# Canarie IIP-03 “Undersea Window” Project Milestone 1 Report
# Appendix 2
# Report on the Web Services Software Development Plan

**Jeremy Cooperstock and Ilia Soukhodolski**

The architectural and functional design of the Web Services application was completed in the first phase of the project. All major application components were identified and described, functional diagrams built and both Operating System and development platform selected.  The application is now being developed under Linux OS, driven by the Apache Web Server and MySQL database. Individual components are being implemented with Java, PHP and C programming languages. The required Web Services were classified into the three classes: user, administration and hardware requests.  The general definitions for these classes have all been constructed.

It was decided that an XML-based SOAP-wrapped communication protocol will be used as a communication layer between all connected application components, making it easier to interconnect with related applications (e.g. DMAS).  On the client side, a proprietary Web Services Java applet running within the web browser will provide general access to the application functionality. An uncompressed HD video stream will be provided by our proprietary /bronto/ transport, controlled by Web Services. Given the direct interaction with graphics and network interface hardware on the client, as necessary for our software, the uncompressed HD stream can only be provided to Linux-based workstations at present.

A series of performance and application response tests were conducted in simulation as part of our OS and development components selection. These tests indicate more than adequate performance and an estimated response time that allows adequate overhead for data exchange with connected hardware (e.g. the HD camera and its control mechanisms).

Further details regarding the architectural and functional design can be found below.

## 1. Types of Requests

Our design is based on the division of requests to the web server into the following three categories:

1. Hardware requests (H.Req): one-way requests (no response from the server) produced by the intelligent control units of the HD camera and possibly optional hardware units. These requests will be driven by selected hardware events and will be responsible for delivering real-time information from the device to the Web Server.

2. Administrator requests (A.Req): two-way requests that allow for control and set-up of connected hardware, operation of the application flow and administration of end-users profiles, priorities and actions.

3. User requests (U.Req): two-way requests that provide access to camera controls, system access reservations, information flow and event-driven batch processes.

## 2. Hardware Control Units / Processes and HAL

Every device in the application scope will have its own control process running. The purpose of these processes is to create a hardware abstraction layer (HAL) between the Web Server and the actual physical device, serving as an intelligent device driver. These processes may run either on the same machine as the Web Server, or on separate dedicated computers, but most likely, a combination of both, depending on network topology, communication speed and performance requirements.

## 3. Functional Diagram

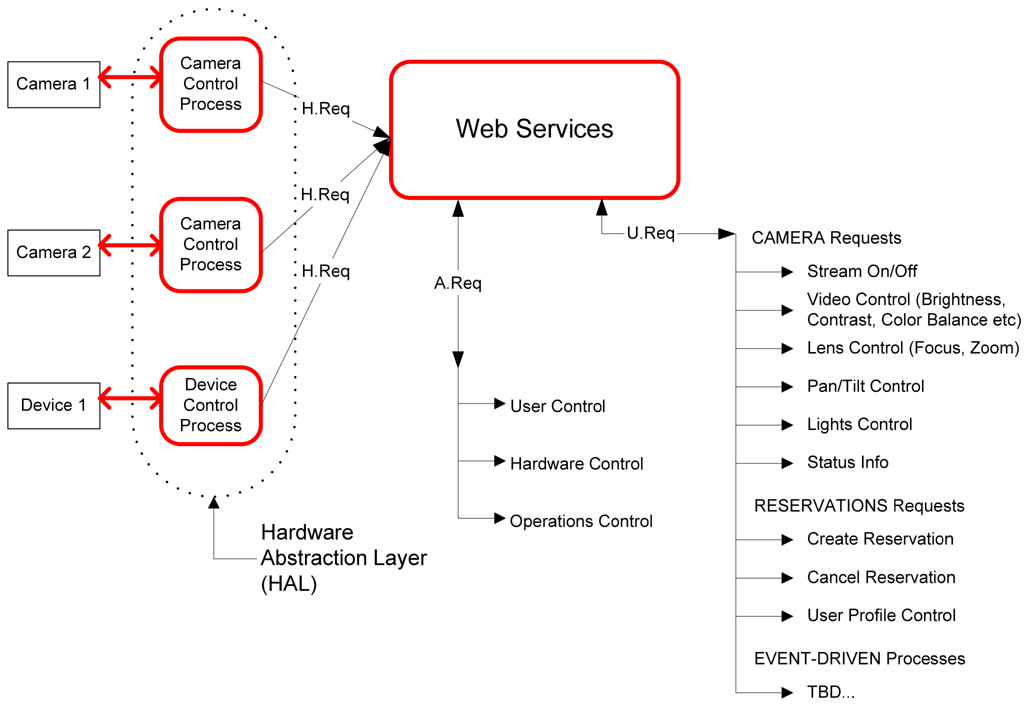

Fig. 1a. Functional diagram

## 4. Collaboration with Neptune Canada

The Neptune Canada project architecture is based on the Enterprise Service Bus (ESB) concept, built on top of the IBM Websphere MQ platform. The general approach of providing XML-based Web Services as an interface to the instruments, as well as access points to the measurements database and business applications is similar to our project. The access to the instrument hardware control will be provided by means of a “rich client”, similar to our concept of “Device Process”. However, the much larger scope of the Neptune project dictates a larger application scale. It includes not only telemetry reading and sensor control, but also distributed signal processing and data management services combined under Data Management and Archiving System “DMAS”.

The Undersea Window project will likely use DMAS services in terms of user accounts management and mass data storage (e.g. for keeping records of HD video stream). The collaboration will be achieved by means of loose-coupling of our Web Services to the Neptune ESB:

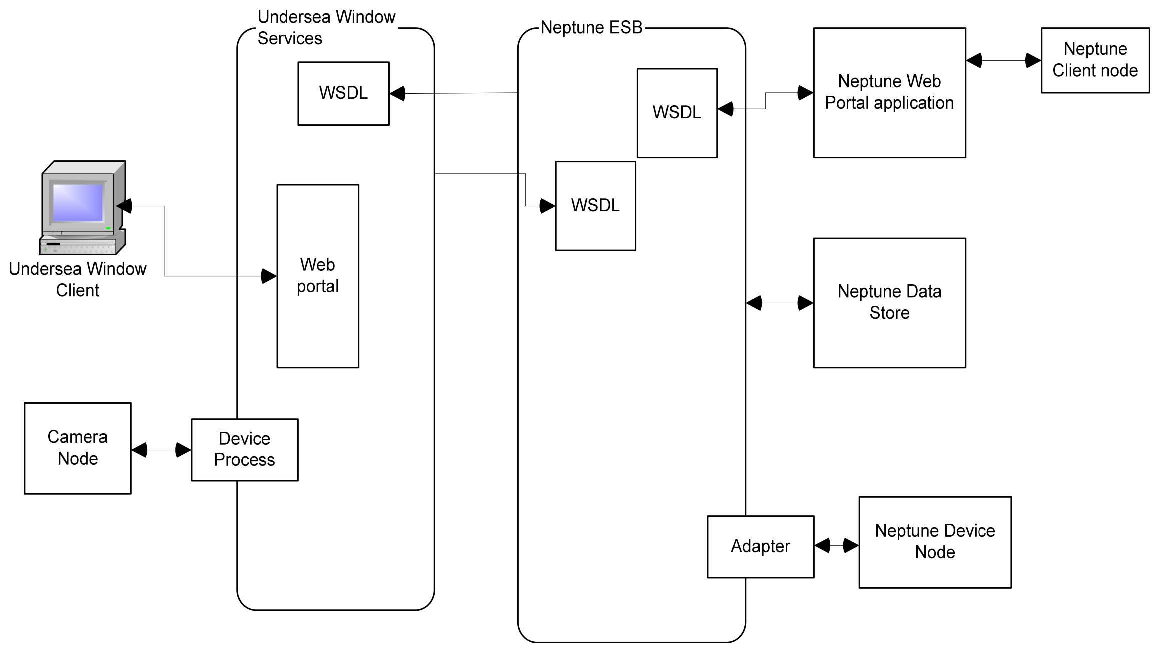

Fig 1b. Collaboration with Neptune ESB

## 5. Project Environment

The project environment will be based on the following components:

- Linux OS
- Web Server
- Script language with Soap functions library
- Database

The detailed functional architecture as shown in Fig. 2, is explained below:

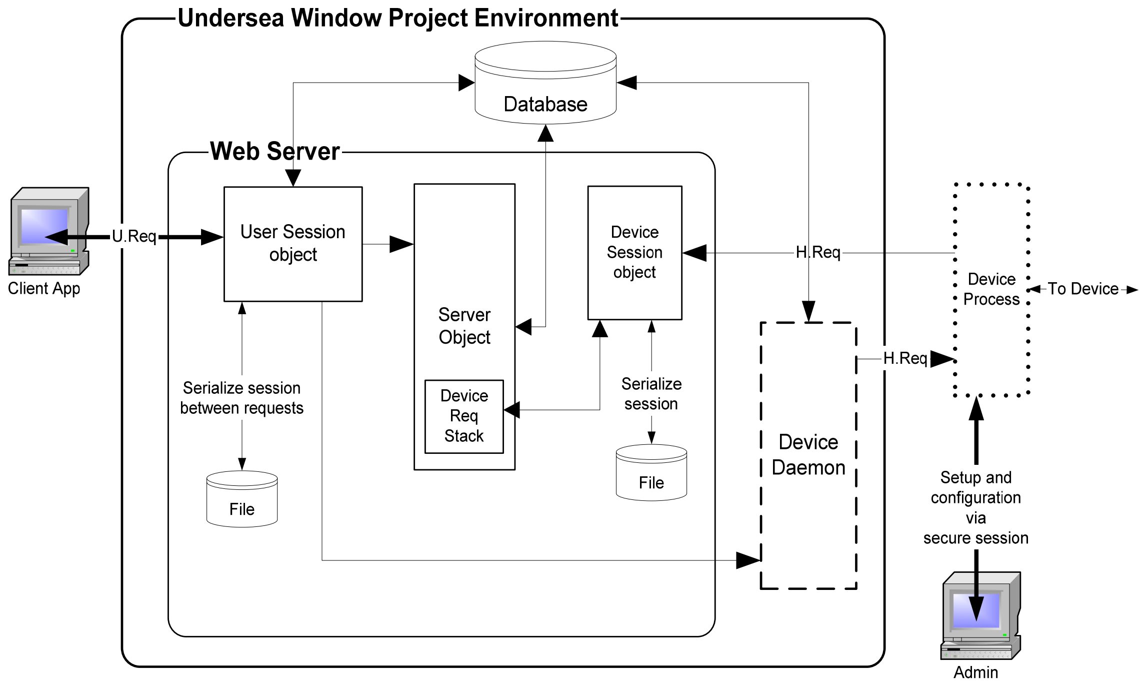

Fig. 2. Project Environment diagram

The Web server will receive SOAP requests from the users, process the requests, record request parameters into the database and pass the information to another process called the Device Daemon.

The Device Daemon will run in the same environment as the Web Server and maintain persistent connections with Device Processes. It will receive requests from user sessions and pass these for execution to the Device Process. The need for a Device Daemon is simple – we cannot maintain persistent connections (and therefore control the execution of the requests) from user sessions, because of their temporary nature.

Each Device Process will act as both client and server. It will listen to the Device Daemon requests to open a device control session, translate high-level commands sent by the Device Daemon into the device hardware control messages and report device status by submitting SOAP requests back to the Web Server. There may be any number of device processes connected, although the diagram only includes one for illustrative purposes.

The database will contain the system administration information (list of users, permissions, reservations etc), hardware configuration and current device status information, all the submitted requests, their status and execution progress.

## 6. Controlling device hardware

The Device Process will operate device hardware. Although there will be one logical device (Camera Node) per Device Process connected, it may consist of several separate physical components. For example, the pan/tilt unit is likely to have a different control interface from the zoom and other camera functions.  Similarly, control of the light source will be performed from a separate physical unit.

The role of the Device Process is to receive control requests from the Device Daemon using the HTTP server and translating these into actual hardware signals by means of the Hardware Manager. The Hardware Manager will send hardware control signals over corresponding interfaces (e.g. serial RS-xxx) to perform requested hardware actions (as shown in Fig. 3, below).

The video feed will be provided by a proprietary software component, Video Manager. After passing all necessary authentication steps, the user will be given a link to the video stream transported over the CA*net4 network. User-Controlled Light Path technology will be utilized where feasible for transport of the video stream, both to user nodes and DMAS (for archival purposes).

A user will be able to set the format of the video feed by passing proper hardware requests to the Device Process, which in turn will translate and pass it to the Video Manager.

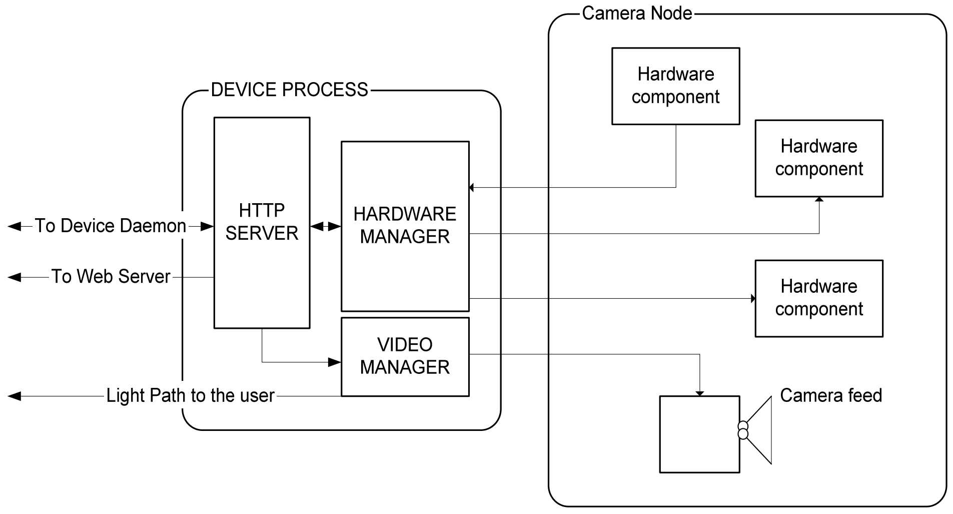

Fig. 3. Hardware control diagram

For simplicity of connectivity, the hardware infrastructure on which the Device Process runs should be located as close as possible to the actual physical device hardware.  While for NEPTUNE, the Device Process hardware will be part of the node itself, for the VENUS project, it will be located at the shore station, where the various I/O communication streams (e.g. serial data) are multiplexed over a fiber optic link to the undersea node.  The advantage of such a configuration, in particular as a prototyping test facility, is that this minimizes the number of components that will go underwater and hence reduces the threats of failure rate.

## 7. Processing Requests

The Web Server will receive and process requests from authenticated clients (U.Req and A.Req types of requests) and from authenticated Device Processes (H.Req).

Hardware requests serve the purpose of informing the Server about the current status of devices. They will be produced either on demand from Device Daemon applications, or by the device process itself on a regular basis and in case of any unexpected change in device status.

User and Administrator requests will either be used to control hardware devices or to affect internal system parameters (setup, administration, user permissions, user profiles etc). All requests to control hardware will be assigned a unique Request Id in order for the user to be able to track the execution of the request or to cancel a previously submitted request if needed.

All types of requests will be of “stateful” type, because they will have to retain at least session authentication parameters between submissions. The detailed request processing is explained in the diagram below (Fig.4a and 4b).

- User submits request to Web Server
- Server restores user session with Passcode$_n$ parameter
- Server authenticates user by Passcode$_n$
- Server records request into the database and generates unique Request Id.
- Server validates user Request
- Server sends request info to Device Daemon for execution
- Server generates new Passcode$_{(n+1)}$ and serializes user session
- Server responds to the user

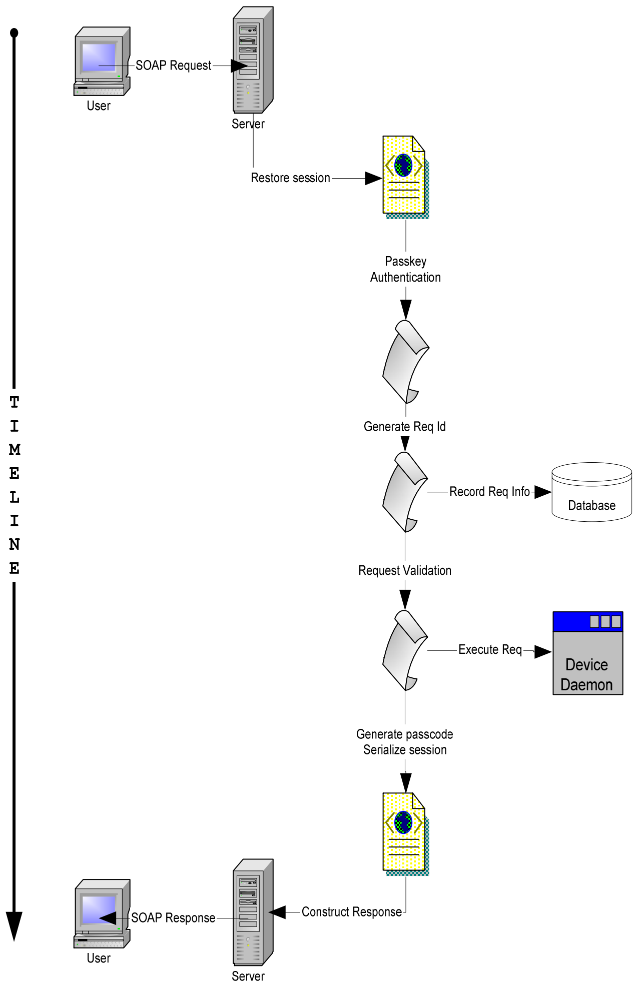

Fig. 4a. Processing user request to control hardware

- Device Process submits request to the Web Server
- Server restores device session with Passcode$_n$ parameter
- Server authenticates user by device key and Passcode$_n$
- Server records updated device info into the database.
- Server generates new Passcode$_{(n+1)}$, encrypts it using server key and serializes device session
- Server responds to device process

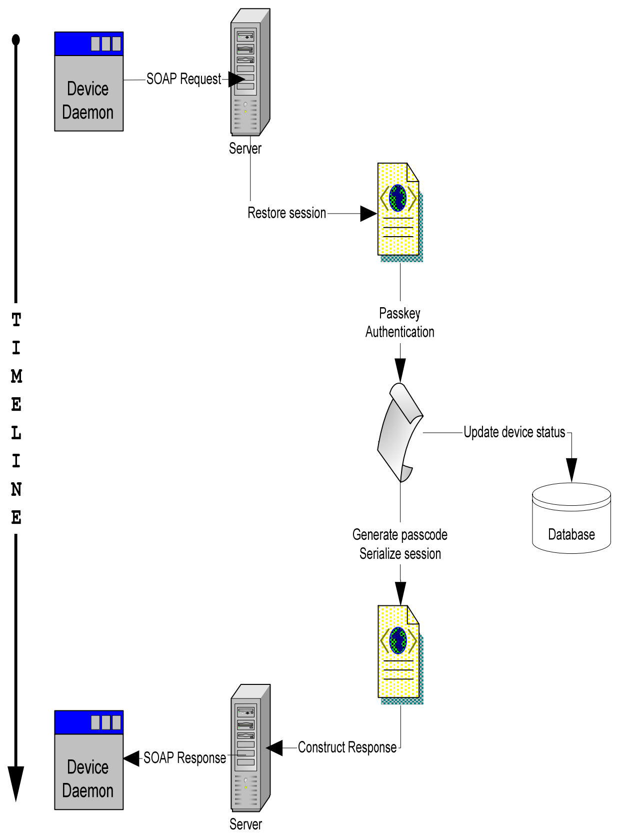

Fig. 4b. Processing device request to update status

Device Processes will also act as miniature Web servers, receiving requests in the form of SOAP messages from their client Device Daemon applications. The detailed algorithm of processing these requests cannot be fully designed until more information is available regarding actual hardware devices and their mechanisms supporting remote control and configuration.

**8. Security and Bandwidth Verification**

Since the software will operate over CA*net4, each component will require some form of security mechanism to ensure that only permitted consumers may access the resources. Obviously, stricter controls over the camera component will be required than over the video or sensor streams, as the latter can be shared with non-critical users. Bandwidth availability for requesting consumers might be verified by a sub-component of the video transmission component in order to provide improved guarantees on reliable delivery or to negotiate a reduced bandwidth stream, as appropriate. Because these aspects of the architecture are hidden from the user, these can be designed and developed without formal HCI methods and as such, should proceed in conjunction with development of the previous components. Security mechanisms will be engineered into the system from the outset, whereas the less critical aspects of bandwidth verification can be added at a later stage in the project.

## 9. Web Service Example

A request to WebServices to obtain parameters of a specified camera:

```
POST /CIIP/getcamera.php HTTP/1.1
Host: ws.underseawindow.org
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://ws.underseawindow.org/CIIP/getcamera.php"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetCameraPrm xmlns="http://ws.underseawindow.org/CIIP/">
      <CameraID>string</CameraID>
    </GetCameraPrm>
  </soap:Body>
</soap:Envelope>
```

The server responses with:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetCameraPrmResponse xmlns="http://ws.underseawindow.org/CIIP/">
      <GetCameraPrmResult>
        <CameraId>int</CameraId>
        <IPAddress>string</IPAddress>
        <ResponseCode>int</ResponseCode>
        <ResponseText>string</ResponseText>
        <Resolution> string</Resolution>
        <Pan>string</Pan>
        <Tilt>string</Tilt>
        <Zoom>int</Zoom>
        <Autofocus>boolean</Autofocus>
        <LightLevel>int</LightLevel>
      </GetCameraPrmResult>
    </GetCameraPrmResponse>
  </soap:Body>
</soap:Envelope>
```