^{2}F – Spoken Language Systems Laboratory, INESC ID Lisboa, Lisboa, Portugal, January 2008.
10. Massachussets Institute of Technology and MITRE Corporation. Galaxy Communicator. http://communicator.sf.net/ (visited Januray 2008), April 2001.
11. Condor Project. Condor – High Throughput Computing. http://www.cs.wisc.edu/condor/ (visited Januray 2008).

12. The Apache Software Foundation. Hadoop. http://lucene.apache.org/hadoop/ (visited Januray 2008).
13. The Globus Aliance. Globus Toolkit. http://www.globus.org/ (visited January 2008).
14. The Globus Aliance. GridWay Metascheduler – Metascheduling Technologies for the Grid. http://www.gridway.org/ (visited January 2008).
15. UserLand Software, Inc. XML-RPC. http://www.xmlrpc.com/ (visited Januray 2008).
16. W3C. Simple Object Access Protocol (SOAP) Version 1.2, April 2007. http://www.w3.org/TR/soap12 (visited Januray 2008).