ATLAS maintenance and operation management system

B Copy, M Tsikanin
IT Department, CERN, Switzerland

E-mail: brice.copy@cern.ch

Abstract. The maintenance and operation of the ATLAS detector will involve thousands of contributors from 170 physics institutes. Planning and coordinating the action of ATLAS members, ensuring their expertise is properly leveraged and that no parts of the detector are understaffed or overstaffed will be a challenging task. The ATLAS Maintenance and Operation application (referred to as Operation Task Planner inside the ATLAS experiment) offers a fluent web based interface that combines the flexibility and comfort of a desktop application, intuitive data visualization and navigation techniques, with a lightweight service oriented architecture. We will review the application, its usage within the ATLAS experiment, its underlying design and implementation.

1. Introduction

The ATLAS detector is the largest particle detector built to date and its upkeep will involve thousands of people from a hundred and seventy physics research institutes. The ATLAS management team has identified the need for a new information system to document, orchestrate and monitor the maintenance and operation of the ATLAS detector.

This document will expose both the requirements and the architecture that drove the implementation of the maintenance and operation management system referred within the ATLAS experiment as the Operation Task Planner tool (OTP).

1.1. Purpose and objectives

Continuing exploitation of the ATLAS detector will pose three considerable challenges which need to be addressed from the onset.

First, the maintenance and operation procedures of the detector need to be established and broken down into tasks according to one or more logical hierarchies, also referred to in project management speak as “breakdown structures”. In the case of the ATLAS detector, two breakdown structures have been retained : A Product Breakdown Structure (also referred to as ATLAS System/Sub-System hierarchy) will reflect the entire structure of the machine, while a Work Breakdown Structure will classify tasks according to the type of work they involve.

Once established, those maintenance and operation procedures are likely to evolve through time as the ATLAS detector's capabilities and configuration parameters are better understood and optimized. It is therefore important to document and provide details about those procedures, so as to be able to estimate the impact of those changes on personnel requirements.
Second, the work of hundreds of individuals will have to be precisely planned, hence ensuring that all maintenance and operation tasks are covered at all time by suitably experienced and knowledgeable staff.
Third, funding agencies and institutes sponsoring the construction of the ATLAS detector will have to contribute in fair amounts to the machine's maintenance and operation, which imposes strict and precise accounting of contributions and personnel availability periods.

The OTP system was designed to address those three challenges and will be used to document and coordinate the maintenance and operation work that will take place during the entire lifespan of the ATLAS detector.

1.2. Technical constraints and goals

The ATLAS OTP system is a web based application that will be used for intensive data entry by a large population of users. Every person that is to contribute to the ATLAS experiment is likely to use this system to enter their personal availability and specify their preferred shift participation schedule.
The ATLAS OTP system must be able to deal with fine grained scheduling information all the way to individual hours, while not imposing strenuous and repetitive data entry actions in case of recurring schedules. For instance, a person wishing to inform the system that they will cover a two hour shift every Wednesday for the next six months should not have to click twenty six times, but simply be able to communicate this recurrence like they would do in any calendaring application such as Microsoft Outlook or Yahoo Calendar (for instance, “every Wednesday and Thursday between the 1st October 2007 and 15 March 2008” or “every first Monday of the month until 31st December 2007”).
Task responsibles, i.e. individuals in charge of maintenance and operation tasks, should not be overwhelmed with individual schedules but be able to understand at a glance when task schedules happen to be understaffed or overstaffed.

We will first consider how the maintenance and operation of the ATLAS detector will be organized, before taking a look at how the system was implemented in support of this maintenance and operation organization.

2. Maintenance and operation in the ATLAS detector

Project management techniques have been used to globally organize the maintenance and operation procedures. In accordance with project management techniques, the ATLAS detector's maintenance and operation has been divided into tasks – coherent units of work under the responsibility of a single individual. Tasks are classified according to two hierarchies (or breakdown structures, in project management speak) which are the Work and System hierarchies.

1.1 Work hierarchy

The Work hierarchy is a Work Breakdown Structure (or WBS in project management speak) which classifies tasks according to the type of work involved (*e.g.* general management, calibration, data quality and monitoring...). The top entries of the WBS are referred to as "Activities". The ATLAS detector is currently comprised of four activities :

1. Computing / Software
2. Data preparation
3. Detector Operation
4. Trigger

1.2 System hierarchy

The system hierarchy reflects directly the systems that compose the ATLAS detector machinery. It can be seen as a product breakdown structure (or PBS), which is not entirely correct in the sense that it reflects an existing set up instead of a coherent set of deliverables. It is currently composed of ten systems :

1. General Tasks
2. DAQ specific
3. LVL1
4. ID gen
5. LAr
6. TILE
7. MUON
8. PIXEL
9. SCT
10. TRT

The ATLAS detector will be maintained and operated by individuals undertaking one of the four identified roles :

- Task Responsible : a person in charge of organizing the task, ensuring that manpower is available for all shifts and possesses an adequate level of experience and technical knowledge to fulfil the task.
- Expert : a person holding expertise in relation to a task's system and activity type
- Shift Manager : a person in charge of overseeing shift work, directing shifters and report on attendance and potential incidents.
- Shifter : a person performing the maintenance and operation work

For each task, personnel needs need to be formulated to specify how much manpower is required to fulfil the task. Such data can get rather complex and a large amount of accuracy and flexibility is necessary when expressing personnel needs.

Users of the ATLAS OTP system can specify personnel needs in three formats :

1. Simple format : for personnel needs that do not require precise scheduling, but rather a constant availability (for system experts for instance, which are not expected to be on site at all time of the detector operation, but should be easily reachable in case of problems) – in this

case, a simple start and end date is enough, signifying that the person is available at any time during that period.

2. Predefined format : for shift work that needs precise scheduling but follows a simple predictable structure, shift patterns are predefined in the system and can be readily used. In the case of the ATLAS detector, three predefined patterns have been isolated and are expected to represent a good ninety percent of all shift work specified in the system. Those shift patterns are :
    1. Single shift : one shift from 8am to 5pm requiring a single person.
    2. Day / Night shift : two shifts, running from 8am to 8pm and from 8pm to 8am, requiring a single person.
    3. Three shifts : splitting a 24 hour day evenly in three 8 hour long shifts requiring a single person.
3. Complex format : for shift work that requires the highest level of flexibility, end users are allowed to specify any shift pattern – how many shifts are needed, when does a shift start and end in the day, how many persons are required. They can also specify exactly which shifts are required through a familiar interactive UI (the shift editor shown in figure 1) which takes the shape of a calendar, either by toggling individual shifts, or by "drag-selecting" a set of shifts or by specifying a recurrence (*e.g.* every week day between the 1st January 2008 and the 31st March 2012).

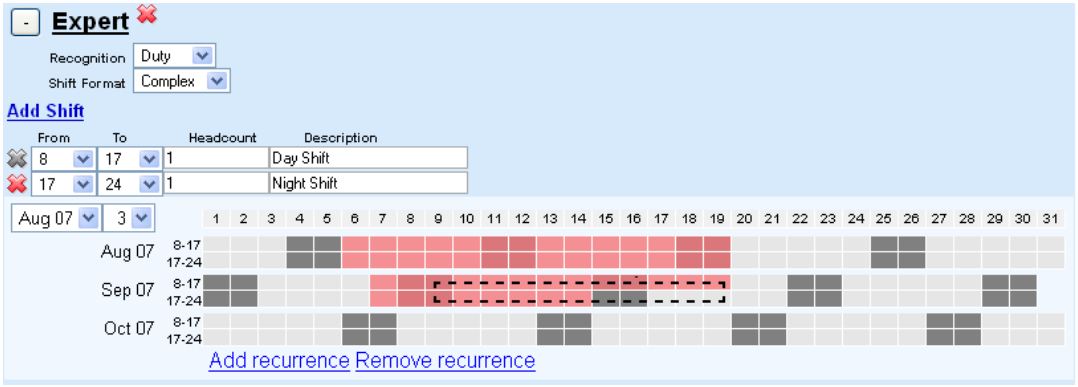

Figure 1: Complex personnel needs data entry

Finally, once the manpower requirements have been expressed, individuals can be identified to fulfil those requirements. Once again, the system use a highly interactive UI widget, the shift editor, to let users specify precisely when a given individual will be performing maintenance and operation work.

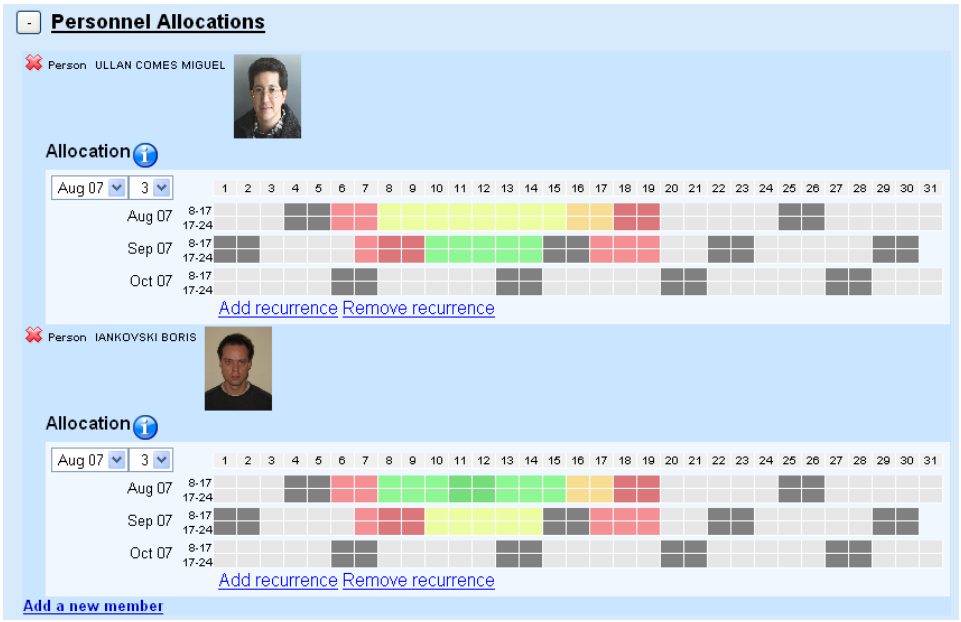

Figure 2: Personnel allocations, with allocation conflict support

Once the tasks have been created and classified in our WBS and system hierarchies, the personnel needs expressed and the individual workers identified, it is possible to answer questions essential to ATLAS management and support teams such as :

- Which tasks are currently understaffed or overstaffed ?
- Which WBS or system are consuming the most manpower ?
- Who is currently acting expert on a given system ?
- What is the contribution of a given individual or institute to the ATLAS detector maintenance and operation during a given time period ?

Figure 3 below illustrate a pivot table that displays tasks classified by their parent WBS and System category.

Figure 3: WBS / System task planning browser

3. Design Approach

From the onset, the ATLAS OTP system was split into two main components, a server side service and a client side, browser based, rich user interface (UI). A new emerging analysis method, known as service oriented design, was used to combine more traditional design methods such as model oriented analysis and user interface design.

3.1. Service Oriented Design

Service oriented design is a design approach that is directly inspired from scientific disciplines that focus on human interactions, such as social sciences, behavioural sciences or even product marketing. It identifies two essential interacting entities, a service provider and a service consumer, and attempts to understand how to make their interactions effective.

Web services, the most recent and prominent implementation of service oriented design, has received a large amount of attention in recent years, as the need for highly distributed, heterogeneous implementations of large scale information systems has emerged. Web service technologies such as SOAP [1] build upon previously existing distributed system architectures such as Sun RPC or OMG CORBA, but does away with the type of binary coupling you find in those architectures by leveraging XML based standards such as XML Schema (XSD) and Web Service Definition Language (WSDL).

Web services in their essence are not new, and the implementation challenges they have to face are exactly identical with the ones found in other distributed architectures like CORBA or Java Enterprise Java Beans (Java EJB) : distributed transaction management, resource location and management, authentication and access control propagation and federation, asynchronous and synchronous messaging etc...

Thanks to their lack of binary coupling and the support of lightweight protocols such as HTTP, web services are also more inclined to deliver :

- Lower coupling between the various services, since web service clients can be generated on the fly and are supported by a large variety of platforms, including a large array of high level scripting languages such as the Python or Ruby languages.
- Eased maintenance whereby service provider instances can be replaced, tested, upgraded or moved without necessarily affecting other collaborating providers.
- Clearly defined system boundaries, as service consumers are necessarily accessing the service through well known service interface definitions.

Most modern distributed architectures such as CORBA or Java EJB already deliver some of these characteristics to some degree but binary independence and lightweight protocol support were afterthoughts and not identified as key features from their onset. While such challenges remain important, it would be a mistake to focus solely on those issues and facilities when designing a service oriented system destined to support a computer based user interface. Service oriented design places the focus on what an end-user facing information system is meant to achieve : smooth, productive and fulfilling interactions.

Robert Glushko in [2] provides the example of a hotel check-in (a typical kind of customer / service interaction sequence many of us are familiar with) and uses the various types of hotel check in procedures to illustrate his conception of a quality of service, when relating to customer / service interactions. For instance, he compares an automated check in (where the hotel customer is identified by their credit card and given a random room number and card key) with a luxury hotel check in (where the hotel customer is identified by their name and full title and their preferences in terms of room location, culinary tastes and opera seating location are remembered and leveraged to provide a personalized and gratifying customer experience). The same information system might be supporting both implementations of a hotel check-in service, but the latter of these implementations proves clearly superior in the eyes of the customer, simply in the way the information it can hold is employed to the customer's benefit.

Glushko [2] lists the following characteristics when evaluating a service :

- Service quality : the difference between the level of service that the customer expected and the level of service she received.
- Service intensity : the number of actions and the duration of the customer transaction – for instance the amount of information a customer must provide when checking into a hotel. The amount of information requested by a service at a given time must be appropriate and worth giving out in return for what the service can provide.
- Service variability : the flexibility of a service, how easily it can be customized to suit a person's specific needs and cope with unusual or fringe business cases while remaining predictable in the results it delivers.

None of those characteristics warrant a suitable and successful service implementation by themselves, they need to be combined and assessed carefully in order to provide what can be qualified by the customer as a good service.

Let us now consider how the service back end and front end parts of the ATLAS OTP system were implemented.

4. Back end implementation

Glushko [2] derives the perceived characteristics of a good quality service and recommends therefore to focus on the service definition, or service interface, and how to define a high quality service interface from which the domain model, relational storage structure and implementation details will naturally emerge.

All necessary means, ranging from use case diagrams [3], through mock screenshots to domain oriented analysis [4] can be employed to refine the service interface and those means were all used during the conception of the ATLAS OTP system.

UML use case diagrams were first drawn to refine the understanding of the system's primary goals and actors. Mock user interface screenshots were also produced and validated together with the project stakeholders in order to drive the overall service definition.

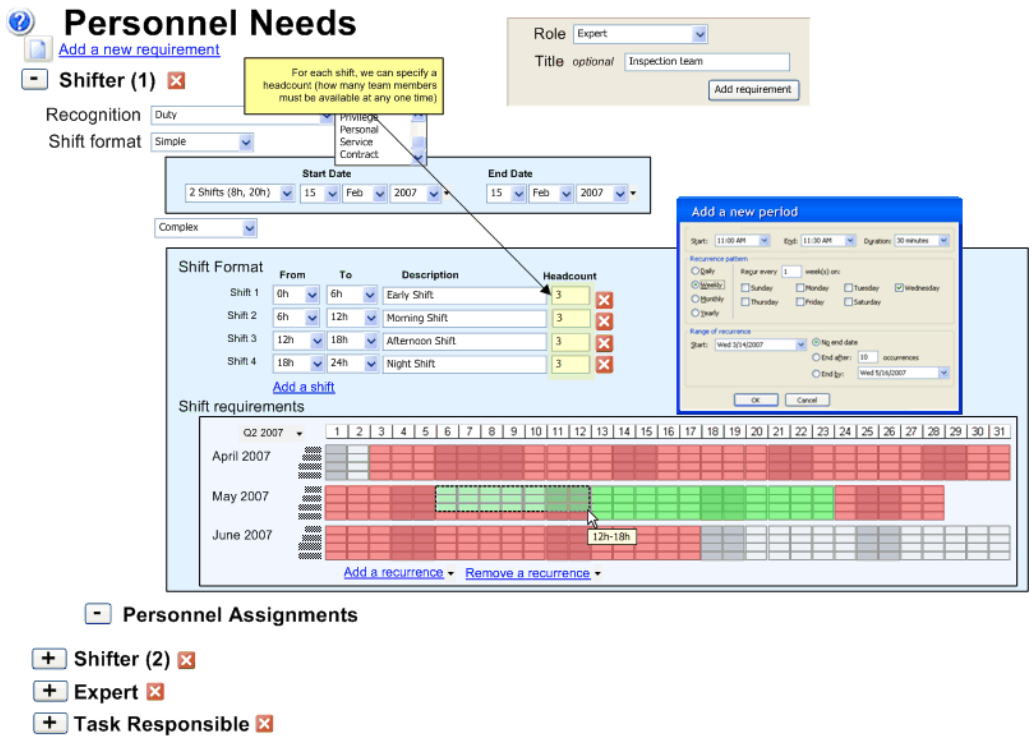

Figure 4: Mock screenshot of the personnel need specification screen

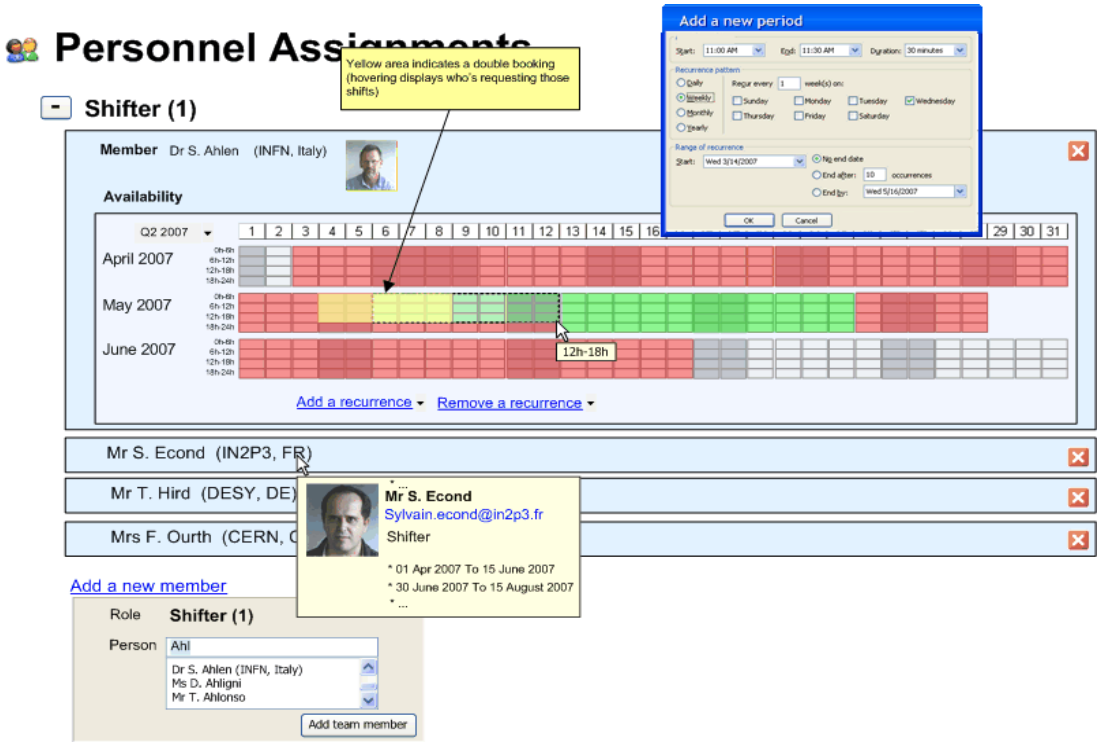

Figure 5: Mock screenshot of the personnel allocation screen

The service definition was refined in order to support the types of server interactions that would be required by the various screenshots.

Finally, the service definition was used as a template to produce back end components, such as domain objects, a relational database based storage layer, as well as enforce transactional boundaries and access control constraints. The Spring framework [5] provided invaluable support towards integrating those various components and ensuring a clean, interface centric implementation was produced. Figure 6 below presents an overview of the implementation and highlights the proprietary and standard components of the system.

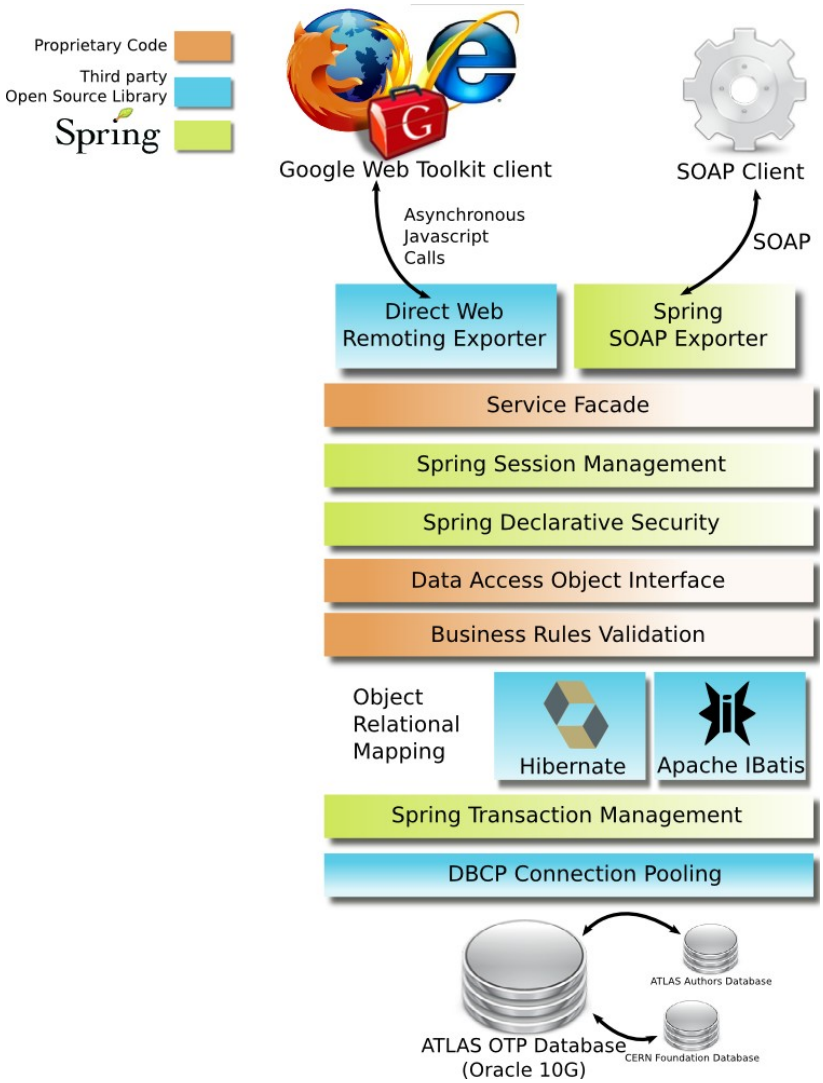

Figure 6: ATLAS OTP system implementation overview

As explained in previous sections of this document. service oriented design provides an excellent lifeline to guide a development team through the intricacies of a service oriented implementation.
Once implemented with support from the Spring framework, it becomes trivial to expose your service implementation through a remoting protocol. Since our target platform was web browsers that support the Javascript scripting language, the Direct Web Remoting (DWR) [6] protocol was selected and we will see in detail its advantages over other remoting protocols when it comes to service remoting.
On the client side however, there is at the moment a crucial lack of service aware UI development tools, which forces developers to match this gap with existing tools. To match the richness of our ATLAS OTP service, we needed to develop a rich user interface that would allow a high level of interaction, and limit its server access requirements to well defined service calls, to avoid placing high and constant load on the application server itself. The Google Web Toolkit [7] (GWT) was selected to fulfil this goal and we will consider in details what this toolkit provides over other web user interface solutions.

Let us first consider our usage of the DWR protocol, then focus on the Google Web Toolkit implementation of the user interface.

4.1. Client side service access with Direct Web Remoting

Services are ideal targets for multiple operating environments : a service interface once defined and implemented can be made available at a very low cost to a large variety of remote clients, using any remoting technology available at the time, such as SOAP [1], Java Remote Method Invocation (RMI) or REST [8] oriented APIs. DWR [6] is yet another open source remoting technology, which targets specifically web browsers.

DWR provides essentially serialization support between a Javascript client and a Java server. Data such as object parameters or method call return values are seamlessly converted from Javascript object structures to plain old Java objects (POJO) structures and vice versa. Any Javascript and HTTP capable client can access a Java based service implementation without any need for pre-compiled stubs or object data type descriptors. This lightness and flexibility comes of course at a price, namely that the Javascript code that will leverage this service will be weakly typed – there is in effect no way to assert through compilation that a given Javascript client implementation will be compatible with a given service implementation. In practice, all Javascript client implementations suffer from this limitation and DWR makes no exception to this rule. The Google Web Toolkit (GWT) provides an elegant solution to this problem by integrated support for unit and functional testing.

4.2. Google Web Toolkit

The Google Web Toolkit (GWT) allows to write web user interfaces using the Java programming language. It lets developers benefit from a familiar development environment and advanced coding tools such as debugging, profiling and refactoring support. Prior to deployment, the GWT Java code is compiled into cross browser Javascript code which is certified by the Google Web Toolkit team to run on a large collection of browsers and operating systems, from Safari on Mac OSX to Internet Explorer 7 on Windows through Mozilla Firefox on Linux machines.

Writing a UI with GWT is extremely similar to writing a desktop based UI with a native toolkit such as Microsoft Foundation Classes or Java Swing UI. Widgets on the screen communicate through familiar UI design patterns such as listener/observer, model/view/controller, command, chain of responsibility and factory patterns.

Once compiled into Javascript, a GWT application integrates seamlessly into a browser environment and an existing web site, thus avoiding the need for non standard plugins and non standard browser interactions. Other UI development targets, such Adobe Flash or Microsoft Silverlight [9] restrict user interactions with an HTML document to a strictly defined rectangle on the screen and a restricted set of APIs. GWT applications on the other hand are fully part of the HTML document, integrate with the Document Object Model (DOM) and even with third-party Javascript code.

GWT reinforces a weakly typed language such as Javascript with a welcome amount of formal support provided by the Java language. It supports functional and unit testing out of the box for two web browsers (Internet Explorer and Mozilla Firefox) which allows to ascertain a client's compatibility with a given service implementation in an automated manner, thus supporting continuous integration and test driven development.

5. Conclusion

The ATLAS OTP system is already available in production to plan the detector's maintenance and operation. Its service oriented design, sound and well structured implementation backed by automated unit and functional tests provide a solid basis for further developments. A certain amount of effort remains to provide a perfectly fluent user experience and customize the system to other shift related work, such as CMS detector maintenance.

6. References

1: World Wide Web consortium (W3C), *Simple Object Access Protocol (SOAP) specifications*, 2000, http://www.w3.org/TR/soap/

2: R Glushko, *Bridging the "Front Stage" and "Back Stage" in Service System Design*, 2007

3: M Fowler, *UML Distilled*, 2003, Addison-Wesley Professional

4: E Evans, *Domain-Driven Design*, 2003, Addison-Wesley Professional

5: R Johnson, J Hoeller, *Expert One-on-One J2EE Development without EJB*, 2004, Wrox

6: J Walker, *Direct Web Remoting documentation*, 2003, http://getahead.org/dwr

7: B Johnson, *Google Web Toolkit*, 2005, http://code.google.com/webtoolkit

8: R Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, 2000

9: Microsoft Corporation, *Silverlight FAQ page*, 2007, http://www.microsoft.com/silverlight/faq.aspx