hifly anywhere remote web operations

T. López-Ciudad[1] and E. Fraga.[2]
GMV. Isaac Newton 11, Tres Cantos, 28760, Spain

Remote satellite operations have been a hot topic during the last years. Recently the ubiquity of Internet access and massive adoption of new mobile devices permanently connected has brought new options into place: the possibility of monitoring and operating satellites at any moment from any place. We discuss the approach followed to gain ubiquitous monitoring of a satellite. Those efforts have crystalized in *hifly anywher*e , the remote access layer to *hifly* GMV's multi-mission satellite control center. Although a real time system provides a wide range of features, a reduced set of capabilities may be advised to be provided remotely in order to keep a clear and secure interface. Service oriented architecture (SOA) for the actual user interface of the real time system grants the capability to expose those features required through a SOAP layer, while a SOA design driven by RESTful services provides the needed security and control. In this way remote access to the control system is always possible. On the other hand technologies like Google Web Toolkit (GWT) provide the creation of high performance, complex and safe browser applications with synchronous and asynchronous data access. At the same time GWT is equipped with a mature and effective development environment, which enables a maintainable and scalable code. Despite the benefits of GWT, native applications need to be developed for mobile OS's to enhance both, reaction times and user experience. The technologies used allow complete reuse of business logic based on the very same SOA layer and upgrading only the human-machine interface.

I. Introduction

hifly is a multi-mission satellite control center (SCC). *hifly* evolved from an old ESA's SCOS-2000[3] version, where HMI interaction had a monolithic design and hence presentation layer and client business logic were coupled. Both *hifly* and current SCOS-2000 versions has been moving on the direction of enhancing the HMI for client applications using the Eclipse RCP framework[4]. In *hifly* one of the main drivers for the architecture design is scalability and to allow data consumption from third party applications. In order to achieve this scalability and flexibility and to add support for data consumption four steps were required:

1. Split the presentation layer from the business logic.
2. Use a Service Oriented Architecture (SOA) for implementing the business logic.
3. Code client applications in Eclipse RCP.
4. Create a Simple Object Access Protocol (SOAP[5]) layer that uses *hifly* services to access satellite data

Satellite operations are traditionally performed on-site with additional support from an on-call team. However, when a problem occurs in the control room a contingency mechanism is triggered which usually includes phone calls to engineers, software support and other experts. This activity has a big impact on people supporting satellite operations. Moreover, tackling the origin of the problem by phone without having access to the actual facilities is difficult and many times moving to the SCC facilities is needed where usually with a simple look to the appropriate telemetry data or to the messages from the system would allow understanding the problem and maybe even

[1] GMV, Ground Systems Engineer, Satellite & Mission Control Aerospace. tlopez@gmv.com.
[2]GMV, Head of Satellite & Mission Control Aerospace. efraga@gmv.com.
[3] See http://www.egos.esa.int/portal/egos-web/products/MCS/SCOS2000/
[4] See http://wiki.eclipse.org/index.php/Rich_Client_Platform
[5] See http://www.w3.org/TR/ws-gloss/. SOAP specification is at http://www.w3.org/TR/#tr_SOAP.

providing a remote solution to it. Nowadays, the wide availability of internet access and powerful mobile devices can solve these issues allowing an immediate access to satellite or SCC status and data.
Granting remote secure access to satellite system requires two kinds of interactions with the user:

- A Web layer accessible from any browser. Web access solves the problem of accessing from any computer to satellite data.
- Mobile native applications. Designed with clarity and ease of use in mind. Native applications reduce reaction times and opens up a huge set of new possibilities with limited work.

The efforts for giving remote access to the system lead to *hifly anywhere*, the remote access layer to *hifly* GMV's SCC.
The paper is organized as follows:
First, we will explain the efforts made and the rationale for the remote access layer to *hifly*. Next the internal architectural changes performed to ensure security and access to third party applications is described. An explanation of the technology used for Web access presentation is then introduced. The last point is the discussion of the extension to mobile native environments. Throughout the paper main steps and lessons learned will be described.

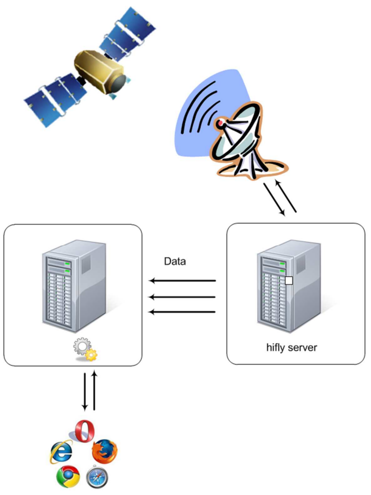

Figure 1. *hifly anywhere* deployment architecture. *Actual data from the SCC is forwarder to another server that provides services for remote access.*

II. Remote access using web services

The initial need of implementing remote Web services came from the fact that any portable system needs to accomplish the following requirements:

- Independence of the operative system used for accessing the data
- Reduce deployment to a minimum
- Remove COTS
- Speed up development
- Security
- Control of the data

A remote access system that needs deployment and installation is not effective as implies specific software installed on the remote machine, which is usually not available. Fortunately nowadays there is a widely used solution for this problem: Web services. Web services only require a browser for accessing the required data. In addition, with a complete set of web services, applications can be quickly customized modifying only the presentation layer. Another important advantages of Web services is that customized client implementation can use secure architectures and operate correctly with firewalls.
Not all satellite data has the same priority for remote operations. Critical events, out of limits alarms and telemetry visualization are the minimum set of data needed for monitoring satellite behavior and give proper support in case of contingency. Historic telecommand data sent to the satellite is also desirable when remotely supporting operations.
The following services have been implemented as services for external data consumption:

- Configuration data of the actual SCC.
- Raw data retrieval (TM, TC and Events)
- Telemetry (both real time and historical)
- Out-of-limits information (both real time and historical)
- Telecommand (both real time and historical)
- Events (both real time and historical)

We have not implemented services that could interfere with the actual real time system controlling the satellite to avoid interference with on site operations. Note that remote Web services, implies big network latencies and asynchronous work of many people simultaneously. Active modifications (remote database changes, telecommand uplink, limit definition modifications…) to the actual system could lead to undesired situations. Deep inside this philosophy of no interference with operations, the deployment of *hifly anywhere* is devised to only consume data from the SCC but not interfere with it. In Figure 1, we present the *hifly anywhere* deployment. Satellite data is processed by *hifly* servers. This data is then forwarded to a *hifly anywhere* server which is the one accessed from external applications or web browsers. Note that *hifly anaywhere* only receives data from the SCC but not the other way around.

III. Service Oriented Architecture

The main challenge when implementing remote services is how to split the presentation layer from the business layer. Usually both layers are tightly coupled but data consumers need a service-based interface. In this context a service *is a set of components which can be invoked and whose interface description can be published and discovered*[6]. By means of a SOA we obtain the following benefits:

- Well-formedness. Well-formed services provides with a unit of management that relates to business usage.
- Services are abstracted from implementation. This simple fact makes new alternative options for delivery and collaboration models.
- Services provides a real synchronization between business layer and implementation.

hifly SOA implementation is based on a component based architecture which provides a one to one mapping between business entities and component implementation. This concept is depicted in
Figure 2. Any application, using the service interface (service contract) only exchange data with *hifly* services layer using CORBA[7]. The services layer, acting as a façade, translate queries and gather data from *hifly* server, which is receiving data from the satellite fleet.

Services need to scale to meet high performance demands. For instance, clusters of servers with load-balancing and failover capabilities often forward requests to other servers to reduce load or reduce overall response time. This requires clients to send complete, independent requests, which include all the data needed to finish the operation. For this matter, *hifly* services have been implemented stateless. This way the server can forward, reroute or load-balance queries without saving locally any internal state. They do not require, when processing requests, to retrieve any kind of application context. This fact allows a reuse of the *hifly* services for providing Web services using SOAP that will be used for HTTP access to SCC data.

IV. Simple Object Access Protocol

Many applications communicate today using Remote Procedure Calls (RPC) between objects like DCOM and CORBA. RPC presents compatibility and security problems and firewalls will normally block this traffic. Although CORBA is intensively used by *hifly* for interprocess communication, CORBA objects have a complex lifecycle and server implementation. In addition CORBA does not suit well for communications with external entities and have mobile environment constrains such as changing and unreliable network addresses. A better way to obtain data from the SCC is to use HTTP. HTTP is supported by all Internet browsers and supports firewalls and secure communications. On top of HTTP we need another application layer to communicate efficiently between applications. SOAP, a XML-based protocol for exchanging structured information in distributed

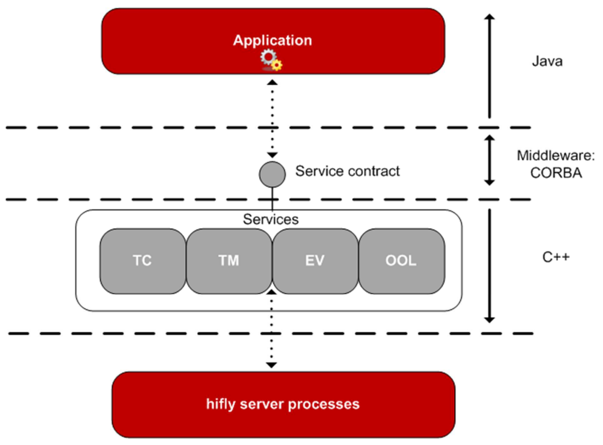

Figure 2. *hifly* SOA layer. *hifly's SOA implementation design. Desktop java applications only exchange data with the services layer.*

[6] See "service-oriented architecture" at http://www.w3.org/TR/ws-gloss/.
[7] See CORBA OMG web site at http://www.corba.org.

environments, is the W3C standardized way of implementing Web services. *hifly anywhere* provides a Web service layer through SOAP for accessing the core services. These services use HTTP/S as transport protocol.
The major benefits of adopting SOAP are:

- Extensibility. Security and encryption is mandatory for accessing satellite data. These extensions are widely used with SOAP.
- Neutrality. It's not linked to a protocol, although HTTP is commonly used.
- Platform and language independency.
- Widely used by the industry.
- XML-based: which makes it operable among a wide range of different applications.
- Use of the HTTP get/response model. This way SOAP tunnels easily through proxys and firewalls.

The main drawback of SOAP is that it needs processing potentially large XML chunks of data and can be slower than competing technologies such as CORBA. However, for small messages this is not an issue. In addition SOAP is considerably more complex that alternative protocols as XML-RPC[8] , but is more flexible.
With SOAP a Web service is documented as a Web Service Description (WSD), which is a specification of the web service interface written in Web Services Description Language (WSDL). It defines a set of endpoints, functionalities exposed, message formats, transport protocols, data types, and transport serialization formats that should be used in order to communicate with that Web service.
Other systems interact with the Web service in a manner prescribed by its description using messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. In essence, the service description (WSDL) represents a contract governing the mechanics of interacting with that service.
Due to security, all Web services are authenticated so credentials have to be provided in order to gain access to the service. Web services authentication is accomplished by using the WS-Security standard[9].
Figure 3 shows the *hifly anywhere* logical model. Third party applications use SOAP to exchange data with the *hifly anywhere* server that implements an authentication layer for accessing the different web services (Core Services). Web services translate SOAP requests to the *hifly* services through a set of libraries (*hifly* connector). In this way the complexity of communication with the *hifly* server is encapsulated. A key feature of this model is that each user (either "WS Client" or "Browsers") may log in to the *hifly anywhere* system with their own user/password so that the authorization to access each individual service at *hifly* is granted/denied by the authentication and authorization module.

V. Presentation Layer

Once the services are implemented we need to present the user a clean interface for consuming the data. There are many possibilities for this purpose: Silverlight, AJAX, Adobe AIR, etc. Adobe AIR and Silverlight have been considered as options. Both performed surprisingly well, however lack of availability in all browsers (mainly in mobile devices) and usage trends made us choose AJAX as preferred option for the presentation layer.

Each time traditional web applications talk to the web server, they fetch a completely new HTML page. In contrast, Asynchronous JavaScript and XML (AJAX) applications offload the user interface logic to the client and make asynchronous calls to the server to send and receive only the data itself. This allows the HMI of an AJAX application to be much more responsive and fluid while reducing an application's bandwidth requirements and the load on the server.
Asynchronous calls are a core principle of AJAX development. The benefits of making asynchronous calls rather than simpler (for the developer) synchronous calls are in the improved end-user experience:

- The user interface remains responsive. The JavaScript engines in web browsers are generally single-threaded, so calling a server synchronously causes the web page to "hang" until the call completes. If the network is slow or the server is unresponsive, this could ruin the end-user experience.
- We can perform other work while waiting on a pending server call. For example, we can build up our user interface while simultaneously retrieving the data from the server to populate the interface. This shortens the overall time it takes for the user to see the data on the page.
- We can make multiple server calls at the same time. However, just how much we can add using asynchronous calls is limited because browsers typically restrict the number of outgoing network connections to two at a time. However this limit can be changed by configuration.

[8] See http://xmlrpc.scripting.com/default.html
[9] OMG WS-Security 1.1, 1 February , 2006. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

- The number of concurrent connections of our browser limits the number of hifly anywhere simultaneously opened applications a client can have in the following way: each login on hifly anywhere opens a connection to the remote server.

The problem with AJAX is that it is based on an intensive use of JavaScript, which is difficult to maintain and requires merging HTML code with JavaScript code. In addition debugging code in JavaScript is pretty hard. The goal would be to work in a pure high level object oriented language and generate dynamic pages. Fortunately there is a set of tools that allows web page creation and maintenance just programing in Java. This set of tools is Google Web Toolkit (GWT[10]). GWT emphasizes reusable and efficient solutions for complex pages, including remote procedure calls, history management, bookmarks, internationalization and cross browser compatibilities. In addition GWT allows inclusion of native JavaScript libraries so we can merge the best of JavaScript and Java realms. GWT goal is to enable productive development of high-performance web applications without the developer having to be an expert in browser quirks, XMLHttpRequest, and JavaScript.
Each GWT project consists of one or more GWT modules, each of which can be made of up one or more entry points which are basically a JavaScript file generated by GWT compiler including the whole user interface, events handling and more. Once a GWT module is loaded all of its entry points are loaded. Shortly, user interacts with the UI the browser has rendered from the JavaScript file. Then, the entry point logic is responsible to handle all the events and, if necessary, communicate with the server sending requests and receiving updates.
A simplistic vision of Web client shows an application that needs to refresh Event data and TM data from a data source (satellites). This data could be retrieved by time triggered polling to the *hifly* anywhere server. This triggered poll reduces client performance considerably. Due to HTTP push, the *hifly anywhere* client can retrieve satellite data without any polling mechanism. The GWTEventService[11] establishes a synchronization mechanism to push new events from server to client. This approach has shown improved efficiency compared to the polling mechanism.
In addition to the advantages mentioned, GWT make intensive use of Cascading Style Sheets (CSS). CSS styles allow straightforward customization of web pages for mobile environments.

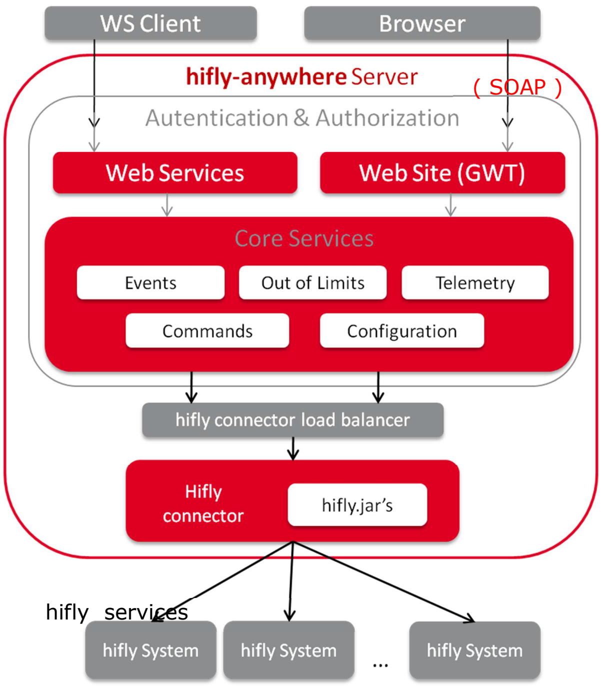

Figure 3. *hifly anywhere* SOAP services logical model

VI. Native applications on mobile devices

Although Web applications have many advantages, native applications are the preferred way in terms of usability, speed and user experience. Nowadays, new advancements on JavaScript engines make web applications closer to native applications. However despite the many benefits of web applications, there are some functionality only accessible to native applications. Synchronous notifications, access to adressbook or device GPS are examples of functionality only allowed natively. The SOAP architecture makes the implementation of the native application pretty easy as the same services used for the web application can be used here.

In this sense *hifly anywhere* has made a big effort to include native Android and iOS applications. The idea is creating very specific applications focused on clarity, speed and ease of use. The main idea is to make use of phone notifications to take action. An example of this is to create an application which pops up an alert and makes the phone beep whenever a critical event occurs in the SCC. Clicking on the notification will open a dedicated native

[10] See https://developers.google.com/web-toolkit/overview
[11] See http://code.google.com/p/gwteventservice/

application with the latest events occurred in the system. From this point the user just clicking one button can make a phone call to the operations room, or retrieve more data.

This kind of applications can complement the web applications and can be easily customized for each user.

VII. Conclusion

We have explained the path that leaded to *hifly anywhere*, the remote access layer for *hifly*, GMV's multi-mission satellite control center. To reach a point where third party applications can consume data in a proper way the internal architecture of the kernel has to be translated into a Service Oriented Architecture. But SOA alone is not enough for exchanging data using web services. For this purpose a SOAP bridge has been added to hide the complexity of the calls to the satellite control center. SOAP enables a new set of applications that can be used at any moment anywhere. Moreover SOAP can be used to enhance usability and speed using native applications. With *hifly anywhere* following operations remotely is much easier and comfortable as only a mobile device or a browser is needed.

VIII. Appendix A
Acronym List

EV	=	Events
HMI	=	Human Machine Interface
OOL	=	Out Of Limits
RPC	=	Remote Procedure Calls
SCC	=	Satellite Control Center
SOA	=	Service Oriented Architecture
SOAP	=	Simple Object Access Protocol
TC	=	Telecommand
TM	=	Telemetry
WSDL	=	Web Services description Language

I. Acknowledgments

This work has been supported partially by ESA contract number 4000100801. *Total Multimission Operations* (TMO) ARTES 3-4 project.

We wish to thank J.C. Gil Montoro and J. Santana for their suggestions and insightful discussions.