Quality Assurance in perfSONAR Release Management

Jeff W. Boote*, Andreas Hanemann†, Loukik Kudarimoti‡, Panagiotis Louridas§, Luís Marta¶, Michalis Michael‖, Nicolas Simar‡ and Ilias Tsompanidis§
*Internet2, 1000 Oakbrook Drive, Suite 300, Ann Arbor, MI 48104, USA
boote@internet2.edu
†German Research Network (DFN), c/o Leibniz Supercomputing Center
Boltzmannstr. 1, 85748 Garching, Germany, hanemann@dfn.de
‡DANTE, 126-130 Hills Road, Cambridge CB2 1PG, United Kingdom
{loukik.kudarimoti,nicolas.simar}@dante.org.uk
§Fundaç̄ao para a Computaç̄ao Científica Nacional (FCCN)
Av. do Brasil, 101, 1700-066 Lisboa, Portugal, lmarta@fccn.pt
¶ Cyprus Research and Academic Network (Cynet), c/o Information System Services
University of Cyprus, P.O. Box 20537, 1678 Nicosia, mikem@ucy.ac.cy
‖ Greek Research and Academic Network (GRNET), 56 Mesogion Ave., 11574 Athens, Greece
itsomp@ccf.auth.gr, louridas@grnet.gr

***Abstract*—Software release management is closely related to the management of software quality since only software with assured quality should be provided to users. While established best practices exist for the development of software within an organization, new challenges arise with the introduction of Service Oriented Architectures which make it possible to develop loosely coupled systems potentially involving different organizations. For these systems it is not sufficient to test parts individually, but the collaboration issues need to be taken into account.**

In the perfSONAR project a set of loosely coupled web services has been developed to perform and manage measurements of network performance in research backbone networks. For the transition of the service development into the provisioning of permanently operated services, a release management process has been devised. It is presented in this paper highlighting the aspects being taken into account. These are also relevant for similar projects where Service Oriented Architectures are deployed.

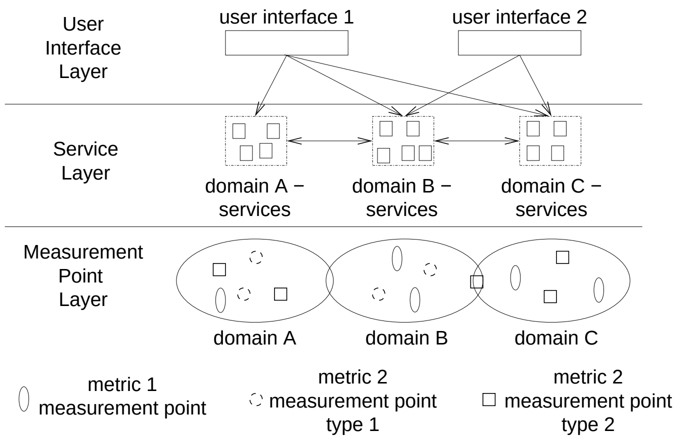

Fig. 1. JRA1 architecture proposal

I. Introduction

The perfSONAR project [1] is a cooperation between the EU-funded GN2 JRA1 project, Internet2 and ESnet to deliver a framework for network performance measurements in research backbone networks. The framework together with measurement and visualization tools will be deployed and operated on a permanent basis within the partner networks.

For the implementation of the framework a Service Oriented Architecture [2] has been devised which is reflected in the name of the framework (*Performance focused Service Oriented Network monitoring ARchitecture*). It is depicted in Fig. 1. In the lower layer measurement tools are in place to perform active or passive network monitoring and to measure metrics like utilization, delay, jitter, packet loss, or available bandwidth. For managing the measurements within a domain, but also inter-domain a set of services has been developed. These provide functionalities like the archiving of measurements, service lookup, or authentication. Visualization tools make use of the services to provide measurement results to different user groups in a suitable manner.

For the transition from the development of services to a deployment and operation on a permanent basis a release management process has been installed which is subject to this paper. As the services are developed by different small groups at the project partner networks and due to the loosely coupling of the services, the testing within the release management has to ensure the collaboration among the services. Therefore, best practices for release management and testing have been extended towards the multi-domain background of the project.

The paper is organized as follows. In Section II the requirements that have to be taken into account for the release management in perfSONAR are derived. These are compared with the state-of-the-art in release management for which models in software management and service management are reviewed (Section III). The release management process is presented in Section IV and the lessons learned including test cases from the first two releases are given in Section V. The future deployment of perfSONAR services as so called *Multi-Domain Monitoring (MDM)* services providing some details about the pilot installation within the Portuguese research network FCCN is outlined in Section VI. The conclusion in Section VII highlights the relevance of this work for similar SOA-based projects.

II. Release Management Challenges

The general aim of release management is to provide services for installation with an assured service quality. Quality in this sense does not only mean that services should be error-free, but should also have provide those functionalities as previously specified. For perfSONAR these aims can be detailed as follows.

a) Release bundling: There is a trade-off between the aim to have a clear versioning of services and a delay in the provisioning of new functionalities. On the one hand, there is the possibility to create a complete bundle of services which are released together so that it can be assured that these services are interoperable. As the installation of such a complete bundle of services is quite time consuming, such a kind of release can only happen with a period of several months. A drawback is that new functionalities or new services are not provided until the time of a new release so that new functionalities do not become available. Here, an additional release method for early adoption may be helpful.

b) Interaction testing: The individual testing of services has to be enhanced with a testing of their collaboration. For doing so, the external specifications of the service functionality (as NMWG [3] descriptions in perfSONAR) have to be applied to construct appropriate test cases.

c) Documentation requirements: For the services a set of documents has to be provided including installation instructions, functionality and usage documentation, and contact data.

d) Visualization tool release: Since the measurements are often of a limited usefulness without visualization tools, release management for visualization tools has to be part of the overall release management. In addition to searching for bugs in the implementation, release management for visualization tools has to ensure that the tools also provide functionalities that are appropriate for getting the view on the measurement data.

e) MDM pilot requirements: In the GN2 project a rollout of services is going to be fixed as part of the project contract. Therefore, the release management has to be compliant with the constraints that are agreed. For instance, the contract will specify conditions to ensure the maintenance over several years.

III. Related Work

In the following the contributions of existing recommendations for service quality assurance are examined with respect to the requirements. These can be decomposed into software-specific guidelines and service management frameworks.

A model that has proven to be useful in practice is the development model for the FreeBSD operating system [4] which allows for the contribution of several thousand developers. Approximately 300 of those have write access to the project's code versioning system (CVS), while the overall development is run by a small group of senior engineers. The development code base is split up into current, stable and release branches. For each release three time periods of 15 days are applied for checking the code.

The FreeBSD project is a very advanced project so that its release management can be regarded as mature. It has therefore been selected as basis for release management in perfSONAR where the time periods and branch concepts have been adopted. However, extensions are necessary with respect to the loosely-coupled development resulting from the SOA and further constraints.

One important aspect of quality assurance are testing methods. It is distinguished between white box and black box testing where internal knowledge about the implementation about a component is used or not. For Java which is the programming language used for the majority of perfSONAR services the JUnit framework [5] exists which can be applied to write automated tests for Java classes and methods. Apart from this white box testing, black box testing is relevant to perform interactions as specified in the functionality of each service.

The IT Infrastructure Library (ITIL) [6] is a collection of best-practice recommendations for IT service providers which is already widely adopted in the industry. Within its Service Support Set it contains a release management process which has however a different focus as the one in our scenario. The aim of ITIL is to manage the installation of new software (or other resources) in an organization, which is usually not developed by organization itself, and its effects on the services provided to users. Nevertheless, the recommendations are useful for organizations that adopt perfSONAR. In the following months it will be examined whether recommendations for quality indicators in COBIT (Control Objectives for Information and related Technology) [7] can be helpful for quality assurance in perfSONAR.

IV. perfSONAR Release Management Process

The release management process of perfSONAR which is specified with respect to the requirements in Section II is presented in the following. It is explained where related work has been applied and where extensions have been necessary.

A. Release of Web Services

The release of the perfSONAR web services comprises guidelines for groups involved in the development, release types, code development phases and in particular for the testing methods.

1) Group Definitions: The perfSONAR source code is publicly available in a Subversion (SVN) [8] repository which allows for unrestricted read access. It has replaced an initially used CVS to profit from the advanced features of the system. In order to be able to manage the code quality, the write access is limited according to the group concept which is an extension of FreeBSD groups.

The *Steering Committee* decides about the general direction of the perfSONAR development, i.e. which services and service functionalities should be developed, who can get involved in the project and about principle changes of the project. It also decides about granting write access to the repository. The Steering Committee has dedicated a smaller team of three people to manage the release process of services which is called *Release Engineering Team*.

The *Authorized Code Committers* group is composed of developers (approximately 15) that work on perfSONAR web services and therefore are allowed to change the code in the repository. The developers are responsible for white box testing their own code.

The *Testing Team* which is responsible for the functional testing of services is currently composed of two people who are only occupied with testing. In addition to giving recommendations for white box testing, the Testing Team writes tests for the black box testing and interacts with the code developers to remove bugs that are witnessed when executing the tests.

A dedicated *Documentation Team* is going to be installed in the future to ensure the quality of the documentation.

2) Micro Releases and perfSONAR Bundle Releases: For the first release of perfSONAR a release process similar to FreeBSD has been examined which results in the provisioning of a complete set of services at the same time.

However, it has turned out that this method is not suitable for perfSONAR where we have a dedicated team of software testing experts. Therefore, the idea is that the services are tested on an individual basis as described below and can be released in so called *Micro Releases*. A Micro Release therefore means that the service is fully tested (internally and functionally), but that it is not tested for collaboration with other services. These releases aim at early adopters who are interested in advanced features of a particular service.

An additional process called *Hand Over Process* has been installed to combine individually released services into *perfSONAR Bundle Releases*. For the hand over, several information templates have to be completed as a prerequisite for the release.

a) Functional Specification: This document contains semantical information about the functionalities of a service. It should be started early in the service development and continuously maintained.

b) Interface Specification: For testing and users who would like to write their own clients the detailed interface specification is the basis for their work. It describes the exact syntax of the XML queries that are supported by the perfSONAR web service.

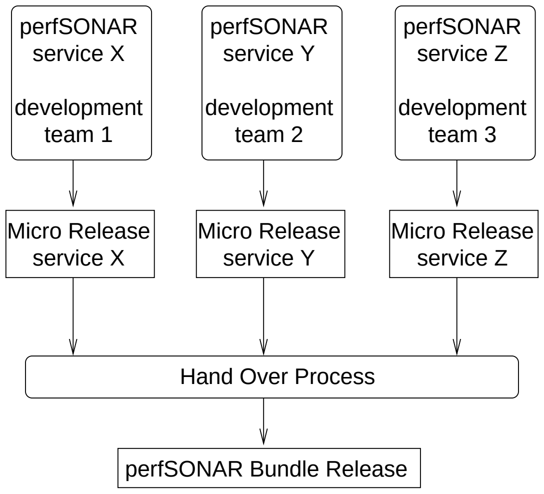
perfSONAR service X development team 1
perfSONAR service Y development team 2
perfSONAR service Z development team 3
Micro Release service X
Micro Release service Y
Micro Release service Z
Hand Over Process
perfSONAR Bundle Release

Fig. 2. Two kinds of releases in perfSONAR: Micro Releases and Bundle Releases)

c) Installation Actions: For installing a perfSONAR web service the steps that have to be carried out are explained in this document. The bundling in the hand over process will combine these documents because a unified installation of perfSONAR is targeted.

d) Metadata Configuration and Sample Configuration: The configuration of an installed service makes use of the first document, while some examples from early adopters are given in the additional samples document.

For the hand over of a perfSONAR Bundle Release additional tests are necessary. These relate to the installation of services where it necessary to make sure that services can be easily installed. Such tests also have to consider the possibility that services may be installed on the same machine which should not lead to unforeseen conflicts.

Another kind of testing is also needed as this point to verify the collaboration of services in the bundle to achieve common goals. This aspect is related to the SOA and should examine workflows as specified by typical use cases.

3) Code Development Phases for Micro Releases: In the code development for perfSONAR three code development branches are distinguished. The CURRENT branch contains the newest versions of the services being developed. The STABLE branch contains tested versions of the services and is available as weekly snapshot for interested parties. The STABLE branch is the basis for the creation of releases which go later into a special RELEASE branch. This naming has been adopted from FreeBSD.

FreeBSD has also been the source for designing three phases for the Micro Releases as depicted in Fig. 3.

a) MFC sweeps period: New releases of perfSONAR web services are derived from the STABLE branch at individual time intervals for the services. The Micro Release process starts 45 days in front of the release date and is announced to all developers after it has been previously discussed with the Release Engineering Team. During the next 15 days developers usually perform so called MFC (“merge from CURRENT”)

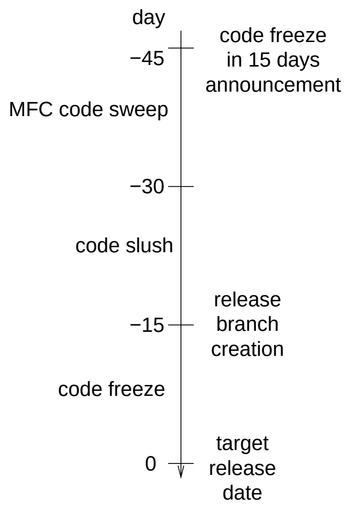

Fig. 3. Code development phases in perfSONAR (refinement of [4])

sweeps which means that (white-box) tested code from the CURRENT branch is transferred to the STABLE branch. This is the last possibility to perform functional changes.

b) Code review period: Thirty days before the anticipated release, the source repository enters a "code slush". During this time, all commits to the STABLE branch must be approved by the Release Engineering Team. The kinds of changes that are allowed during this 15-day period include bug fixes, documentation updates, security related fixes, and changes the Release Engineering Team feels are justified given the risk. Tests are performed by the Testing Team.

c) Testing period (Micro Release): After the 15 days of the code slush, a release candidate is released for widespread testing and the code enters a code freeze where it becomes much harder to justify new changes to the system unless a serious bug-fix or security issue is involved. During the code freeze, at least one release candidate is released per week and tested by the community until the final release is ready. After approval by the Release Engineering Team, the Micro Release is announced.

4) Phases for the Hand Over Process: Phases are also defined for the Hand Over Process which are needed to ensure the collaboration of the web services.

a) Selecting services period: The release process of perfSONAR bundles starts with the selection of services that go into the bundle. Usually, a bundle will consider updates of those services that have been part of previous bundles, but also new services for which Micro Releases have been performed. As a prerequisite for the acceptance of a service, the five documents described above have to be delivered. These are compiled into summary use cases and installation instructions. Initially, a period of four months has been envisioned for the frequency of perfSONAR Bundle Releases.

b) Workflow and installation testing period: Similar to the code review period for Micro Releases, the Testing Team is taking care of testing the services which is done with respect to verifying the collaboration among services and for testing the joint installation procedures (Ant targets). Here, e.g. problems related to dependencies on software packages can be detected.

c) Testing period (Bundle Release): A testing period for early adopters of perfSONAR Bundle Releases comparable to the testing period for individual services is also foreseen for bundle releases. Based on the experience from the perfSONAR release 2.0, testing periods of 25 days each are needed for workflow and installation testing and for the testing or release candidates.

5) Testing Methods: For Micro Releases both white box and black box testing are applied. For the white box testing of services JUnit tests are used which are written in parallel to the code development. The JUnit framework is a mighty framework and turned out to be useful for this purpose. These tests are performed by the code developers on their own for which guidance is giving by the testing team.

Conceptually more interesting is the black box testing part where the XML interactions that have been implemented are tested. It has been decided to categorize the tests according to four criteria.

1) Tests with allowed values which are typically encountered. These tests are useful to check the compliance with functionality semantics.
2) Special tests with border values which are often causing problems like "off-by-one" in arrays.
3) Tests with not allowed values, both near the border and far away from the border to examine whether the service responds with a predefined error message.
4) Some tests are also needed for checking the error messages on wrong syntax in the XML query generation.

Due to the exponential growth in the number of possible tests with respect to the number of parameters, a subset of tests has to be defined. In perfSONAR this is done according to the possible interactions so that a sufficient of number of tests for the parameters in the interactions are performed. Some tests with wrong syntax are also carried out for the service in general.

Similar to JUnit the functional tests in perfSONAR are written in Java classes so that they are available also for the checking the effects of changes lateron.

The white box and black box testing described so far are applicable for the Micro Releases. Additional tests are needed for the Bundle Releases where the descriptions of complex workflows that involve multiple services are taken as basis. These workflows are used to construct more complex test cases involving several services.

B. Release of Visualization Tools

The release of visualization tools that make use of the perfSONAR services is going to be addressed in the next months before the MDM pilot (see Section VI) starts. The aim is that visualization tools are available for the implemented web services so that the benefit arising from the execution of measurement is enhanced with respect to the user needs.

The release process of the visualization tools is going to run in parallel with the development of services. Once stable branches are created for services, the visualization tool development will target to be able to collaborate with the updated services. Here, it is distinguished between minor updates and new functionalities/new services, where the latter ones require a more long term preparation (three months advance planning).

For testing the visualization tools, the use of automated GUI testing tools requires much effort so that it has been decided to perform manual tests of the GUIs instead. For each visualization a special testing document is going to be provided where tests that should be carried out are described. For testing the internals of a visualization tools, methods like JUnit also have to be applied.

One thing that makes the development of visualization tools a bit less critical is that the tools do not require time consuming updates by the user. The current tools either use the Java mechanism "JavaWebStart" to automatically download the newest version from the server or run completely in a Web browser where the user does not have to install anything.

V. Experiences from the First Releases

A first release of perfSONAR became available in July 2006, while the second release has been provided in March 2007 (see Fig. 9). In the following the services being part of the releases are briefly explained and the lessons learned are summarized. Furthermore, two examples from the functional testing are provided.

A. perfSONAR 1.0 Release

The first release of perfSONAR has contained the *Round Robin Database [9] Measurement Archive (RRD MA)* service and the *Lookup Service (LS)*. The RRD makes utilization data available to the perfSONAR framework by implementing a wrapper around a special kind of database. The LS is a major building block for the flexibility of the framework and provides information about currently available other services.

Lessons learned: The installation of these services turned out to be a major difficulty in the adoption of perfSONAR. It has not been possible that the two services share an installation of the Tomcat [10] application server which has resulted in some misunderstandings. These experiences have led to the adoption of installation testing as part of the hand over process.

The Testing Team had some difficulties building the functional testing scripts, due to the lack of documentation provided by the development teams. The Release Engineering Team has learned that it is necessary to ask the developers of the chosen services for detailed documentation about each possible input accepted or output generated by each service, and which parameters are mandatory and which are optional.

Another lesson was that the production of release candidates is a very important part of the release process, mainly because of the bugs and other issues found by the community of users and early adopters that installed those release candidates.

User feedback has shown that the installation of basic software needed by the services (Axis, eXist, Tomcat, etc.) was still too complicated. The Release Engineering Team has decided that it is necessary to build an automatic installation mechanism for these dependencies, which is a solution ideal for users, or to use package management solutions. Lack of information and automation concerning the configuration of the services was another issue pointed out by users of perfSONAR, and it was also something to improve on the next release, both on the documentation and programming sides.

To tackle the overall feeling that perfSONAR was too difficult to install, the Release Engineering Team proposed that installation steps have to be consistent for all services, and comprise four steps: 1. pre-install, 2. configure, 3. deploy, 4. test. Furthermore, every service has to provide separated installation scripts that automate each step, and then a bundle installation script is going to function as a wrapper around the individual service installers.

B. perfSONAR 2.0 Release

In addition to updates of the two services released as part of perfSONAR 1.0, the second release has included four additional services. The *SQL (Structured Query Language) MA* has similar capabilities as the RRD MA, but is a wrapper around another kind of database and can provide layer 2 status data in addition to the utilization data. The Telnet/SSH Measurement Point (Telnet/SSH MP) is a service that executes queries to Cisco or Juniper routers within a network and translates common commands into specific commands for the router type. The Command Line Measurement Point (CLI MP) is a wrapper around a set of testing tools such as ping and traceroute. It can also make use of BWCTL (BandWidth ConTroLer) tests of the available bandwidth for which also a special service, the BWCTL Measurement Point (BWCTL MP) has been included into the release.

Lessons Learned: In this release, the documentation written by the development teams and given to the Testing Team was greatly improved, but the Testing Team was still not satisfied, mostly with the level of detail on documentation about result and error codes of each service, and of the explanation of the business logic behind a service. It was very important to use a bug reporting tool, Bugzilla [11], to help on the the flow of information between the Testing Team and the Development Teams, to control the bugs assigned to each version of each individual service, and to be able to follow the resolution of each issue. Something that will be investigated for the next release is the capacity of this tool to avoid the duplicated reporting of bugs using a matching to already reported problems.

During this run of the release process, it was evident that a release specification document has to be written by the Release Engineering Team together with a Development Team Leader, clearly specifying what is expected from the developers in the next release, including software features, installation, configuration, documentation and what versions of basic software are going to be used.

Even though the bundle installation process was greatly improved with the lessons learned from the first release, there is a need to further enhance the ease of installation of each service, namely trying to make the installation questions identical for all services, and improving the stitching part (configuration) of each service. A software update functionality also is going to be studied, so that users will not have to install the whole new package when only a part of it has changed on a release.

Testing examples: In addition to the general lesson learned, two examples of errors and behavior not coherent with the perfSONAR specifications are given in the following which have been resolved in the functional testing. The examples relate to the Lookup Service and the SQL Measurement Archive Service.

The Lookup Service is a crucial part of the perfSONAR framework, as it enables end users to locate perfSONAR services according with their capabilities or their location. Each service may use LS requests to register (LSRegister) and therefore announce its presence, deregister (LSDeregister) or update a previous registry (LSUpdate). End users can query the service using XPath/XQuery expressions encapsulated in a request especially defined for this functionality (LSQuery).

The SQL MA provides access to a relational database containing information about the L2 Path Status and utilization metrics. An end user can use the service for retrieving measurements regarding a certain interface or path directly or with the help of a key structure which was previously retrieved by the service. In addition the service, with the help of a specially defined request, provides users the ability to store new measurements inside the relational database.

In the case of the LS, functional testing helped in revealing a significant error in the implementation and a possible security problem. The aim of this test case was to test the update functionality of the LS. An update request was constructed aiming to update a previously made registry, describing a perfSONAR service. The request used the previously stored access point element of the service as a key, in order to modify the registry by changing the access point value and some description data. The request is shown in Fig. 4.

The service response is depicted in Fig. 5. It is informing us that the update process was a success and the registry now contains the new access point and data. The service should also erase the old registry and replace it with the new updated registry. Relying only in the response from the service would be misleading, since the response does neither provide any proof if the changes in the database containing the registry have been made correctly nor if the old registry was deleted. In order to check this, access to the database is needed afterwards, firstly to check if the updated data exist and secondly to check if the old registry was deleted. A piece of code doing just that is shown in Fig. 6. This code is using internals of the implementation of the LS and has therefore to be regarded as white box testing code, in contrast to the execution of the queries before. It helps to pinpoint to issues in the internal realization of the service and has been written in collaboration with the service developer.

Let us explain what the previous piece of code actually does. At first, queries are constructed in order to test if the new access point and data are stored in the database. Also queries are constructed to check if the old access point and data were deleted. Then, the results of these queries are retrieved from the database and the result code of the response is checked. Afterwards, the code checks if any results were returned by the database regarding the new access point and if there are any results about the old access point. If no results are returned for the new access point or if results are returned for the old access point, the test is a failure. Next, the results regarding the new access point are checked against the expected value. If this is also true, then the tests moves forward and checks the data in a similar way as the access point. In order for the test to be successful, the retrieved new data value must match to the expected data value and there should be no results returned regarding the old data. If all checks are successful, the tests returns true and the service is considered to have passed that particular test.

In the case of our service the test failed, because the old access point and the old data were not deleted and subsequently co-existed with the new access point and data. This could lead end users to obtain wrong information about perfSONAR services thus making the LS unreliable. Furthermore, additional checking revealed that these old entries were never removed from the database, as it should have happened with the use of an Clean Up scheduler which has been integrated into the service. So the former entries became transparent to the service, occupying valuable resources and kept growing after such similar update. The danger of spending system resources is clearly also a security threat as the exploitation of this weakness of the service can bring the service down or the entire server. After the bug was reported the developer fixed the problem and provided another Release Candidate (RC).

perfSONAR specifications demand that if an error occurs in the service or if a request sent by a user is not properly constructed, the service should fail “gracefully”; meaning that the service should respond with a response message containing a result code describing the type of error or malfunction that has happened, rather than the user facing a SOAP (Simple Object Access Protocol, [12]) error on his screen. Although testing for this specification was not the purpose of the constructed tests, the compliance of the services with the former specification was indirectly tested, since if the service did not fail gracefully under the tested conditions, this would be immediately apparent to the tester. Such incident occurred during the functional testing of the SQL MA. A test request was constructed in order to check the behavior of the service in cace of malformed or not properly constructed requests. The test expected the service to response with a result code specifying what went wrong. Instead a SOAP error appeared thus revealing that the service did not fail “gracefully”. The request is shown in Fig. 7.

The request should have contained an event type element inside the metadata element. In the absence of such element the service should respond with a response message containing a result code and describing what went wrong. The proper response is shown in Fig. 8.

Instead the user faced a SOAP error on his screen. Producing proper responses containing proper result codes is impor-

```xml
<?xml version="1.0" encoding="UTF-8"?>
<nmwg:message xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" xmlns:perfsonar="http://ggf.org/ns/
nmwg/tools/org/perfsonar/1.0/" xmlns:psservice="http://ggf.org/ns/nmwg/tools/org/perfsonar/
service/1.0/" xmlns:nmwgt="http://ggf.org/ns/nmwg/topology/2.0/" xmlns:nmtm="http://ggf.org/ns/
nmwg/time/2.0/" xmlns:netutil="http://ggf.org/ns/nmwg/characteristic/utilization/2.0/" xmlns="
http://ggf.org/ns/nmwg/base/2.0/" type="LSRegisterRequest" id="msg1">
 <nmwg:metadata id="serviceLookupInfo0">
  <nmwg:key>
   <nmwg:parameters id="param1">
    <nmwg:parameter name="lsKey">http://update_request_2_1_0.net:8080/axis/services/MA
    </nmwg:parameter>
   </nmwg:parameters>
  </nmwg:key>
  <perfsonar:subject id="commonParameters">
   <psservice:service id="serviceParameters">
    <psservice:serviceName>My_MA</psservice:serviceName>
    <psservice:accessPoint>http://new_update_request_2_1_0.net:8080/axis/services/MA
    </psservice:accessPoint>
    <psservice:serviceType>MA</psservice:serviceType>
    <psservice:serviceDescription> A testing MA</psservice:serviceDescription>
   </psservice:service>
  </perfsonar:subject>
 </nmwg:metadata>
 <nmwg:data id="data0" metadataIdRef="serviceLookupInfo0">
  <nmwg:metadata id="meta1">
   <netutil:subject id="subj1">
    <nmwgt:interface>
     <nmwgt:hostName>xyz.sdf.edf.edu</nmwgt:hostName>
     <nmwgt:ifName>uknown</nmwgt:ifName>
     <nmwgt:ifDescription>not_initial_data</nmwgt:ifDescription>
     <nmwgt:ifAddress type="ipv4">123.23.34.0</nmwgt:ifAddress>
     <nmwgt:direction>in</nmwgt:direction>
     <nmwgt:capacity>1000000000</nmwgt:capacity>
    </nmwgt:interface>
   </netutil:subject>
    <nmwg:eventType>utilization</nmwg:eventType>
  </nmwg:metadata>
 </nmwg:data>
</nmwg:message>
```

Fig. 4. NMWG request for the Lookup Service

```xml
<?xml version="1.0" encoding="UTF-8"?>
<nmwg:message id="msg1_resp" messageIdRef="msg1"
 type="LSRegisterResponse" xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/">
 <nmwg:metadata id="resultCodeMetadata">
  <nmwg:eventType>success.ls.register</nmwg:eventType>
  <nmwg:key id="localhost.localdomain.-6cb9e676:10f3cae1b13:-7ee6">
   <nmwg:parameters id="localhost.localdomain.-6cb9e676:10f3cae1b13:-7ee5">
    <nmwg:parameter name="lsKey" value="http://update_request_2_1_0.net:8080
     /axis/services/MA"/>
   </nmwg:parameters>
  </nmwg:key>
 </nmwg:metadata>
 <nmwg:data id="resultCodeData" metadataIdRef="resultCodeMetadata">
  <nmwg:datum value="Data has been registered with key [http://update_request_2_1_0.net:8080
   /axis/services/MA]"/>
 </nmwg:data>
</nmwg:message>
```

Fig. 5. NMWG response for the Lookup Service

```
// Retrieving result code
String resultCode = ((response.getRootElement()).getChild("metadata", nmwg)).getChild
 ("eventType", nmwg).getText();

// creating the old access point
String accessPointOld = "http://update_request_" + testCase + "_" + subCase + "_0.net:8080
 /axis/services/MA";

// creating the new access point
String accessPointNew = "http://new_update_request_" + testCase+ "_" + subCase + "_0.net:8080
 /axis/services/MA";

// creating the query to check the new access point
String metadataQueryNew = "for $a in /nmwg:store/nmwg:metadata//psservice:service[psservice:
 accessPoint='" + accessPointNew+ "'] return $a/psservice:accessPoint/child::text()";
// creating the query to check the old access point
String metadataQueryOld = "for $a in /nmwg:store/nmwg:metadata//psservice:service[psservice:
accessPoint='" + accessPointOld + "'] return $a/psservice:accessPoint/child::text()";

// creating query to check the old data
String dataQueryNew = "for $a in /nmwg:store/nmwg:metadata let $metadata_id:= $a/@id let
$data := /nmwg:store/nmwg:data[@metadataIdRef=$metadata_id]"+ " where $a/perfsonar:subject/
psservice:service[psservice:accessPoint='" + accessPointNew + "'] return $data/nmwg:metadata";

// creating query to check the new data
String dataQueryOld = "for $a in /nmwg:store/nmwg:metadata let $metadata_id:= $a/@id let $data
:= /nmwg:store/nmwg:data[@metadataIdRef=$metadata_id]" + " where $a/perfsonar:subject/psservice:
service[psservice:accessPoint='" + accessPointOld + "'] return $data/nmwg:metadata";

ResourceIterator meta_ItNew;
try {
 meta_ItNew = dbClient.queryDB(metadataQueryNew).getIterator();
 ResourceIterator meta_ItOld = dbClient.queryDB(metadataQueryOld).getIterator();
 ResourceIterator data_ItNew = dbClient.queryDB(dataQueryNew).getIterator();
 // System.out.println(getResult(data_ItNew));
 ResourceIterator data_ItOld = dbClient.queryDB(dataQueryOld).getIterator();

// checking if the result code is the appropriate
 if (resultCode.equals("success.ls.register")) {

// checking to see if the old access point is deleted
// and the new access point is registered

  if ((meta_ItNew.hasMoreResources())&& (!meta_ItOld.hasMoreResources())) {
   if (meta_ItNew.nextResource().getContent().equals(accessPointNew)) {

// checking to see if the new data are in place
    if ((data_ItNew.hasMoreResources()) && (!data_ItOld.hasMoreResources())) {
     String storedData = getResult(data_ItNew);

//Are the stored data the updated data?
     if (data.equals(storedData)) {
      return true;
     } else return false;
    } else return false;
   } else return false;
  } else return false;
 } else return false;
} catch (XMLDBException e) {
 e.printStackTrace();
}
```

Fig. 6. Code example for white-box testing the internals of the Lookup Service

```
<?xml version="1.0" encoding="UTF-8"?>
<nmwg:message xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/" xmlns:nmtl2="http://ggf.org/ns/
nmwg/topology/l2/3.0/" xmlns:nmwgtopo3="http://ggf.org/ns/nmwg/topology/base/3.0/" xmlns:
select="http://ggf.org/ns/nmwg/ops/select/2.0/" xmlns:ifevt="http://ggf.org/ns/nmwg/event/
status/base/2.0/" type="MeasurementArchiveStoreRequest">
 <nmwg:metadata id="meta1">
  <nmwg:subject id="subject1">
   <nmtl2:link>
    <nmtl2:globalName type="logical">PSNC-DFN-MUE-003</nmtl2:globalName>
   </nmtl2:link>
  </nmwg:subject>
 </nmwg:metadata>
 <nmwg:data id="data1" metadataIdRef="meta1">
  <ifevt:datum timeType="Unix" timeValue="1170878409">
   <ifevt:stateAdmin>PSNC</ifevt:stateAdmin>
   <ifevt:stateOper>up</ifevt:stateOper>
  </ifevt:datum>
 </nmwg:data>
</nmwg:message>
```

Fig. 7. NMWG request for the SQL MA

```
<?xml version="1.0" encoding="UTF-8"?>
<nmwg:message id="localhost.localdomain.-73a96cb2:111735abda6:-2f6c_resp"
 messageIdRef="localhost.localdomain.-73a96cb2:111735abda6:-2f6c"
 type="MeasurementArchiveStoreResponse" xmlns:nmwg="http://ggf.org/ns/nmwg/base/2.0/">
 <nmwg:metadata id="resultCodeMetadata">
  <nmwg:eventType>error.ma.query</nmwg:eventType>
 </nmwg:metadata>
 <nmwg:data id="resultDescriptionData_for_resultCodeMetadata"
  metadataIdRef="resultCodeMetadata">
  <nmwgr:datum xmlns:nmwgr="http://ggf.org/ns/nmwg/result/2.0/">SQLTypeMAServiceEngine.
   getStoreKey: eventType in the request metadata is empty</nmwgr:datum>
 </nmwg:data>
</nmwg:message>
```

Fig. 8. NMWG response for the SQL MA

tant to perfSONAR services in the terms of user friendliness. A user facing a SOAP error has many difficulties understanding what went wrong or if he is to blame for this behavior. It is also important for services using perfSONAR services such as visualization tools to have a response under any circumstances with a result code, since a SOAP error is hard to interpret. The developer of the service was informed of the service misbehavior and produced another RC fixing the problem.

VI. MDM Service and Pilot Phase

Apart from depicting the perfSONAR releases, Fig. 9 shows the timeline of the pilot instatllation of perfSONAR services in several National Research and Education Networks (NRENs) and GEANT2 over the following months. The aim of the deployment is to collect feedback from people working in the Network Operation Centers (NOCs) and to refine the services for a permanent installation. A first phase from June till October will include five NRENs, while 11 NRENs in sum are going to operate the services in the second phase from December 2007 till April 2008. The final widespread deployment in Europe is then planned for the period starting in June 2008.

FCCN has volunteered to install perfSONAR services already in the first phase and has opted for the variant *Managed Service* where maintenance of the service is operated by the perfSONAR group. The installation will include three servers for BWCTL tests of available bandwidth and also for remotely managed measurements of delay, jitter, and packet loss. Furthermore, a server for exporting layer 2 status data making use of the *SQL MA* and a server for installing the SSH/Telnet MP is going to be delivered. In addition to a server for providing utilization data using the RRD MA, all services will be registered with an LS.

The CNM visualization tool will provide a dedicated FCCN map for use by FCCN member institutions and interested end users. All services will be accessible by perfsonarUI, another visualization tool.

VII. Conclusion and Future Work

In the paper the release management process in perfSONAR has been presented which aims at ensuring a high quality of the developed web services and visualization tools. Due to the SOA of the project additional considerations have been necessary to ensure for the reliable collaboration of the developed services. As SOAs have attracted an increasing interested

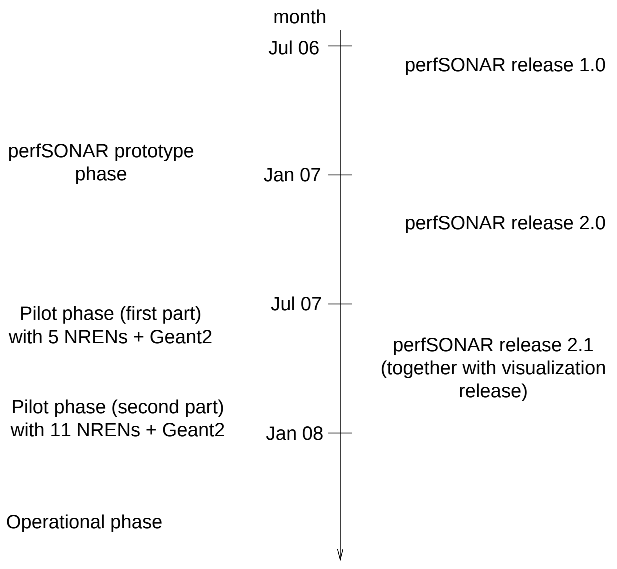
month
Jul 06
perfSONAR release 1.0
perfSONAR prototype phase
Jan 07
perfSONAR release 2.0
Jul 07
Pilot phase (first part) with 5 NRENs + Geant2
perfSONAR release 2.1 (together with visualization release)
Pilot phase (second part) with 11 NRENs + Geant2
Jan 08
Operational phase

Fig. 9. Timeline for perfSONAR releases and MDM pilot deployment

of the previous years, the release management process with its recommendations for micro and bundle releases, necessary documentation, and testing methods is also valuable input for related projects.

References

[1] "perfSONAR project," http://www.perfSONAR.net.
[2] A. Hanemann, J. Boote, E. Boyd, J. Durand, L. Kudarimoti, R. Lapacz, N. Simar, M. Swany, S. Trocha, and J. Zurawski, "Perfsonar: A service-oriented architecture for multi-domain network monitoring," in *Proceedings of 3rd International Conference on Service-Oriented Computing (ICSOC 2005)*. Amsterdam, The Netherlands: ACM, December 2006.
[3] "Network measurements working group (NMWG), Open Grid Forum," http://nmwg.internet2.edu/.
[4] M. Stokely, "FreeBSD Release Engineering," http://www.freebsd.org/-doc/en_US.ISO8859-1/articles/releng/article.html.
[5] "JUnit, Testing Resources for Extreme Programming," http://www.junit.org/index.htm.
[6] OGC (Office of Government Commerce), Ed., *Service Support*, ser. IT Infrastructure Library (ITIL). Norwich, UK: The Stationary Office, 2000.
[7] "Common Objectives for Information and related Technology (COBIT 4.0)," http://www.isaca.org/cobit.
[8] "Subversion (SVN)," http://subversion.tigris.org/, CollabNet.
[9] "Round robin database tool homepage," http://people.ee.ethz.ch/ oetiker/webtools/rrdtool/.
[10] "Apache Tomcat, Apache Jakarta project," http://jakarta.apache.org/tomcat/.
[11] "perfSONAR's Bugzilla installation," https://bugzilla.perfsonar.net/.
[12] "Simple Object Access Protocol, World Wide Web consortium," http://www.w3.org/2000/xp/Group.