SEVENTH FRAMEWORK PROGRAMME

CAPACITIES

Research Infrastructures

INFRA-2007-1.2.1 Research Infrastructures

DRIVER II

Grant Agreement 212147

“Digital Repository Infrastructure Vision for European Research II”

Enabling Services Enhancement Specification

Deliverable Code: D7.1

Document Description

Project	
Title:	DRIVER, Digital Repository Infrastructure Vision for European Research II
Start date:	1st December 2007
Call/Instrument:	INFRA-2007-1.2.1
Grant Agreement:	**212147**

Document	
Deliverable number:	D7.1
Deliverable title:	Enabling Services Enhancement Specification
Contractual Date of Delivery:	1st of February 2008
Actual Date of Delivery:	1st of March 2008
Editor(s):	CNR
Author(s):	Paolo Manghi, Wojtek Sylwestrzak
Reviewer(s):	M.Hatzopoulos
Participant(s):	Michele Artini, Federico Biagini, Natalia Manola, Marko Mikulicic, Vassilis Stoumpos
Workpackage:	WP7
Workpackage title:	Enhancing Infrastructure Sustainability and Research Integration
Workpackage leader:	CNR
Workpackage participants:	NKUA, ICM, CNR
Distribution:	Public
Nature:	Deliverable
Version/Revision:	1.3
Draft/Final:	Draft
Total number of pages: (including cover)	18
File name:	D7_1.pdf

Key words:	Security, Discovery, Registration, Subscription and Notification, Core communication libraries

Disclaimer

Table of Contents

Table of Figures

Summary

This document describes the changes to be applied to the Enabling Services of the DRIVER Testbed in order to enable a sustainable production infrastructure.

1 Introduction

The DRIVER Project delivered a Testbed infrastructure made of a number of Services organized into three main layers: Enabling layer, Data layer and Functionality layer (see Figure 1). The Services were deployed on a number of machines, i.e. *DRIVER nodes*, in Europe and provided an effective and running proof of concept. The DRIVER-II Project has the objective to deliver a running production-quality infrastructure, including all Services envisaged in DRIVER, plus a set of extra Services, supporting Complex Objects storage and management.

This document describes the changes to be applied to the Services in the Enabling Layer of the DRIVER Testbed in order to enable a sustainable production infrastructure (Section 2). Such changes have mainly to do with:

- *Improvement of the Services* (Section 3): refining the internal design of the Services to make them more efficient and stable.
- *Enrichment of the Services* (Section 4): endowing the Services with additional features which will improve the kind of "applications" to be built on top of the infrastructure.
- *Architecture alignment* (Section 5): establishing common design for the DRIVER Service detailed architectures and for the DRIVER Nodes internals.

The changes will be here described from the functional point of view; for the architectural specification details, refer to [1].

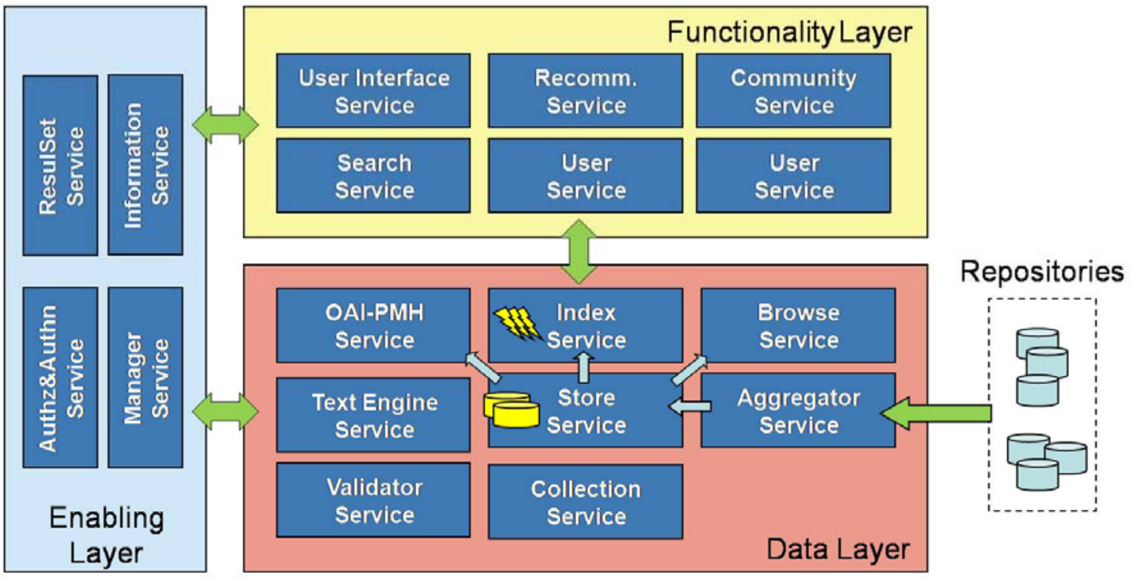

Figure 1 – DRIVER Infrastructure: the Architecture

2 Enabling Services

The DRIVER Enabling Services have been extensively described in the DRIVER Project. Still some introductory explanation is required in order to be able to understand the content of this document, which is mainly destined to developers than generic readers.

The Enabling Services are the heart of the DRIVER Infrastructure. They ensure that DRIVER Services at other layers can safely interact and cooperate to achieve the results expected by the application scenarios running on top of DRIVER. In Figure 2 the layers are visible as well as possible different communities running different applications, sharing part of the services, in the same infrastructure. The services provide the following functionalities.

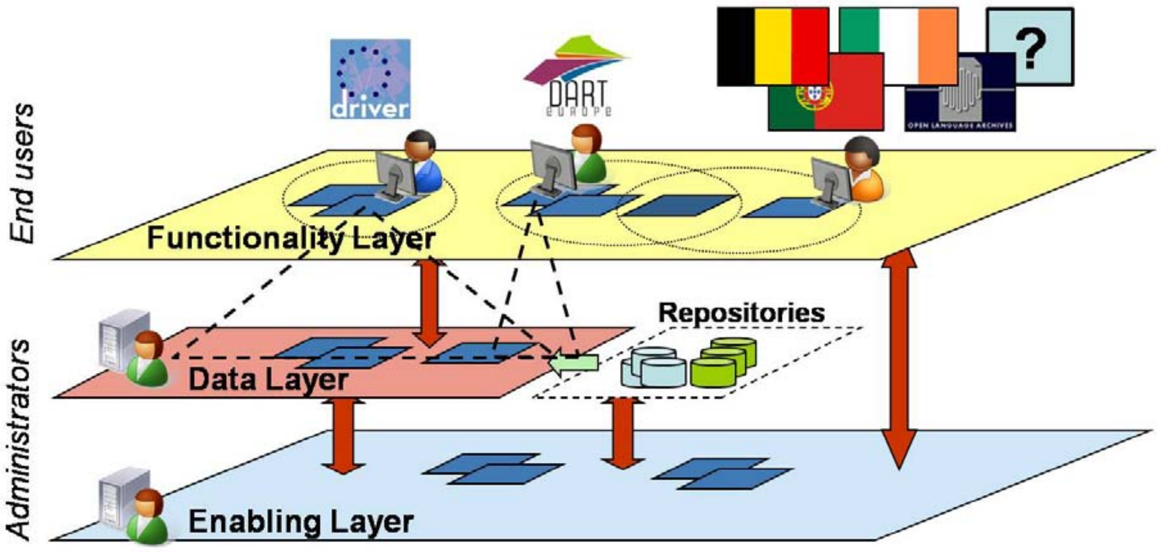

Figure 2 – DRIVER Infrastructure: running environment view

Information Service (resource registration, update and discovery)

Any resource in the system should publish information about its existence, its location and its current status through its own profile. Services can therefore search among the resources available to the system, and find those that match their functional needs. For example, at start-up, each Service needs to interact with the Information Service to register its own profile. Thus, a Search Service needing to execute a query can search in the Information Service the Index Service available (registered) and decide which to use to run the query.

Authentication Service (resource authorization and authentication)

Any resource needs to authenticate before operating within the infrastructure and needs to have the proper rights to execute some operations. Typical examples include user authentication, but also Service authentication and secure Service-to-Service communication, not to occur in malicious attacks.

Manager Service (resource orchestration and monitoring).

Resources available to the system can be orchestrated to accomplish certain operations in a workflow; besides, resource behavior can be monitored automatically, in order to prevent or adjust faulty scenarios. For example, the registration of a new Repository in the system must be coordinated with the creation of the relative storage space into an MDStore Service. The action is ruled by the Manager Service, which waits for special events and reacts accordingly by orchestrating the Services available in the infrastructure.

ResultSet Service (Service APIs and data management)

Services can count on common mechanisms for acquiring or delivering large data sets called ResultSet resources. ResultSets are containers of xml files, which offer paging and optimized caching mechanisms for use by both providers and consumers. Indeed, data providers, i.e. Services that must return a data set (e.g. Index Service on queries, MDStore Service on record requests etc.), may consider to use a ResultSet Service to create their output ResultSets rather than implement the ResultSet mechanism on their own. On the other hand, Service that need to acquire large data sets, do not need to upload the entire set, but can count on ResultSet references and paging functionalities to optimize data transfer.

In the DRIVER project, DRIVER service implementation could be based on different language platforms (Java, Perl and Phyton were used). In particular, services had to independently implement the modules required to interact with the Enabling Services, hence to join and run in the infrastructure context.

In the DRIVER-II Project, the Services will be implemented in the same Java platform and adopt the same CXF-based framework for SOAP and Web Service support. Furthermore, the Services will be based on a number of Core Common Libraries, meant to offer APIs on the most common interaction patterns with the Enabling Services. The gained independence from the Enabling layer enforces the following advantages:

- The time spent to implement a new Service is notably reduced;
- Efficiency, robustness and quality of the interaction with the Enabling layer are isolated to the same set of libraries, which will therefore maximize these parameters once for all;
- The Service implementation is not affected by changes in the implementation of the Enabling Services; the Core libraries will be changed, tested, and sent over for recompilation purposes to all Service developers.

3 Improving the Services

In what follows we describe the improvement to be applied to the Services in the Enabling Layer: Information Service, Manager Service, Authentication and Authorization Service, ResultSet Service and Hosting Node Manager Service.

3.1.1 Information Service

The Information Service has to be improved in two main directions:

- IS-SN (subscription & notification management): the Service conforms to the W3C subscription notification standard [8][9]. However, for debugging reasons, notification messages are currently delivered sequentially rather than being broadcasted;
- IS-Store (storage scalability): currently the IS-Store component is built on top of Exist v1.0, which does not perform well under certain concurrency conditions and over a certain threshold of entries.

IS-SN (subscription & notification management)

The IS-SN is currently designed to send a sequence of notifications to the subscribed Services. Blackboard notifications correspond to action messages sent by the Manager Service to the commanded Service via *orchestration protocol*. According to the protocol (well described in ScrewDRIVER [1]), such notifications have to be "consumed" by the receiver, meaning that the receiving Service should answer to the notification with the right message in its own profile blackboard. Currently, the IS-SN waits for such an answer before the next blackboard notification is sent out. The approach, almost transactional, is clearly tailored for debugging and development issues, but "optimistic" in production, where Services might not answer for some time or adopt different answer techniques; e.g. waiting for an action to be completed as for the UniBi Index Service.

The solution is that of "breaking" the queue dependencies. This can be done by creating notifications and sending them out through two different parallel processes running in the context of the IS-SN. The former will feed a queue of notifications, while the second will try to consume the queue by exploiting time-out methodologies and raising exceptions when needed.

IS-Store (storage scalability)

The solution to the potential bottle-neck, caused by the current implementation in correspondence with the increment of entries and requests, is that of porting to Exist v1.2, which seems to include more efficient algorithms and features, and re-design a distributed low-level architecture (hidden to the consuming Services) for the IS-Store.

The IS-Store should "hide" a *master* physical store plus a number of synchronized *replicas*. The master store is the one in charge of contacting the IS-SN, while all stores can be used for discovery and access.

The approach can be further improved by devising another layer of access to IS-Store, capable of satisfying special discovery needs of Services. The layer should maintain a "hot-cache" of results to some discovery queries, frequently posed by Services. The idea is again that of decreasing the load from the physical stores, but also to speed up the answers to the consuming Services.

3.1.2 Manager Service

The Manager Service needs no special improvements, but enrichments as discussed below. Its re-implementation in Java will also make it more stable and efficient.

3.1.3 Authentication and Authorization Service

The Authentication and Authorization Service needs to implement a higher level of distribution, in order do manage the potentially large amount of Service and User authentication and authorization requests.

3.1.4 ResultSet Service

The ResultSet Service needs to be re-designed in order to be more efficient and more scalable. Its re-implementation in Java should be enough to overcome the current limitations, which are mainly due to the inconsistencies of threads management in the Perl environment.

3.1.5 Hosting Node Manager Service

The Service has to be re-implemented in Java, in order for all sites to have a common Hosting Node Manager technology. Currently the Service does not require improvements, but enrichments as indicated below.

4 Enriching the Services

In what follows we describe the enrichments to be applied to the Services in the Enabling Layer: Information Service, Manager Service, Authentication and Authorization Service, ResultSet Service and Hosting Node Manager Service.

4.1.1 Information Service

In realizing the DRIVER Testbed, the process of Services and "applications" development revealed some flaws which are to be solved in DRIVER-II. In particular, the Information Service has to be enriched in terms of the underlying logic of the resources it will handle. The following changes must be applied at the Repository resource level and at the DRIVER resource meta-level.

Repository Resources

Repository management should be modified in order to favor the generalization of the "harvesting" pattern and integrate it with the ResultSet pattern. Currently, in the IS, Repositories are described as special Service Resources answering to the OAI-PMH protocol interface; Aggregator Services are in charge of harvesting one Repository Service accordingly. This scenario reveals a number of flaws and inconsistencies:

1. Aggregator Services hard-code the notion of OAI-PMH harvesting, channeling records from Repositories to MDStore Data Structures; this means that other Services, for different purposes, cannot OAI-PMH harvest Repositories straightaway, but need to pass through the corresponding MDStore DS or internally re-implement OAI-PMH harvesting. For example, the Validator Service hard-codes the same OAI-PMH harvesting process, because it needs to evaluate the records directly from the Repositories, before these have been even visible and assigned to an Aggregator Service.
2. The harvesting process is limited to OAI-PMH protocol, while others are to be made available in the future.
3. Repositories are indeed Data Structures more than Services: DRIVER Services return ResultSets and are actively registering and updating their profile to the IS.

Repository management logic in D-NET will draw a scenario where:

1. Repository Management Services are in charge of importing into DRIVER data coming from Repository Data Structures. The Service is to be called with a Repository DS and the OAI-PMH call to be called; it will return a ResultSet with the relative results. Note that Repository Management Services can provide consuming Services with extra methods, enriching the OAI-PMH protocol. An example, which might turn out very useful in the DRIVER aggregation process, is that of a method returning OAI-Items from a Repository. The form of the result would be a set of Object Records, as well described in ScrewDRIVER wiki [1].
2. Aggregator Services are separated from Repository Management Services. They do not handle Harvesting Instances anymore, but Transformation Data Structures. Such DSs describe the input metadata formats of the transformation, the output metadata format, and the set of rules in between. See ScrewDRIVER for further details [1].
3. Repository DS can enter different statuses, as defined by the Repository workflow in [7]. According to the workflow a Repository has first to be registered, then be approved by an Aggregator Manager, and finally be harvested to integrate its

content into the DRIVER Information Space. The quality of its content is then measured by means of the Validator Service, which will assign a rank to the Repository. Searches based on the DRIVER repository rank might be possible.

Generic Resources

A number of resource logic notions have to be encoded in Resource Profiles and with new types of Resources:

- *Regions*: set of resources defining the interaction boundaries of such resources. A resource is activated in the context of a region and can only interact with the resources in the same region at that moment in time. For example, an UI belongs to one region; a query execution can only be demanded to the Search Services available in the same region of the User Interface at search-time.
- *Groups*: set of resources bound together by particular application-oriented properties. Groups do not define any interaction boundaries between the resources they contain. For example, the Manager Service deals with scenarios where some orchestration actions need to be sent to the same group of services; e.g. for maintaining Index DS or MDStore DS replicas. By means of groups we can maintain persistently groups of Service resources and write code that copes with groups, no matter what the content is. Other application scenario may apply; for example groups could be used to organize users into subgroups and write functionalities that operate bulk operations on groups of users.
- *API contracts*: a Service implements an API whose *contract* establishes its method's signatures and semantics. WSDL and Java Stub of the Services are available from the support site.
- *Sub-Services*: Services implementing only a part of an API contract. The existence of sub-Services must be declared as an explicit dependency in the IS. Examples are: the embedded ResultSet of the ICM Index obeys only to a sub-part of the ResultSet Service contract, that of ResultSet access. As such, is a sub-ResultSet.
- *Service status*: service profiles should contain XML tags describing their current status (pending, suspended, active etc), which is currently implemented by storing the profiles in separate IS-Store file collections. This requirement naturally calls for the extension of the "group of properties" notion adopted for Repository profiles to all Resource profiles.

The Information Service will have to be aware of the notion of region and return resources discovery results based on the region from which they were invoked. More specifically Services willing to find resources will be returned an answer that depends on the region from which they are acting. For example, the User Interface Service will act in a context that depends on the logged User's region context (for simplicity we may assume that all users are running in the same region as the user interface service). A User firing a query is implicitly asking the UI Service to send the query to one Search Service available in the same region. In turn, the Search Service will discover which Index Data Structures in the same region are available and capable of solving the query.

4.1.2 Manager Service

The Manager Service needs to work based on a special *region* of resources. The infrastructure will feature one or more Manager Services per region, in charge of applying the specific orchestration required within the region.

4.1.3 Authentication and Authorization Service

The Service needs to be aware of the notion of *region* of resources and check for the correctness of Service-to-Service communication. To this aim, Services should always perform actions within a specific region context and therefore interact with others based on the boundaries of such context. Two problems arise here:

1. Region contexts should be part of each Service call to the Information Service and to any other Service;
2. SOAP security protocols implementations offer a complete and tested framework but are known to be cumbersome and have considerable impact on performance; special solutions, exploiting the shape of SOAP envelope messages, are to be studied.

4.1.4 ResultSet Service

No new features need to be added to the ResultSet Service.

4.1.5 Hosting Node Manager Service

In its current implementation the Service is limited to the update of its own profile to the IS with information about the performance measures on the local resources. This information is to be used by the Manager Service to undertake corrective actions, and improve the overall quality of service, or warn administrators of potentially dangerous behaviors.

The next version of the Service will offer monitoring functionalities, by playing the role of Munin and SmokePing *client* and serving one of the available Munin and Smokeping *servers* available. Servers of both kinds gather information about clusters of DRIVER Nodes and send specific alerts to the Manager Services. Alerts are fired by updating in the profile of the Manager Service XML elements corresponding to special harming events.

Other implementations may be considered, following the similar Subscription&Notification pattern and other strategies. For example, a Manager Service dedicated to the management of all regions, rather than to a specific region.

How to form node clusters is to be decided. They could be based on DRIVER regions, proximity, distribution measures etc.

5 Aligning the Services

Although not clearly indicated in the DoW [2], the design of a common DRIVER Node architecture as well as the definition of Core Common Libraries, i.e. common software tools for DRIVER Service construction, is naturally included in this specification.

5.1.1 DRIVER Node Architecture

The whole architecture will be Java-based. The choice does not hinder external technologies, built within different platforms, to be integrated into DRIVER. On the other hand it enables a helpful and uniform way of designing, developing and deploying DRIVER Services. D-NET v1.0 will deliver common Service classes to be implemented, a set of common libraries implementing core functionalities, and the definition of a common DRIVER Node architecture.

In particular a DRIVER Node consists of Java Servlet container (for example, Apache Tomcat) running a Hosting Node Manager Service (HNM). The Service communicates information about the node status to the infrastructure Information Service; such information regards network measures as well as local performance measures.

DRIVER Services

DRIVER Services implement a special DRIVER class Service (but may also not) and run on a DRIVER Node as Servlet Container WebApps. Each Service is packaged as a WAR file, containing the Service classes, resources (property files, spring bean definitions) and dependencies (including the DRIVER Core Libraries, see Section 2.1). Finally, stub classes files are to be placed in a special directory in the WebApps (details can be found at: http://technical.wiki.driver.research-infrastructures.eu/index.php/Software_Development on ScrewDRIVER [1]).

Such scenario is the one that requires minimal modifications to the existing Java DRIVER Services, while leaving the door open to future, more elegant, solutions. One of this would be that of having all Services running inside the same WebApp, with the Hosting Node Manager Service responsible for their life-cycle and their communications with the outside world. In such scenario Services would we packaged in a DRIVER Archive (DAR) and could be deployed on-the-fly as well as switched-off or restarted automatically by the infrastructure (through the HNM) on demand. Not only, incoming and outgoing communications could be filtered and optimized by the HNM: e.g. Services running on the same node could communicate without HTTP/SOAP protocols. Note that, due to the IOC development strategy currently adopted by the DRIVER Java developers, such changes turn out to be quite natural.

Property files

We shall consider two main phases with respect to the alignment of the Services: development and production. The difference between these emerges in the property file of Services, which in development phase may contain references to all Services it needs to refer, while in production it should only contain an indirect reference to the Information Service (probably a constant domain and a reference to a DNS; this way the Services will not even need a special Information Service address) and discover other services dynamically.

During development the property-based configuration of all Services should follow the same nomenclature in order to simplify the operation of deployment of the Services. The structure and organization of these files can be found at: http://technical.wiki.driver.research-infrastructures.eu/index.php/Software_Development, on ScrewDRIVER [1].

5.1.1 Core Common Libraries

DRIVER Services will have to be “aligned”, i.e. build according to the same technology, best practices and methodologies. In particular, they will be Java based, based on the same SOAP support (Apache CXF is the candidate), adhere to the same Service configuration and packaging rules and, where possible, share the same Java libraries. Such libraries, namely Core Common Libraries, deal with functionalities common to all Services for the interaction with the enabling layer Services. The libraries will deal with:

- IS-Registry client: resource registration, deregistration, and resource profile updates;
- IS-Lookup client (resource discovery): finding resource profiles whose properties match given criteria
- Service locator: getting Service stubs;
- IS-SN client: subscribing a topic to the IS-SN.
- AAS client: security management
- Automatic IS registration

Dynamic Resource Discovery

On DRIVER nodes all Services will have to deal with dynamic discovery of Services through the IS. The implication of this requirement is that the configuration files of the Services will only consist of one IP reference to the IS. The approach can be further improved. Since changing the IS reference implies an intervention in the configuration file of all Services referring to the IS, the IS reference can be indirect, i.e. a domain and a DNS address.

6 References

[1] *ScrewDRIVER* Wiki, http://technical.wiki.driver.research-infrastructures.eu

[2] DRIVER Annex I - “Description of Work”, Proposal no. 212147.

[3] *Munin* Project. http://munin.projects.linpro.no

[4] D6.1 Software Release Plan

[5] D6.2 Supporting Tools and Databases

[6] D7.2 Overall Research Design Report

[7] D5.1 Information Space Report

[8] Web Services Base Notification 1.3 by OASIS. http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf

[9] Web Services Topics 1.3 by OASIS. http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf