Rosemary: A Flexible Programming Framework to Build Science Gateways

Shayan Shahand* and Sílvia D. Olabarriaga*
*Department of Clinical Epidemiology, Biostatistics, and Bioinformatics
Academic Medical Center of the University of Amsterdam, The Netherlands
Email: {s.shahand | s.d.olabarriaga}@amc.uva.nl

***Abstract*—The lessons learned during six years of experience in design, development, and operation of four Science Gateway (SG) generations motivated us to develop yet another generation of platforms coined "Rosemary". At the core of Rosemary the three fundamental SG functions, namely related to data, computing, and collaboration management, are integrated together. Our earlier studies showed that complete integration between these functions is a feature that is usually overlooked in the existing SG platforms. Rosemary provides a generic data model, RESTful API, and responsive UI that can be customized through programming to build customized SGs. Moreover, Rosemary is designed and implemented to be flexible to changes in e-Infrastructures and user community requirements. The software frameworks, tools and libraries employed in the realization of Rosemary streamline the development, deployment and operation of customized SGs for the users needs. So far the platform has been used to implement prototypes of three SGs for high-throughput analysis and management of neuroimaging data, sharing of data in in-vitro fertilization research, and provenance tracking of DNA sequencing data. This paper presents the design considerations, data model, and system architecture of Rosemary and highlights some of the features that are intrinsic to its design and implementation with examples from the three prototypes.**

***Keywords*—Science Gateway, Science Gateway Platform, Programming Framework, Data Management, Computing Management, Collaboration Management**

I. Introduction

Science Gateways (SGs) are web-based enterprise information systems that provide scientists with customized and easy access to community-specific data collections, computational tools, and collaborative services on e-Infrastructures [1], [2]. The construction of SGs is challenging because they include a large number of heterogeneous, distributed, and evolving software components and services. Moreover, there is a plethora of evolving and alternative technologies and platforms to choose from, as well illustrated by the rich information contained in the EGI Science Gateway Primer [3] and the XSEDE Science Gateway cookbook [4].

In this paper we present yet another SG framework, Rosemary, that was designed in response to lessons learned during 6 years of SG development and operation at the Academic Medical Center (AMC) of the University of Amsterdam. Rosemary addresses the need of flexible and light platform to address modern user interface requirements while integrating data, processing and collaboration functions as first-class concepts. The paper details the motivation and presents the requirements and system design of Rosemary in Sections II and III. The data model and system architecture of Rosemary are described in Sections IV and V. The current SGs implemented based on Rosemary are introduced in Section VI followed by a discussion in Section VIII and conclusion in Section IX.

II. Motivation

Between 2010 and 2015 we have designed, developed, and deployed four SG generations at the AMC. The first and second generations were prototypes and exploratory, which were evaluated with a small number of users in scientific projects and courses [5]. They consisted of simple web applications implemented in JSP that could start scripts to run an image analysis tool on the Dutch grid infrastructure. The third generation was based on a custom platform built on the Spring framework and the MOTEUR workflow management system [6]. It focused on providing computing power to biomedical scientists for neuroimaging, genomics and proteomics data analysis [7]. The fourth generation enriched SGs utility by integrating data management, in addition to facilitating access to the Dutch grid infrastructure. This generation was based on the WS-PGRADE/gUSE SG framework [8], and used to build customized SGs for neuroimaging data analysis [9] and molecular docking simulations [10]. Both the third and fourth generations were deployed and used daily in biomedical research projects, and helped researchers to handle large computations, for example refer to [11].

These SG generations were developed and studied in partnership with the computational neuroscience, bioinformatics, and medical chemistry research communities from AMC and Amsterdam region. The lessons learned in this process motivated the development of yet another generation of SG coined Rosemary, which are described in the following sections.

III. Requirements and System Design

To better introduce Rosemary SG platform, the user community requirements and the functions to address them are explained in Section III-A. Additionally, the considerations during the design phase are explained in Section III-B to support its design, implementation, and technology choices.

A. Three Pillars of Requirements and Functions

During the life cycle of a research project, several researchers collaborate with a wide spectrum of complementary background and expertise, such as domain-specific science, statistics, data processing, and information technology. These collaborators take various roles along the project life cycle, such as data collector, data analyst, support, and principal investigator. Moreover, researchers perform several tasks in different phases of the research life cycle based on their roles. Finally, on top of all that, some characteristics may vary in each discipline, depending on local culture and research project setup. All these characteristics translate to an overwhelming and complex set of requirements for a science gateway that can efficiently and effectively support research activities.

In spite of all this diversity, in our research we identified three foundational groups of functional requirements at the core of any SG, namely related to data, computing, and collaboration. These are extensively described in [12] and summarized below:

- Data-related requirements concern management of complex, distributed, and heterogeneous datasets on e-Infrastructures. Examples of data-related requirements include storing, annotating, searching, retrieving, replicating, and archiving datasets, as well as capturing metadata and provenance.
- Computing-related requirements concern management of complex, computationally demanding, distributed and coordinated data processing on e-Infrastructures. Examples of such requirements include negotiating resources, instantiating and setting up computer programs from application repositories, scheduling as well as managing the execution order and data flow between computer programs and handling failures.
- Collaboration-related requirements concern management of communications and interactions among scientists involved in a research project. Such collaborations entail, among others, definition of team membership; sharing or reusing data, computer programs, and resources; and exchanging information with reference to data and computing.

We experienced that it is easier to discover and see through the complex set of requirements of the research communities in the context of SGs when organizing them explicitly around data, computing, and collaboration requirements. For example, we started in the first generation motivated by the need for high-throughput computing. It was only when we deployed the third SG generation, which addressed the most prominent computing-related requirements, that the scientists and ourselves understood that there was a second dimension to consider, namely the data-related requirements. Likewise, when the data-related requirements were better addressed in the fourth generation, the need for more sophisticated collaboration mechanisms became evident, motivating for a fifth SG generation.

Furthermore, we found that all of the three functional requirement groups should be considered in the design. The SG should integrate data, computing, and collaboration resources seamlessly. Failing to integrate any of these resources will result in a SG that is not effective and will find limited use. A high-level illustration of SGs is depicted in Figure 1, where the SG functions related to these three functional requirement groups are shown by columns that are rooted on data, computation and collaboration. The rows represent the e-Infrastructure resources and the integration and enrichment layer implemented by the SG. Integration functions cross-cut and manage the complex relations and interplay among data, computing, and community management functions. These functions for example capture of data and process provenance, coordination of computation and data-related tasks, security, monitoring, notification, discussion among users, analytics of domain data and SG activity, and billing management. The detailed explanation of these functions and a comparison of some of the well-known SGs based on their provided functions can be found in [12].

It is also worth noting that SG design should be flexible enough to accommodate potential changes in the e-Infrastructure resources and in the three functional requirement groups due to expected evolution of research practices.

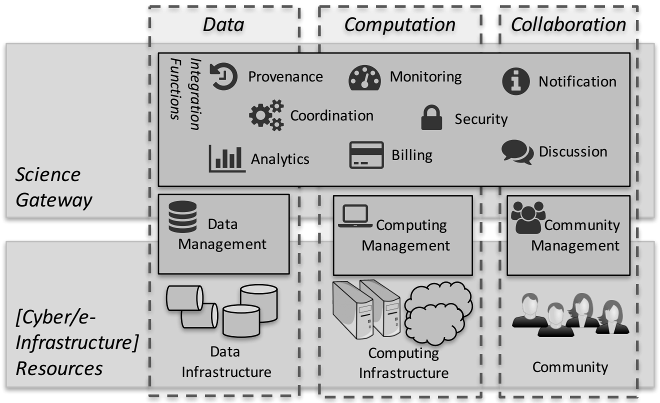


Figure 1. High-level illustration of SGs. Data, computation, and collaboration functions provided by e-Infrastructure resources are integrated together and enriched by SGs. Integrated functions cross-cut and manage the complex relations and interplay among data, computing, and community management functions. These functions can be further categorized into more specific groups as discussed in [12].

The design of Rosemary considers all the three functional requirement groups. It provides a metadata-rich abstract integration layer in which data, computing, and collaboration management functions are integrated together at its core. The rich metadata and integration level provided by Rosemary foster its capabilities to supply data provenance, increase automation, and unlock advanced features such as full search, user guidance, and data mash-up. Moreover, its implementation aims to ensure flexibility to accommodate changing requirements and e-Infrastructures.

B. Software Design Considerations

In general, there are two approaches to develop Science Gateways: *a)* to customize existing SGs or *b)* to develop from scratch, for example based on web application frameworks or Content Management Systems (CMS) [3], [13]. Customizing the existing SGs enables utilization of already provided SG functions such as integration and management of computing resources, which reduces development time. However, this approach also confines the developers to a certain software stack and SG functions, which implies understanding the architecture and details of the existing SG before being able to customize it or add new functions. For example, take the case of the Distributed Research Infrastructure for Hydro-Meteorology project (DRIHM) science gateway, where they report that building a SG based on a sophisticated modular and extensible SG framework was not smooth because of the amount of necessary modifications to address the user requirements, being confined in a certain software stack, and lack of integration between main entities [14]. In contrast to this approach, developing SGs from scratch gives the developers full control over the software stack and SG functions, but it increases the development time. However, even then it is possible to integrate with existing software components, for example from other SG frameworks, when these are decoupled and provide a clear abstraction layer. It is therefore important to choose the approach that suits best the requirements based on available resources and expertise.

Our third SG generation was developed from scratch based on the Spring framework [7], while in the fourth generation we customized the WS-PGRADE/gUSE SG framework [8] to develop our specific SGs [9], [10]. In the fifth generation, Rosemary, we opted for developing it from scratch based on the Play framework to be able to have full control over the implementation of the new SG data model (see Section IV) and its software stack. Note that we only implemented an "integration layer" from scratch that integrates data, computing, and collaboration functions seamlessly. Base on our previous analysis in [9], this level of integration is not offered by existing SG frameworks, as explained in Section VII.

In summary, the following items were the key features considered in the design and implementation of Rosemary:

- Fully integrate the three pillars of SG and provide functions for data, computing, and collaboration management.
- Maximize utilization of existing services and software components.
- Maximize flexibility for accommodating changes in e-Infrastructures and user community requirements.
- Minimize changes when customizing the SG for various disciplines.
- Streamline software development, deployment, and operation.
- Provide an easy-to-use and ubiquitous user interface (UI).

These features have various consequences on the design and implementation of Rosemary. The most prominent consequences are the effect on the data model and its implementation, utilization of a Service Oriented Architecture (SOA) and microservices, and use of modern software platforms and tools.

The new data model was essential to take into account all the three functional requirements and to integrate them all into the SG. When designing the new data model we also aimed at maximizing flexibility and generality so that it can cope with the changes and can be customized to address various requirements of different research disciplines. More details are presented in Section IV.

Although we opted for developing Rosemary SG framework from scratch, we didn't (still don't) intend to repeat ourselves and others [15]. It was our goal to maximize reusing code and software components developed for the previous generation of our SGs and by others. Therefore we used a SOA through microservices and wrapped and packaged (existing) software components into independent services. More details are presented in Section V.

In our previous SG generations a big portion of development time was spent on integration tests. Moreover, operation of those SGs turned out to be challenging because of the complexity of the system. We aimed at mitigating these challenges by utilizing modern web application frameworks (Play Framework for the back-end and AngularJS for the front-end) and compartmentalizing system components (using Docker and Ansible). Rapid development is facilitated by utilization of high-level programming languages (Scala, CoffeeScript) and the availability and richness of tools and libraries (Bootstrap, SBT, Gulp, Bower, Apache Lucene, Akka). These modern software platforms and libraries also enabled us to provide a Rich Internet Application (RIA) and responsive UI that can be used from various devices with various dimensions. More details are presented in Section V.

IV. Basic Concepts: Data Model

The basic concepts of the data model are explained through the three pillars (see Section III-A) and the integration layer between them. The simplified Entity Relationship (ER) diagram is depicted in Figure 2, where the most relevant attributes that are explained in this paper are highlighted. Note that all entities have a set of *<key, value, unit>* attribute attached to them to store metadata.

A. Data Management

Datum entity holds the domain data and metadata. Datum objects can form a hierarchical structure by declaring other objects as their children. One Datum object can be declared as child of multiple Datum objects. This relationship is used to capture the notion of association and derivatives, for example, multiple samples that belong to a subject or a 3D model that is reconstructed from multiple brain scans.

B. Computing Management

Recipe entity captures what can be done with data and is meant to store data processing functions. Examples of Recipe objects are program (or workflow) descriptions or simply a document that describes how one sort of Datum can be derived from another one. Recipe objects may also have a set of input abstract ports that describe the inputs expected from users. A Recipe object may have a program description attached to it, and in that case, it also includes *a)* the abstract input and output ports as expected and generated by that program; and *b)* a transformer that specifies how user inputs could be validated and transformed into program inputs, and how program outputs should be interpreted and transformed by the system to Datum objects.

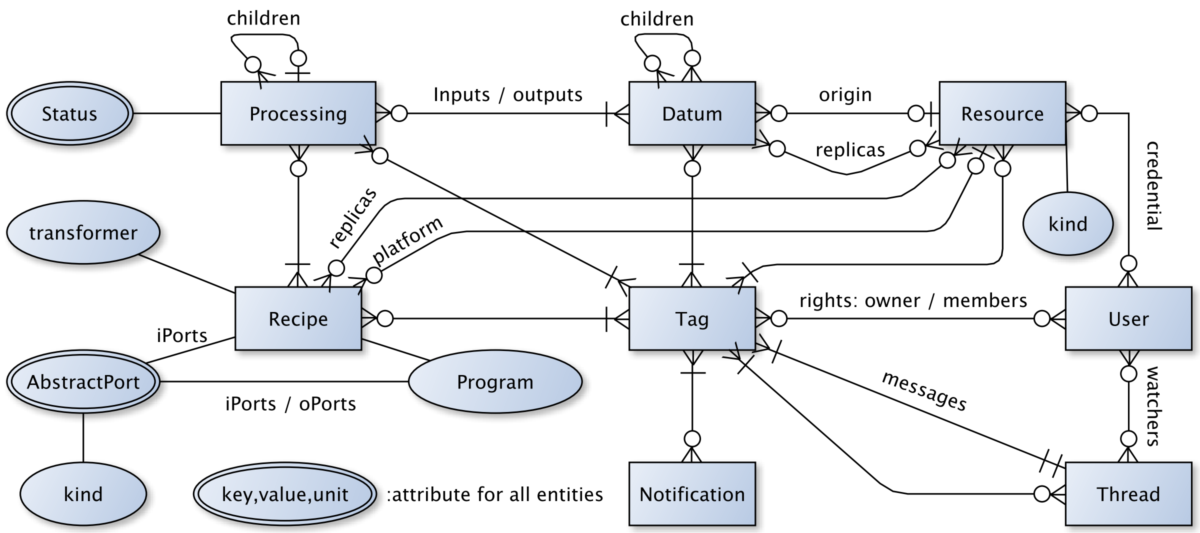


Figure 2. Simplified ER diagram with highlighted attributes. All entities have a set of <*key, value, unit*> attribute to store metadata. Tag entity is located at the heart of the data model and is one of the key factors that make the data model flexible and customizable.

Processing entity captures the actual application of Recipe(s) on data. A Processing object has one or multiple Datum as inputs and outputs, a set of status, and one or multiple Recipes to specify which data processing functions have been used. Examples of Processing objects are program (or workflow) executions or a user action such as manual file edit. Processing objects can form a hierarchical structure to group them together and capture multiple levels of abstraction. For example, the Datum selected by the user in the SG as the input of one Processing can be (automatically) transformed based on the domain information into multiple files that will be used in multiple processings organized into a group.

C. Collaboration Management

User entity captures user information such as their name, email address, and preferences.

Thread entity stores an array of messages (see Tags in Section IV-D) to capture the communication between users around a specific topic involving SG activity. For example, users can discuss the outputs of a certain data processing or receive a domain or system expert opinion for troubleshooting a data processing.

Notification entity holds the information about a system event or a user action, for example, when a Processing has been completed, or a message has arrived on a Thread.

D. Integration of the Three Pillars

Integration between the three pillars is managed by two entities, Tag and Resource, and the relationships between these entities and the rest of the entities.

Tag entity is located at the heart of the data model and is one of the key factors that make the data model flexible and customizable. Tags are used to annotate objects to create groups and deliver functions such as filtering, access control, and communication. Access to Tags is controlled by defining their owner and members. Examples of the usage of Tags include:

- Create *workspaces* in which users collaborate with each other. The access to Datum, Recipe, Processing, Resource, Thread, and Notification objects is controlled through the access control of the workspace tags. This means that when an object is annotated with a workspace Tag it would become accessible to anybody that has access to that Tag. Workspace-specific information, such as community credentials for resources, can also be stored in these Tags.
- Annotate Datum or Processing objects with a category name that can be used for filtering during search. Tag labels are used to customize the SGs for specific domains.
- Group together or filter some Datum or Processing objects based on user-defined Tags.
- Annotate Datum and Processing objects with messages and form a discussion around them. A message in a communication Thread is indeed a Tag.

Resource entity stores the information about various data resources or computing platforms. Datum entities are originated from, and have replicas on, data Resources. Recipes can also have replicas on data Resources, i.e., to specify the location of program binaries or Recipe descriptions. Additionally, Recipes can be eligible to be executed on a certain computing platform. Regarding credentials to access Resources, users could also have their own personal credential in addition to community credentials.

V. System Architecture and Implementation

System Architecture of Rosemary is depicted in Figure 3. It consists of three layers: front-end, back-end and the resource layer provided by the e-Infrastructures.

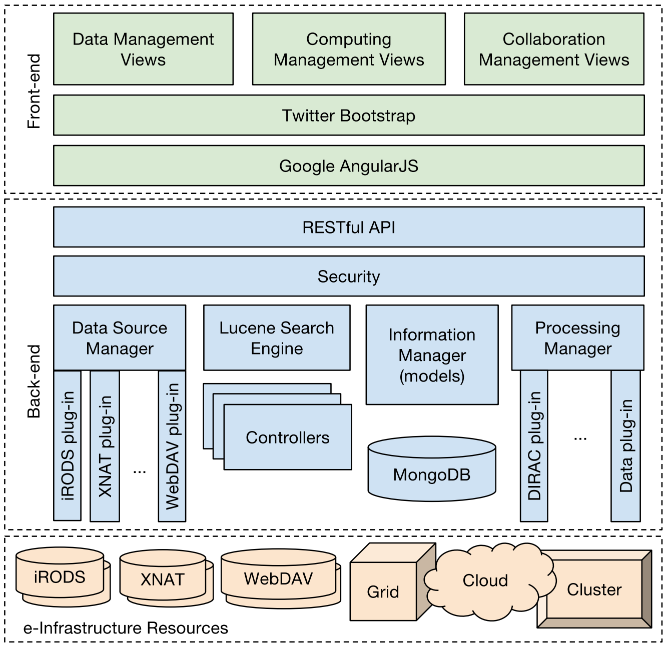


Figure 3. Rosemary system architecture showing Front-end, Back-end, and e-Infrastructure layers The back-end provides a secure RESTful API to integrate, enrich, and manage data, computing, and collaboration functions provided by e-Infrastructure resources.

The front-end uses the back-end RESTful API to provide data, computing, and collaboration management functions to users. The UI is made responsive to the screen dimension through the Twitter Bootstrap[1] front-end framework. The front-end has been implemented based on the Google AngularJS[2] web application framework to deliver a single-page RIA. The front-end controllers are implemented using the CoffeeScript[3] programming language that enhances JavaScript's readability and conciseness. All these frameworks and high-level programming languages improved the development and test cycles and therefore enhanced front-end stability.

The back-end provides a secure RESTful API to integrate, enrich, and manage data, computation and collaboration functions provided by the e-Infrastructure resources. It is implemented based on the Play web application framework[4] and coded in the Scala[5] functional and object-oriented programming language. The SG information is stored in MongoDB[6], a document-oriented database that offers the required flexibility and scalability. The information about Datum and Processing objects, including their metadata and annotating Tags, is indexed and searched upon using Apache Lucene[7] search engine. The back-end also includes a *Data Source Manager (DSM)* that provides an abstraction layer over management of data and metadata that is stored on various data sources such as iRODS[8], XNAT[9], and WebDAV[10]. The DSM utilizes the API of the e-Infrastructure data sources to provide the functionality to import data and metadata into the SG. Similarly, it provides export and replication functions to those data sources. Another important component of the back-end is the *Processing Manager (PM)* [16], which provides a RESTful API and an abstraction layer over the computation management functions on computing platforms. The PM potentially handles simple jobs and workflows depending on the underlying middleware or service used through its plug-ins. Currently, the PM uses Distributed Infrastructure with Remote Agent Control (DIRAC [17]) Interware to interface with various computing platforms such as Grid, Cloud, and Cluster. DIARC utilizes the API of the e-Infrastructure computing platforms to run, manage, and monitor computations on them. The PM also stages input and output data and application to/from the computing platforms using a simple data plug-in. Compared to the version described in [16], the PM included in Rosemary has been reimplemented in Scala based on the Akka toolkit[11] and RabbitMQ[12] message broker to enhance its scalability and transparency for troubleshooting.

The development of Rosemary is streamlined by the ecosystem of tools including SBT[13], Gulp[14], and Bower[15] to manage library dependencies and automate system builds. The Play framework in particular makes it easy to hot redeploy the SG and test its functionality during development. Deployment is streamlined by encapsulating various parts of the system, i.e., MongoDB, Processing Manager, Back-end and Front-end, in separate Docker[16] containers and configuring those containers using Ansible[17] configuration management platform.

VI. Current Implementation of Gateways

The common functions provided by Rosemary SG platform include:

- Create and manage workspaces in which users collaborate in data analysis tasks. Workspaces bring a collection of data, recipes (applications), processings, communication threads, notifications, users, and community resources and their credentials together.
- Authenticate to various external data and computing resources using the available community or user credentials.
- Import data and metadata from external data sources into a workspace so that it can be managed or processed through the SG.
- Configure available applications and submit and manage data processings to computing platforms. Necessary steps

[1]http://getbootstrap.com/
[2]https://www.angularjs.org/
[3]http://coffeescript.org/
[4]https://www.playframework.com/
[5]http://www.scala-lang.org/
[6]https://www.mongodb.com/
[7]https://lucene.apache.org/
[8]https://irods.org/
[9]http://www.xnat.org/
[10]https://en.wikipedia.org/wiki/WebDAV
[11]http://akka.io/
[12]https://www.rabbitmq.com/
[13]http://www.scala-sbt.org/
[14]http://gulpjs.com/
[15]http://bower.io/
[16]http://www.docker.com/
[17]https://www.ansible.com/

for data processing, such as application and data staging and cleanup, are performed automatically by the SG.
- Search and filter data and processings based on metadata and annotated tags such as category or user-defined tags.
- Communicate with other users with reference to data and processings.
- Receive notifications of system events and user actions.
- Investigate provenance information to establish trust and troubleshoot problems.
- Export data and metadata to an external data source, for example for data preservation or archival.

So far, three community-specific SG prototypes have been implemented based on the Rosemary platform:

- The Neuroscience gateway provides complex and multi-site neuroscience data management, computing management on Grid computing resources, and collaboration management among scientists. See example of UI on Figure 4.
- The In Vitro Fertilization (IVF) gateway provides complex multi-site IVF data sharing management, specifically to follow a workflow that involves authorization from multiple centers for each requested dataset.
- The Genomics gateway provides wet-lab and dry-lab sequencing data management with detailed provenance tracking, as well as collaboration management among scientists.

VII. Related work

A detailed analysis of science gateway software frameworks has been presented in our recent work [9]. From an analysis of literature published after 2011, 11 frameworks have been selected for analysis: Apache Airavata [18], Catania SG framework [19], Distributed Application Runtime Environment (DARE) [20], Globus [Online] [21], HUBzero + Pegasus [22], ICAT Job Portal [23], InSilicoLab [24], iPlant [25], NEWT Platform [26], SINAPAD SG [13], and WS-PGRADE/gUSE [8]. Note that this is a subset of the large number of existing tools and systems that potentially could be used in software ecosystem of a SG. From that analysis, we concluded that most of these frameworks do not include software for all the proposed functional groups, such as we attempted to provide with the development of the new Rosemary SG platform. See more details in [9].

Here we build upon this previous work to compare Rosemary to existing systems. Only the information available in the scientific publications was considered, therefore some degree of inaccuracy is possible in the summary presented below.

Delivery functions for humans and programs are implemented in Rosemary as the responsive front-end and the RESTful API provided by the back-end. These are also found in all of the frameworks, with the exception of Apache Airavata and NEWT, which do not implement user interfaces.

Coordination functions are also present in all of the frameworks, with the exception of ICAT and NEWT. Some frameworks integrate advanced workflow management systems such as Pegasus [27] and WS-PGRADE/gUSE [8], while others are based on pilot job abstractions (e.g., DARE and inSilicoLab). In Rosemary coordination functions are realized through the Processing Manager, which includes data and DIRAC plug-ins for data staging, program instantiation, pilot job abstractions, and simple workflows.

Security functions are implemented by all of the frameworks at various levels of sophistication. For example, the Catania SG and iPlant use Shibboleth[18] for single sign-on, whereas DARE uses standard security mechanisms from SAGA[19]. In Rosemary security is still primitive at the moment, being implemented through internal credential management components and token-based authentication.

Monitoring functions are also present in all of the frameworks to some extent. The Catania Portal and NEWT platform, for example, provide system, resource and job monitoring and accounting functions. Rosemary implements monitoring functions not only for computation, but also for data transfers and collaboration, which are all delivered to the users through a notification system.

Provenance functions are present in only a few frameworks. The best example is Pegasus, which keeps track of executed software, data locations and execution parameters. In Rosemary special attention has been given to provenance functions. PM in particular captures and stores the provenance information about data processing in the PROV standard[20]. Additional information about the sources of data and their replicas is captured in the Rosemary Data Model.

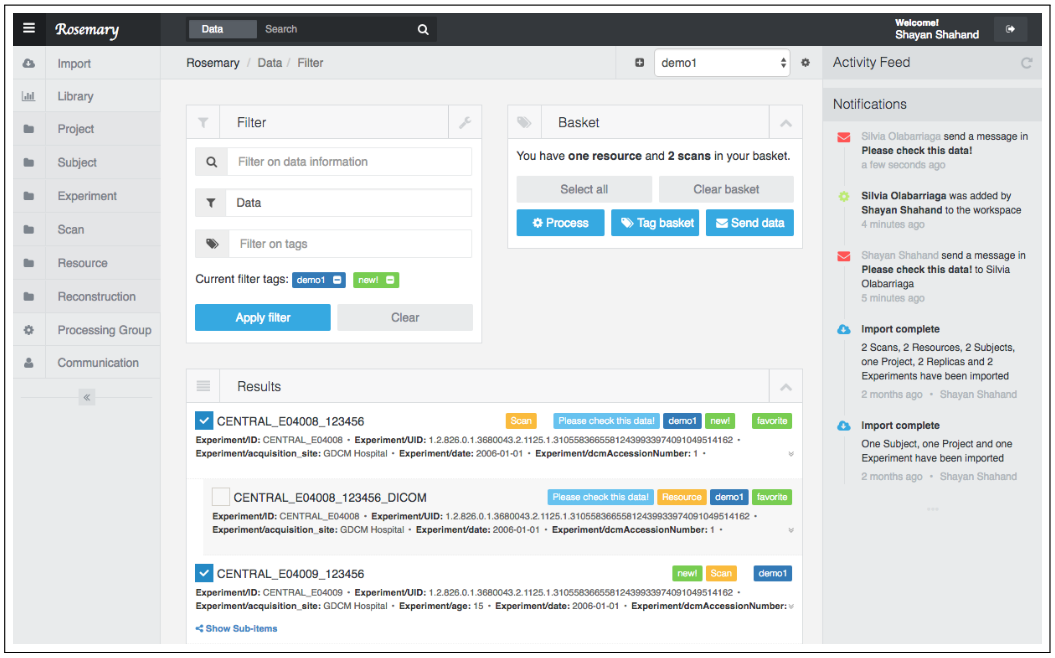
Functions for Data and Computing management are found in all the studied frameworks, with the exception of Globus Online (no computation) and DARE (no data). Typically the frameworks exploit existing middleware and standardized abstractions, such as SAGA (e.g., Catania SG, DARE) and BES (e.g., WS-PGRADE/gUSE) for data and computing management. In Rosemary, the Processing Manager and the Data Source Manager play an important role as abstraction layer to the resources, where plug-ins can be developed to interface with nearly any required resource. It however does not adopt any standard.

Finally, Community management functions are present in most frameworks, normally in simple form such as group and role management. We believe that this is the aspect that most differentiates Rosemary from other frameworks, since it has integrated the community management functions with the data and processing management functions at its core, which is demonstrated by features such as workspaces and communication threads.

VIII. Discussion

Rosemary is a SG framework that needs to be customized for a particular user community by programming and possibly by adding new functions to it. Therefore it requires understanding different parts of the system, as well as being able to

[18]http://shibboleth.net/
[19]http://saga-project.org
[20]https://www.w3.org/TR/prov-overview/



Filter on data information

Data

Filter on tags

Current filter tags: demo1 new!

You have **one resource** and **2 scans** in your basket.

CENTRAL_E04008_123456 — Scan · Please check this data! · demo1 · new! · favorite

Experiment/ID: CENTRAL_E04008 • **Experiment/UID:** 1.2.826.0.1.3680043.2.1125.1.31055836655812439933974091049514162 • **Experiment/acquisition_site:** GDCM Hospital • **Experiment/date:** 2006-01-01 • **Experiment/dcmAccessionNumber:** 1 •

CENTRAL_E04008_123456_DICOM — Please check this data! · Resource · demo1 · favorite

Experiment/ID: CENTRAL_E04008 • **Experiment/UID:** 1.2.826.0.1.3680043.2.1125.1.31055836655812439933974091049514162 • **Experiment/acquisition_site:** GDCM Hospital • **Experiment/date:** 2006-01-01 • **Experiment/dcmAccessionNumber:** 1 •

CENTRAL_E04009_123456 — new! · Scan · demo1

Experiment/ID: CENTRAL_E04009 • **Experiment/UID:** 1.2.826.0.1.3680043.2.1125.1.31055836655812439933974091049514162 • **Experiment/acquisition_site:** GDCM Hospital • **Experiment/age:** 15 • **Experiment/date:** 2006-01-01 • **Experiment/dcmAccessionNumber:**

Silvia Olabarriaga send a message in **Please check this data!**
a few seconds ago

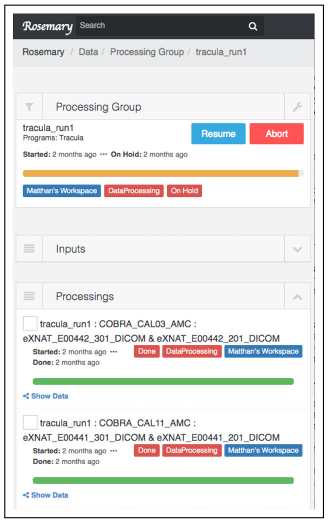
Silvia Olabarriaga was added by **Shayan Shahand** to the workspace
4 minutes ago

Shayan Shahand send a message in **Please check this data!** to Silvia Olabarriaga
5 minutes ago

Import complete
2 Scans, 2 Resources, 2 Subjects, one Project, 2 Replicas and 2 Experiments have been imported
2 months ago • Shayan Shahand

Import complete
One Subject, one Project and one Experiment have been imported
2 months ago • Shayan Shahand

(a)



tracula_run1
Programs: Tracula
Started: 2 months ago ··· **On Hold:** 2 months ago
Matthan's Workspace · DataProcessing · On Hold

tracula_run1 : COBRA_CAL03_AMC : eXNAT_E00442_301_DICOM & eXNAT_E00442_201_DICOM
Started: 2 months ago ··· **Done:** 2 months ago
Done · DataProcessing · Matthan's Workspace

tracula_run1 : COBRA_CAL11_AMC : eXNAT_E00441_301_DICOM & eXNAT_E00441_201_DICOM
Started: 2 months ago ··· **Done:** 2 months ago
Done · DataProcessing · Matthan's Workspace

(b)

Figure 4. Screenshot of the Neuroscience gateway prototype. *(a)* Shows a desktop GUI that contains notifications (right), filters for data types (left), working area (center) with search interface and basket with selected items. *(b)* shows a mobile GUI with a processing group information (top) and its corresponding processings (bottom).

program in Scala or Java. Our experience with the three SG implementations based on Rosemary showed that the generic data model and platform implementation, the high-level programming languages, and the employed software development ecosystem facilitate adding or customizing functions quickly. The Neuroscience gateway was the initial use-case by which Rosemary platform was implemented. Based on that, the IVF gateway was implemented in one month by a programmer that was not familiar with the platform. Finally, the Genomics gateway was implemented in one month by two programmers that were familiar with the platform. However, with this approach it is also necessary to re-factor the generic parts of new functions and make them available for existing and future gateway instances. To streamline such a code re-factoring process it seems necessary to implement more SGs and reevaluate and re-organize the code based on that experience.

The three SG prototypes have attracted significant attention so far, mainly because of their flexibility, lightness, and being able to customize them quickly for various disciplines. In addition, based on the initial feedback, its responsive UI, richness of metadata management, and search functionality are the most attractive Rosemary features. Although the initial feedback and enthusiasm from users are promising, usage by a larger group is required to understand shortcomings of the system and further improve it.

IX. Conclusion

Rosemary is a SG framework that integrates data, computing, and collaboration management functions. It provides a generic and metadata-rich data model and a software framework that can be customized by programming for rapid implementation of new SGs that fulfill user requirements and provide rich user experience. It also utilizes high-level programming languages, modern software framework and libraries, and a software ecosystem that supports continuous testing and rapid deployment, all of which streamline software development, deployment, and operation. Rosemary framework will be published as an open source project to increase its uptake and invite other developers to improve it by their contributions.

Acknowledgment

We thank Jorrit Posthuma, Allard van Altena, Mohammad Mahdi Jaghoori, Juan Luis Font-Calvo, Sara Ramezani, Jalmar Teeuw, and Matthan Caan for their various contributions to the Rosemary platform. We are grateful to our collaborators at various departments of the AMC (KEBB; Radiology; Immunology; Medical Chemistry; ADICT) and the University of Amsterdam (ICTS and FNWI), as well as and our former colleagues at the AMC e-Science group for various contributions, inspiration and support. We also thank the anonymous reviewers for their feedback and suggestions. This work was carried out on the Dutch national e-Infrastructure with the support of SURF Cooperative. This publication was supported by the Dutch national program COMMIT/ and the High Performance Computing and Networking (HPCN) Fund of the University of Amsterdam.

References

[1] A. Carusi and T. Reimer, “Virtual research environment collaborative landscape study-A JISC funded project”, Joint Infrastructure Systems Committee, Tech. Rep., 2010.

[2] N. Wilkins-Diehr, “Special Issue: Science Gateways—Common Community Interfaces to Grid Resources”, *Concurrency and Computation: Practice and Experience*, vol. 19, no. 6, pp. 743–749, 2007.

[3] European Grid Infrastructure (EGI) Science Gateway Virtual Team, *Science Gateway Primer*, https://documents.egi.eu/document/1463, 2012.

[4] *XSEDE Gateways Cookbook*, Online: https://www.xsede.org/web/gateways/gateways-cookbook.

[5] M. W. A. Caan, S. Shahand, F. M. Vos, A. H. C. van Kampen, and S. D. Olabarriaga, “Evolution of grid-based services for Diffusion Tensor Image analysis”, *Future Generation Computer Systems*, vol. 28, no. 8, pp. 1194 –1204, 2012.

[6] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec, “Flexible and Efficient Workflow Deployment of Data-Intensive Applications On Grids With MOTEUR”, *International Journal of High Performance Computing Applications*, vol. 22, no. 3, pp. 347–360, Aug. 2008.

[7] S. Shahand, M. Santcroos, A. Kampen, and S. Olabarriaga, “A Grid-Enabled Gateway for Biomedical Data Analysis”, *Journal of Grid Computing*, vol. 10, no. 4, pp. 725–742, 2012.

[8] P. Kacsuk, Z. Farkas, M. Kozlovszky, G. Hermann, A. Balasko, K. Karoczkai, and I. Marton, “WS-PGRADE/gUSE Generic DCI Gateway Framework for a Large Variety of User Communities”, *Journal of Grid Computing*, vol. 10, no. 4, pp. 601–630, 2012.

[9] S. Shahand, A. Benabdelkader, M. M. Jaghoori, M. a. Mourabit, J. Huguet, M. W. Caan, A. H. van Kampen, and S. D. Olabarriaga, “A data-centric neuroscience gateway: design, implementation, and experiences”, *Concurrency and Computation: Practice and Experience*, vol. 27, no. 2, pp. 489–506, 2015.

[10] M. M. Jaghoori, A. J. V. Altena, B. Bleijlevens, and S. D. Olabarriaga, “A Grid-Enabled Virtual Screening Gateway”, in *Science Gateways (IWSG), 2014 6th International Workshop on*, IEEE, Dublin, Ireland, 2014, pp. 24–29.

[11] G. A. van Wingen, E. Geuze, M. W. A. Caan, T. Kozicz, S. D. Olabarriaga, D. Denys, E. Vermetten, and G. Fernández, “Persistent and reversible consequences of combat stress on the mesofrontal circuit and cognition”, *Proceedings of the National Academy of Sciences*, vol. 109, no. 38, pp. 15 508–15 513, 2012.

[12] S. Shahand, A. H. van Kampen, and S. D. Olabarriaga, “Science Gateway Canvas: A Business Reference Model for Science Gateways”, in *Proceedings of the 1st Workshop on The Science of Cyberinfrastructure: Research, Experience, Applications and Models*, ser. SCREAM ’15, New York, NY, USA: ACM, 2015, pp. 45–52.

[13] A. T. A. Gomes, B. F. Bastos, V. Medeiros, and V. M. Moreira, “Experiences of the Brazilian national high-performance computing network on the rapid prototyping of science gateways”, *Concurrency and Computation: Practice and Experience*, vol. 27, no. 2, pp. 271–289, 2015.

[14] D. D’Agostino, E. Danovaro, A. Clematis, L. Roverelli, G. Zereik, A. Parodi, and A. Galizia, “Lessons learned implementing a science gateway for hydro-meteorological research”, *Concurrency and Computation: Practice and Experience*, n/a–n/a, 2015.

[15] G. Wilson, D. A. Aruliah, C. T. Brown, N. P. C. Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. Huff, I. M. Mitchell, M. Plumbley, B. Waugh, E. P. White, and P. Wilson, “Best Practices for Scientific Computing”, *In Press*, Oct. 2012.

[16] M. M. Jaghoori, S. Shahand, and S. D. Olabarriaga, “Processing Manager for Science Gateways”, in *Science Gateways (IWSG), 2015 7th International Workshop on*, Jun. 2015, pp. 1–7.

[17] V. M. Muñoz, A. C. Ramo, R. G. Diaz, and A. Tsaregorodtsev, “Powering Distributed Applications with DIRAC Engine”, in *The International Symposium on Grids and Clouds (ISGC)*, vol. 2014, 2014.

[18] M. Pierce, S. Marru, L. Gunathilake, T. Kanewala, R. Singh, S. Wijeratne, C. Wimalasena, C. Herath, E. Chinthaka, C. Mattmann, A. Slominski, and P. Tangchaisin, “Apache Airavata: Design and Directions of a Science Gateway Framework”, in *6th International Workshop on Science Gateways*, ser. IWSG 2014, Jun. 2014, pp. 48–54.

[19] V. Ardizzone, R. Barbera, A. Calanducci, M. Fargetta, E. Ingrà, I. Porro, G. La Rocca, S. Monforte, R. Ricceri, R. Rotondo, D. Scardaci, and A. Schenone, “The DECIDE Science Gateway”, *Journal of Grid Computing*, vol. 10, no. 4, pp. 689–707, 2012.

[20] S. Maddineni, J. Kim, Y. El-Khamra, and S. Jha, “Distributed Application Runtime Environment (DARE): A Standards-based Middleware Framework for Science-Gateways”, *Journal of Grid Computing*, vol. 10, no. 4, pp. 647–664, 2012.

[21] R. Ananthakrishnan, K. Chard, I. Foster, and S. Tuecke, “Globus platform-as-a-service for collaborative science applications”, *Concurrency and Computation: Practice and Experience*, vol. 27, no. 2, pp. 290–305, 2015.

[22] M. McLennan, S. Clark, E. Deelman, M. Rynge, K. Vahi, F. McKenna, D. Kearney, and C. Song, “HUBzero and Pegasus: integrating scientific workflows into science gateways”, *Concurrency and Computation: Practice and Experience*, vol. 27, no. 2, pp. 328–343, 2015.

[23] S. M. Fisher, K. Phipps, and D. J. Rolfe, “ICAT Job Portal: a generic job submission system built on a scientific data catalog”, in *Proceedings of 5th International Workshop on Science Gateways for Life Sciences*, ser. IWSG 2013, 2013.

[24] J. Kocot, T. Szepieniec, P. Wo’jcik, M. Trzeciak, M. Golik, T. Grabarczyk, H. Siejkowski, and M. Sterzel, “A Framework for Domain-Specific Science Gateways”, in *EScience on Distributed Computing Infrastructure. Achievements of PLGrid Plus Domain-Specific Services and Tools: LNCS 8500.* Switzerland: Springer, 2014, pp. 130–46.

[25] A. Lenards, N. Merchant, and D. Stanzione, “Building an Environment to Facilitate Discoveries for Plant Sciences”, in *Proceedings of the 2011 ACM Workshop on Gateway Computing Environments*, ser. GCE ’11, New York, NY, USA: ACM, 2011, pp. 51–58.

[26] S. Cholia and T. Sun, “The NEWT Platform: An Extensible Plugin Framework for Creating ReSTful HPC APIs”, in *Proceedings of the 9th Gateway Computing Environments Workshop*, ser. GCE ’14, Piscataway, NJ, USA: IEEE Press, 2014, pp. 17–20.

[27] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. da Silva, M. Livny, *et al.*, “Pegasus, a workflow management system for science automation”, *Future Generation Computer Systems*, 2014.