Safe Resource Sharing in an Application Building Environment

David M. de Matos, Ângelo Reis, Marco Costa, Nuno J. Mamede

L^2F – Spoken Language Systems Laboratory – INESC ID Lisboa
Rua Alves Redol 9, 1000-029 Lisboa, Portugal
{david.matos,angelo.reis,marco.costa,nuno.mamede}@l2f.inesc-id.pt
IST – Instituto Superior Técnico

Abstract. Galinha is a system that allows simple albeit powerful access to NLP tools and resources. Due to the sometimes sensitive nature of those resources, access control in "unsafe" environments is crucial. This document presents work carried out in the context of the Galinha III project. We detail improvements regarding access control to backend tools and resources, as well as in application definition capabilities, and how these features allow us to deploy our resources in environments such as the Web and the classroom. A contrastive analysis, a summary of open points, and future directions close the paper.

Resumo. O sistema Galinha permite acesso simples mas eficaz a recursos e ferramentas linguísticas. Devido à natureza sensível de alguns recursos, o controlo de acesso em ambientes "inseguros" é crucial. Este documento apresenta o trabalho desenvolvido no contexto do projecto Galinha III. São detalhadas melhorias no que respeita ao controlo de acesso à infraestrutura e aos recursos, assim como na capacidade de definição de aplicações, e esta funcionalidade nos permite disponibilizar recursos em ambientes como a Web e em salas de aula. Uma análise da evolução, um resumo de pontos em aberto e direcções para desenvolvimentos futuros concluem o documento.

Keywords. Application building environments; Access control to linguistic resources and tools; Language engineering architectures.

1 Introduction and background

Galinha is a system that allows simple and powerful access to NLP tools and resources over the World Wide Web. The system presents a user-friendly interface and allows users to build distributed applications in a network-transparent way, using servers and modules and to effectively use those services [5, 6]. Thus, the user is free to ignore details that are irrelevant for the task at hand, e.g., someone interested in building an NLP application should not have to worry about infrastructure details, such as the whereabouts or particular environment required by a given service. Freedom of use, though, has the downside of having to control access to those resources, due to their sometimes sensitive nature: access control in "unsafe" environments is crucial.

This document presents work carried out in the context of the Galinha III project. Galinha III aims at providing access control capabilities to the NLP resources and tools made available through the Galinha system[1]: in particular, session-, traffic-, and user- and group-based control. The control aspects do not hamper the user, and are useful to protect the service provider. For instance, without control, a dictionary could be downloaded through repeated service execution; by imposing quotas, the problem may be dealt with, even to the extent of banning unruly users.

In addition to the control aspects and in contrast with previous versions, the system now allows applications to be built even when the support infrastructure is unavailable, i.e., offline use is possible. Moreover, users are now able to share application sequences among themselves without using any infrastructure support. Thus, if someone builds something, that piece of work is readily available to other users. Successful reuse will, it comes as no surprise, depend on the privileges of each user.

1.1 Motivations and objectives

Our previous Galinha environment already provided adequate support for application building, as demonstrated by our implementation of ATA, a semi-automatic terminology extraction system [10, 11]. ATA made use of morphological analyzers (SMorph [1], a tokenizer/tagger; PAsMO [9], a rule-based rewriter; and Marv [12], a morphological disambiguator); syntactic analyzers (SuSAna [2]); and other tools to carry out its tasks.

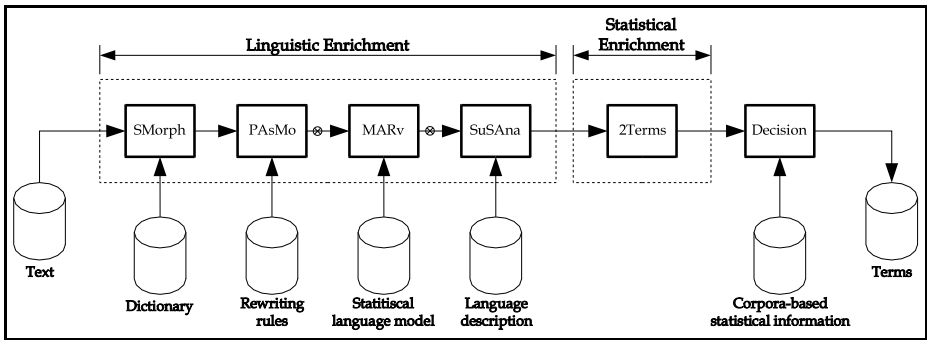

Fig. 1. The architecture off ATA [6].

Although the usefulness/adequacy aspect of the Galinha system was successfully demonstrated, the problem of controlling access to the critical resources and

[1] It should be noted that although Galinha is a general purpose environment, it was developed with NLP in mind, in particular, for easing the tasks associated with application creation and software access, reuse, and management.

tools remained. We sought a way to reconciliate these two conflicting aspects: allowing users to execute our code, using our resources; and keeping those users in line. On the other hand, we want to differentiate between users, e.g. a student with a language engineering assignment does not have, or rather, may not necessarily have, the same kind of access rights of a research partner in a remote facility.

The objectives of project Galinha III were, thus, to provide user profiles and access restrictions associated with various aspects: resource definition and deployment; execution time limits; data traffic limits; and system access frequency. Furthermore, in keeping with our commitment to provide simple access, all of these new aspects should also be controllable via dedicated interfaces available in and through the system.

2 Backend management

The presentation of Galinha as a user tool, i.e., how a user can use the system to build and run NLP applications, as well as a full description of the tools that were or may come to be connected to the system is outside the scope of this document. For a description, please see the architecture papers [5, 6], and, for examples, see the ATA papers [10, 11].

An area identified as problematic, during our experiences using the Galinha system and portal, was the abstract and deployment-independent definition of several concepts. The first we consider are NLP modules, i.e., sets of related operations, possibly handled by a single active computational agent; as an example, consider a morphological processing module such as SMorph: it provides several operations, all available from the same program. The second concept are systems, i.e., groups of jointly managed modules. Grouping decisions are somewhat arbitrary, but may take into account their relative locations and of the data resources they use. The last concepts to be considered were users and groups, i.e., the entities necessary for managing access to the other resources.

As of Galinha III, backend management consists of three modules: definition of systems; definition of modules and corresponding services; and deployment of systems and modules to actual physical machines running the actual programs. A system is, in principle, associated with a single machine: we say "in principle" because a system's definition may be cloned to run at another location – nevertheless, even if a clone, the new system is formally different. As for modules: they are only subject to deployment when associated with a system; they are otherwise completely independent of physical location and implementation means. This allows for a wide range of deployment options, which in turn make for simpler project decisions, when preparing use of a computational module in an NLP application built using Galinha.

Deployment of individual modules is not much different from running a server such as Apache [13] and is usually much simpler than the traditional approach of building a monolithic application with equivalent functionality, as proposed

by other systems, e.g. GATE[2] [4]. This fact stems from the separation of the problems raised by each module and the fact that no new problems arise from putting them all together, other than figuring out how to make data go from one to another. Moreover, a single module may remain in memory after completing each request and serve different applications, often simultaneously; in contrast, linked modules can only be used for a single purpose, that of the application they integrate.

Other systems and architectures for connecting modules exist, e.g. Galaxy Communicator [3], for building dialogue applications; or RAGS[3] [7], for building natural language generation systems. Galinha differs from them and builds upon their good aspects: namely, Galinha does not assume any particular function for any of the modules it includes, and the fact that modules are manageable in a transparent way in a variety of settings, helps to ensure simple deployment, even for non-expert users.

3 Access control

As mentioned above, the main motivation behind project Galinha III was to provide a means of controlling access to shared resources and data available through the Galinha system, thus safeguarding their correct use. Access control was designed to provide a good balance between granularity and usage, i.e., while very fine grained in some points, it remains easy to define and understand.

Access control is defined at the system-, module-, and service levels: e.g. use of a system; use of a module within the context of a given system; access to a module (i.e., controls whether the user/groups have access at all); access to a service. Besides these capabilities, there are user/group privileges regarding definition/modification/removal of system entities, as well as quotas on system access (frequency) and execution time.

The screen shots in figure 2 depict situations of system access editing. At the top, the main console for a system shows all modules and their services: all restrictions regarding system use (second part); use of specific modules in the system (third part); and use of individual operations[4] within the system (bottom part), are available from it. Restrictions are as follows:

- For systems: use, modify, add/remove modules, and delete;
- For modules within systems: use;
- For services within systems: execute, and examine;
- For individual modules: add to system, modify, delete, add/remove services to/from module;

[2] General Architecture for Text Engineering. Note that GATE represents a significative progress towards modularizing and reusing parts of applications. Not all problems are solved, though, especially those relating to actual deployment decisions.

[3] Reference Architecture for Generation Systems.

[4] Programs (see §4) are handled like services, except that, unlike services, they may be always examined.

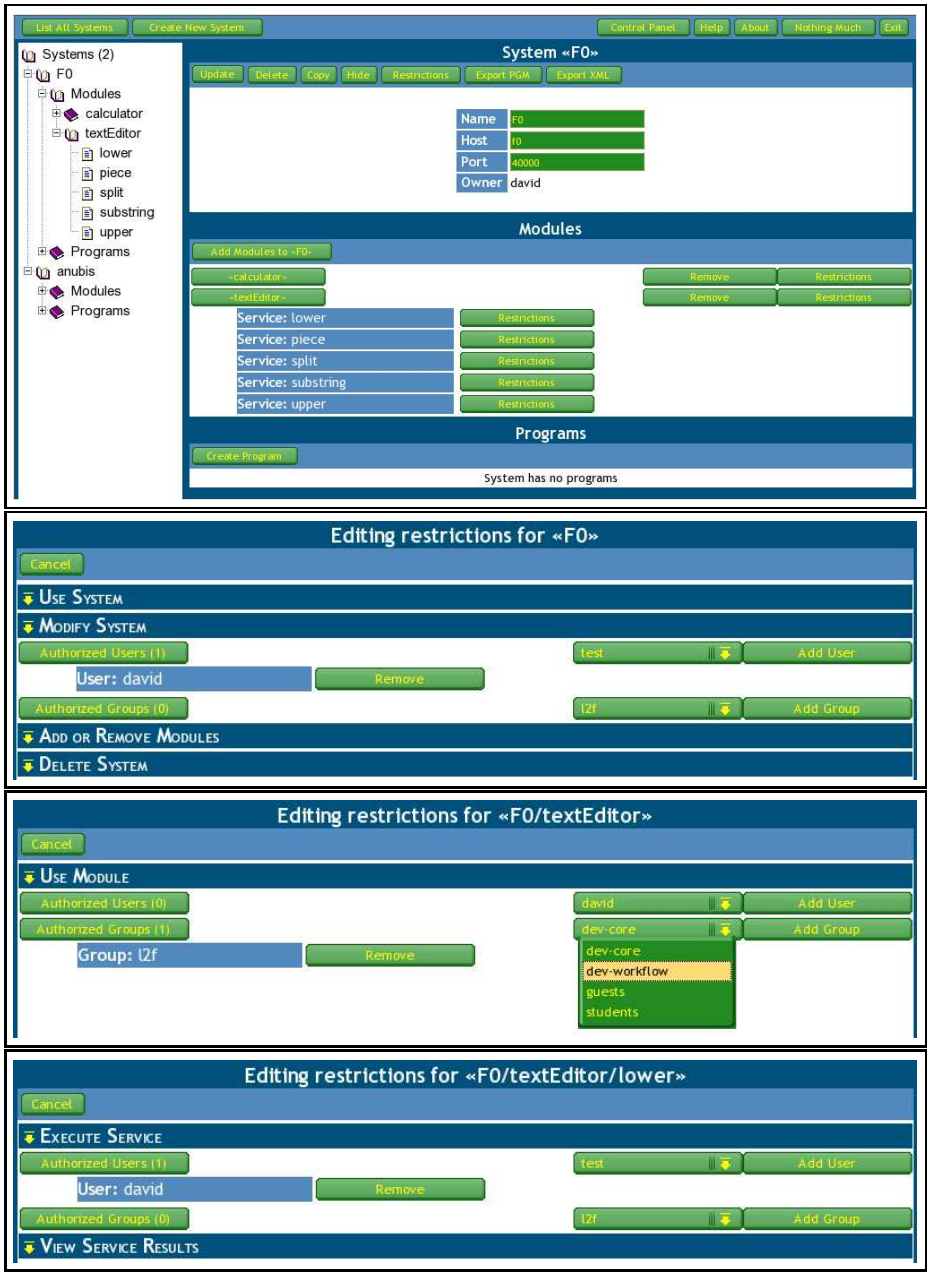

Fig. 2. System control capabilities: from top to bottom, main system control, showing available operations; system restrictions editor; use of specific modules within this system (independent from module access controls); fine access control at the level of individual operations.

Access control to modules and services, regardless of system, are defined in consoles similar to the one presented here for systems. Note that access to a given module or to a given service may be different, depending on the system including them.

4 Resource organization and sharing

Currently, modeled resources are organized in two categories: backend-side resources and user-side resources. On the backend side, resources are systems, modules, and services. If the backend supports it, the interface also provides the concept of program, i.e., special opaque backend-supported service sequences. Programs behave as a single service would and are not subject to internal inspection. Systems are collections of modules and programs and are associated with deployment features (such as hosts and ports) whose exact nature depends on the system's type[5]. If systems help deploy sets of modules, these are purely abstract entities describing sets of related services. Two examples are shown in the left part of figure 3, along with their services: `calculator` and `textEditor`.

On the user side, the only relevant entities are sequences of services, which we call chains (right frame of figure 3). These are simple entities that correspond to compositions of services. In contrast with (backend-side) programs, chains may be fully inspected and partial results may be analyses (cf. central frame of figure 3).

Regarding sharing, systems, modules, and services are shared by default and may be used as long as the user has sufficient privileges to do so. Chains, on the other hand, are only visible to users, other than their creators, if these share them explicitly. If a chain is shared, then the client user may choose to use it as is, or to copy it. In the latter case, modifications are possible. The main idea is that backend-defined resources belong to some entity that provides computing or data services and wants to make them available, but subject to certain conditions imposed by the restrictions on client access. Chains are another matter: they are made of backend parts and their sharing is to provide a degree of cooperation between users, i.e., if a user has built something that is deemed of interest, then the systems provides a way for it to be reused.

5 Final remarks and future directions

Preliminary results are very positive: for previous users of the Galinha system, the user console remains essentially unchanged, meaning that we where successful in seamlessly integrating the new functionality. In addition, new access modes are possible: profile-based access is a powerful way of providing various degrees of access restrictions to different user groups. It also means that the same infrastructure may be more flexibly used for widely differing objectives. As

[5] Galaxy systems are supported. Support for Web services, although working, is preliminary.

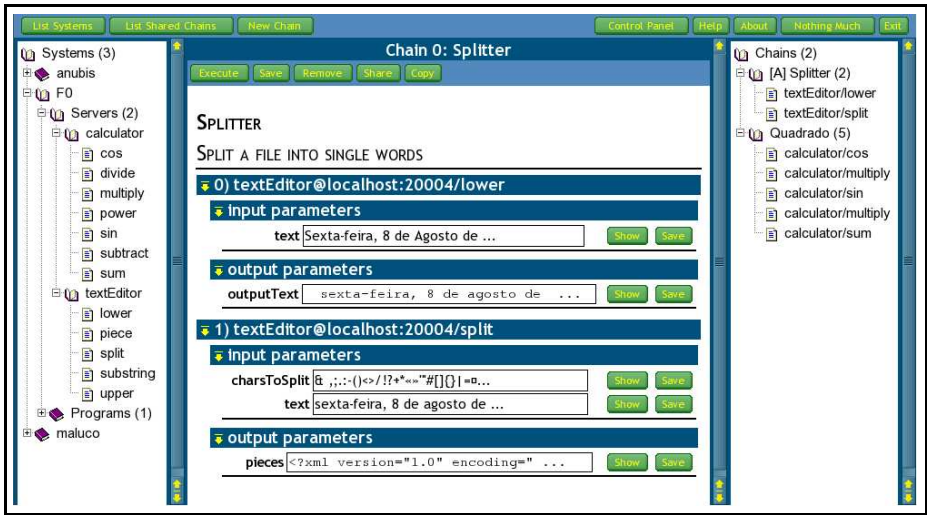

Fig. 3. The user console showing the results of a very simple application. This example takes an input text and splits it into words, according to a set of separator characters provided as argument.

examples, consider the following: using the same system, we are able to provide token access for demo purposes (on the Web); limited access to selected systems and modules to students of NLP courses; and full access (or however it may be defined) to researchers or other selected partners.

Among the future directions for this work, in what concerns the topics of this article, is the compatibility with other sources of user/group information that may be deemed useful: these include, in our case, our AFS users database [8, 15]; and, in the case of our students, information from IST's Fénix system [14] (an academic portal). This kind of interoperability is useful in helping to define usage permissions and in simplifying access to the system by delegating policy decisions on trusted third parties.

References

1. S. Aït-Mokhtar. *L'analyse présyntaxique en une seule étape*. Thèse de doctorat, Université Blaise Pascal, GRIL, Clermont-Ferrand, 1998.
2. Fernando Manuel Marques Batista. Análise sintáctica de superfície. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa, July 2003.
3. MIT/MITRE Corporation. Galaxy Communicator. `http://communicator.sf.net/`, April 2001.
4. Hamish Cunningham, Yorick Wilks, and Robert J. Gaizauskas. Gate – a general architecture for text engineering. In *Proceedings of the 16th Conference on Computational Linguistics (COLING96)*, Copenhagen, 1996.
5. David M. de Matos, Alexandre Mateus, João Graça, and Nuno J. Mamede. Empowering the User: a Data-Oriented Application-Building Framework. In *7th ERCIM*

Workshop "User Interfaces for All" Adjunct Proceedings, pages 37–44, Chantilly, France, October 2002. European Research Consortium for Informatics and Mathematics.

6. David M. 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*, number 2721 in Lecture Notes in Artificial Inteligence, pages 135–142, Faro, Portugal, June 26–27 2003. Springer-Verlag, Heidelberg. ISBN 3-540-40436-8.
7. ITRI. *RAGS – A Reference Architecture for Generation Systems*. Information Technology Research Institute, University of Brighton, nd. See: `http://www.itri.brighton.ac.uk/projects/rags/`.
8. OpenAFS. `http://www.openafs.org/`.
9. J. L. Paulo. PAsMo – Pós-Análise Morfológica. Technical report, L^2F – INESC-ID, Lisboa, 2001.
10. J. L. Paulo, M. Correia, N. J. Mamede, and C. Hagège. Using Morphological, Syntactical, and Statistical Information for Automatic Term Acquisition. In E. Ranchhod and N. Mamede, editors, *Advances in Natural Language Processing, 3rd Intl. Conf., Portugal for Natural Language Processing (PorTAL)*, pages 219–227, Faro, Portugal, 2002. Springer-Verlag, LNAI 2389.
11. Joana Lúcio Paulo, David Martins Matos, and Nuno J. Mamede. Terminology Mining with ATA and Galinha. In António Branco, Amália Mendes, and Ricardo Ribeiro, editors, *Language Technology: Linguistic Resources and Tools for Portuguese*. Edições Colibri, Lisboa, Portugal, (to appear).
12. Ricardo Daniel Santos Faro Marques Ribeiro. Anotação morfossintáctica desambiguada do português. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa, 2003.
13. The Apache Software Foundation. The Apache HTTP Server Project. `http://httpd.apache.org/`.
14. Instituto Superior Técnico. Fénix. `http://fenix.ist.utl.pt/`.
15. Carnagie Mellon University. AFS Reference Page. `http://www-2.cs.cmu.edu/afs/andrew.cmu.edu/usr/shadow/www/afs.html`.