UMOC – A C Library for Clients of ONVIF Network Video Transmitters
Library Design and Device Discovery Support

Sérgio F. Lopes[1], Sérgio Silva[2], José Cabral[1] and João L. Monteiro[1]
[1]*Centro Algoritmi, School of Engineering, University of Minho, Guimarães, Portugal*
[2]

Keywords: Video Surveillance and Control, ONVIF, SOAP, WS-Discovery, WS-Security, Web Services, NVT.

Abstract: Video surveillance and control systems are becoming increasingly important as video analysis techniques evolve. The interoperability of IP video equipment is a critical problem for surveillance systems and other video application developers. Open Network Video Interface Forum (ONVIF) is one of the two specifications addressing the standardization of networked devices interface, but it is a complex specification and difficult to implement. This paper describes a library that helps to develop clients of ONVIF video cameras, by taking advantage of opportunities to abstract useless details and to provide higher-level functionalities. The library architecture is explained and it is shown how it can be used to implement operations and features that present challenges to developers. The module supporting Device Discovery is addressed. We demonstrate how the library reduces the complexity, without affecting flexibility. The work presented has been validated by an industry partner.

1 INTRODUCTION

Since Internet Protocol (IP) digital cameras became less expensive, video surveillance and control systems are becoming increasingly important as video analysis techniques evolve. Furthermore, this equipment comes with digital outputs and inputs, allowing more integration of control and automation functions through the same network interface.

However, IP video cameras differ in a variety of ways, from provided features, to network configuration, user management, video encoding/compression schemes, supported network protocols, etc. The software interface offered by manufacturers to configure and use this diversity of features also varies. In this scenario, interoperability becomes a serious challenge, with numerous shortcomings for everyone (end users, integrators, and manufacturers). To tackle the problem, two industry groups – the Open Network Video Interface Forum (ONVIF) and the Physical Security Interoperability Alliance (PSIA) – were formed, with the goal of standardizing IP video surveillance. Both groups came out with specifications, both based on web services.

Web Services (WS) enable machine interoperability over a network by defining a standard way to describe service operations, data format and network protocol. Usually they are based on open and well-established standards such as HTTP for information transport and XML for data serialization. Two main approaches exist: REST-compliant and SOAP-based WSs. PSIA follows the former (PSIA, 2011), while ONVIF follows the latter (ONVIF, 2012). They have two other main differences: PSIA has a broader scope, addressing systems-level concerns like access control, while ONVIF is more focused on the device-level functionalities; and, ONVIF is built on WS standards and specifies more constraints than PSIA. For example, the former uses WS-BaseNotification (OASIS, 2006a) for events, while the latter defines only the format of message headers but not the bodies. The current PSIA vagueness means more complexity for implementing interoperability.

Our work focuses on ONVIF, which adopts a significant number of web standards (e.g., WSDL, SOAP, WS-Discovery (Beatty et al., 2005)) and consists of several specifications. Consequently, it is not trivial to implement, and to get acquainted with all specification details. Moreover, some ONVIF specifications have a significant level of complexity by themselves. This scenario applies to both service provider and consumer development, and our experience in using cameras from manufacturers not involved in ONVIF specifications shows that they are still making progress towards full support. We address the difficulties of developing applications that are clients of video cameras, called Network Video Transmitters (NVT) in ONVIF, namely by means of a library.

In the literature, there are few works involving ONVIF. The more focused ones (Senst et al., 2011); (Yi-Hsing et al., 2011) describe parts of ONVIF and particular applications. They do not address the difficulties of implementing an ONVIF library that provides generic functionality to help to develop NVT client applications. To our knowledge, ONVIF Device Manager (Synesis, 2013) has the only non-proprietary library available. It is an open source C# library to manage NVT devices that comes with a demo application. However, its API mirrors ONVIF operations, not offering a higher-level API easier to use. Additionally, industrial applications seek to increase efficiency, to raise competitiveness and mark a position on markets. So, computational performance and resource usage is very important and non-interpreted programing languages have advantages in this matter.

This paper introduces a C library supporting the development of NVT clients. The library was developed for an industry partner that integrates IP cameras to produce a complete line of surveillance equipment. The main contributions are (1) the library design and (2) an API that is much simpler than a direct reflection of ONVIF commands, (3) without affecting flexibility. More specifically, this is achieved by taking advantage of several opportunities, namely (4) abstract useless details, (5) make use of context, (6) provide higher-level functionalities, and (7) normalize some ONVIF aspects. We demonstrate how the library reduces the complexity by exemplifying how it supports the implementation of operations and features that present more challenges to developers.

In the next section ONVIF is briefly introduced, and its relevance for control and automation systems based on video is explained. The library architecture is explained in section 3. In section 4 we describe the API of Device Discovery module. The paper ends with sections 5 and 6 which respectively summarize the results and conclusions.

2 ONVIF OVERVIEW

ONVIF functions are defined as SOAP operations. Core specification (ONVIF, 2012), contextualizes the usage of WS-Discovery, WS-Security (OASIS, 2012a) and WS-BaseNotification, and defines Device Management (DM) and Event services. Other specifications define a single service. Besides NVTs, ONVIF devices are classified in types – NV Display, NV Storage and NV Analytics – for which a set of mandatory (M), mandatory if the device has a related feature (C), and optional (O) services are defined. A NVT device has the following set of services: M = {Device Management, Event, Media, Streaming, Device IO}, C = {PTZ}, and O = {Imaging, Video Analytics}.

Services' operations are protected by user authentication and a security policy. Authentication credentials should be provided at either the transport-level (HTTP), using Digest Access Authentication (IETF, 1999), or the message-level (SOAP), using any token profile defined by WS-Security. Servers shall at least support the WS-Security UsernameToken Profile (OASIS, 2012b), which requires clients to send the username of an existing account and respective password digest.

ONVIF enables the development of automation systems based on NVTs, wherein the controller is a client. The input of such systems can be both video and digital inputs, and the control can be made through NVT relay outputs or any other devices. The detection of actuation scenarios can be made by processing video streams at either ONVIF clients or NVTs supporting the optional Video Analytics service. NVT's inputs sensing and outputs command is made through the mandatory Device IO service. Asynchronous real-time control can be implemented by using the Event service. This service enables the subscription of input and Video Analytics events that, when detected by NVTs, are automatically sent to the client controller (using Basic Notification Interface (OASIS, 2012a); (ONVIF, 2012). Events can also be received on request (using Real-time Pull-point Notification Interface (ONVIF, 2012) and through metadata streaming (Notification Streaming Interface (ONVIF, 2012).

3 LIBRARY DESIGN

UMOC is a C library that provides functions to manage ONVIF NVT devices. Figure 1 introduces the library, showing how it is built and its

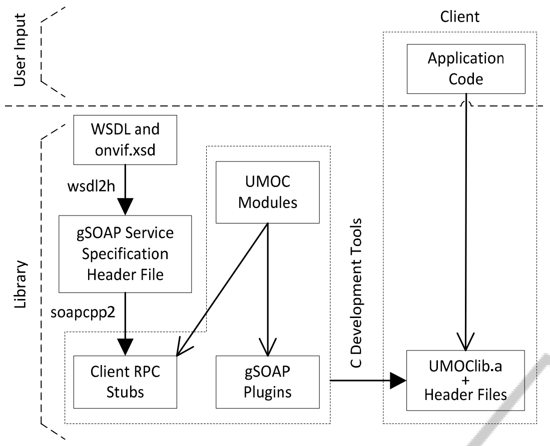

Figure 1: UMOC library development and integration in client applications.

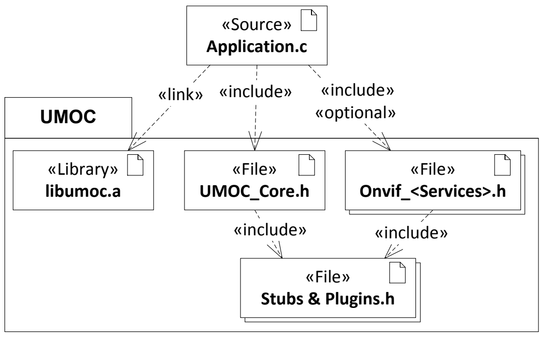

Figure 2: UMOC structure and usage.

integration with application code. We have used gSOAP toolkit (Engelen and Gallivan, 2002) to develop the library, see (Lopes, 2013) for more information. It is built on top of ONVIF client stubs generated by gSOAP and plugins supporting base web services (e.g., WS-Discovery, WS-Security) from different organizations.

The library is composed of several modules, each of which implements the features of an ONVIF service, see figure 2. This approach does not require applications to include all UMOC code, gives structure and simplifies the library architecture, and makes it easier to maintain individual services. The exception is UMOC_Core, which is a module common to all services. The application must always include UMOC_Core.h, and others header files according to the desired services.

To simplify usage without losing flexibility and options, the library is designed in two layers: High Level (HL) and Low Level (LL).

3.1 Low-Level Layer

LL functions invoke stubs (see figure 3) but provide a higher level API much easier to use. They abstract several details (including gSOAP, in its totality), by handling internally and hiding the parameters:

- SOAP context and message action (which are respectively the 1st and 3rd parameters of all stubs); and,
- empty requests and responses; and automating a set of crucial operations:
- verify mandatory request data before calling any stub (i.e., sending a message);
- add authentication; and,
- identify errors returned by stubs and register their description.

Moreover, LL functions also abstract parts of ONVIF messages, namely (optional) extensions and a few other (mandatory) fields. Extensions allow vendor specific functionalities to be provided and also ONVIF to evolve while preserving backwards compatibility (because they are optional). Therefore, they are not supported by many devices (which is the case of all that we have used). The library only supports extensions that are used by ONVIF and are important for automation and control, namely the configurations concerning AudioOuput (of Media profile). Other fields are, for example, the mandatory UseCount of all Media configurations, which are automatically updated. Consequently, simplified structures are defined for function parameters types, which in turn is an opportunity to also shorten their generated name.

For example, consider the differences between the stub and setNTP_LL functions, respectively in figures 4 and 5. As mentioned earlier in this section, stub's context (1st), action (3rd) and empty response

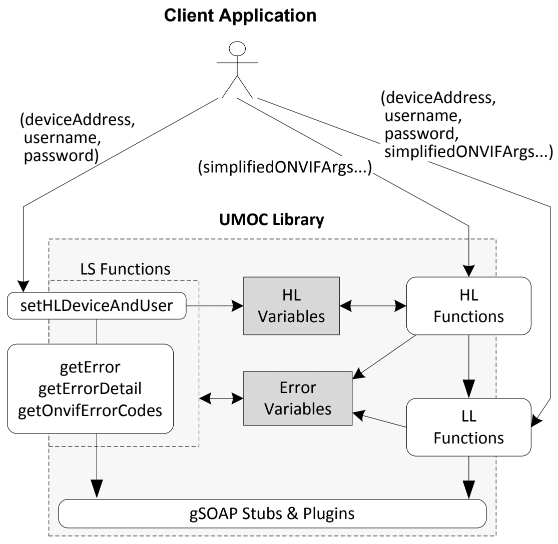

Figure 3: UMOC architecture.

```
struct schema__NetworkHost {
    enum schema__NetworkHostType Type;
    char **IPv4Address;
    char **IPv6Address;
    char **DNSname;
    struct schema__NetworkHostExtension
        *Extension;
};

struct _device__SetNTP {
    enum xsd__boolean FromDHCP;
    int __sizeNTPManual;
    struct schema__NetworkHost *NTPManual;
};

int soap_call___device__SetNTP(
    struct soap *soap,
    const char *soap_endpoint,
    const char *soap_action,
    struct _device__SetNTP
        *device__SetNTP,
    struct _device__SetNTPResponse
        *device__SetNTPResponse);
```

Figure 4: SetNTP stub and input parameters' structure.

(5th) parameters are hidden, and credentials' parameters are added to the library function for authentication. (The 2nd stub parameter becomes the 1st parameter of setNTP_LL.) Besides this, stub's request (4th) parameter is a pointer to a structure that contains three fields, respectively: indication to update NTP server list via DHCP or manually, number of manual servers, and an array with servers' identities. Each element of the latter is another structure which contains: type of server identity (IPv4, IPv6, or DNS), server address/name, and optional extension. The setNTP_LL function contrasts with such a complicated structure nesting and indirections, by offering a minimalist set of request parameters: number of servers, and array of strings with addresses/names. This is possible because setNTP_LL automatically detects each kind of server identity that is passed to it (according to RFC 1123 rules). The optional and (in this case) non-standard extension is discarded. Hence, the inner structure of stub parameter is replaced by a single string. Furthermore, for the library function an argument of zero servers means that DHCP should be used, thus making redundant the DHCP/manual boolean of stub's parameter outer structure.

This way the function greatly simplifies the interface by reducing the number and complexity of parameters, and, in a smart way, it accordingly sets the proper stub arguments.

In sum, LL functions offer the same functionalities and freedom as stubs with several advantages: much simpler interface, avoidance of incomplete requests, automatic authentication and registration of error descriptions.

```
int setNTP_LL (
    const char *devMgmtAddress,
    const char *user,
    const char *pass,
    int numOfManualServers;
    char **manualServerAddressOrName);
```

Figure 5: setNTP_LL function.

3.2 High-level Layer

HL functions do not require the service address and security credentials (username and password) to be provided. For that purpose the setHLDeviceAndUser function was introduced, which receives as arguments a service address, username and password, and stores them internally in HL variables. This is illustrated in figure 3 that shows the library architecture. setHLDeviceAndUser uses its arguments to consult the capabilities of the selected device and saves (in HL variables) the address of each supported service (such as Event, PTZ, etc.). Whenever a HL function is called by the application, the library will use (from HL variables) the proper service address, and username+password as security credentials.

Since ONVIF has optional services, HL functions check if the respective services are provided by the selected device before sending any requests (i.e., calling LL functions). For example, if the application calls a function that uses PTZ and capabilities do not include a PTZ service address, an error is automatically returned, thus saving time and network bandwidth.

It is also very important to validate requests' content before sending any message. As mentioned in previous section, LL functions check if mandatory request data is present. HL functions go further, and implement the validation of arguments whose values depend on configuration options (of Media, Imaging and PTZ services). HL functions that retrieve possible options, store them in HL variables. HL functions with input arguments containing configuration options check if every option value is valid. In case of invalid arguments, these functions return an error and identify the argument. It should be noted that LL functions cannot validate configurations options, unless they request them, because they operate with any camera (i.e., they do not know what a selected device is), and thus cannot store valid options.

To summarize, HL functions offer the most simplified interface, maximum avoidance of

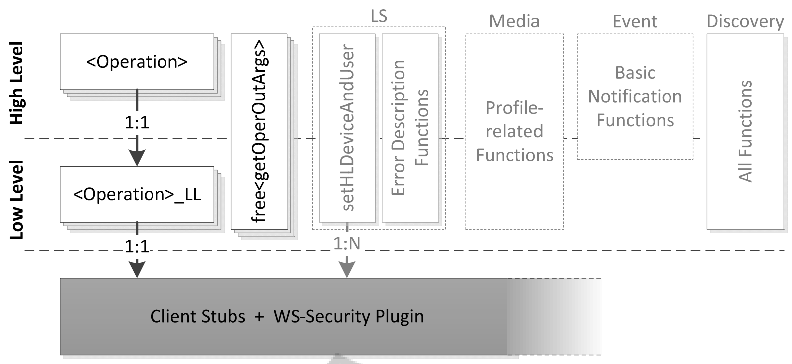

Figure 6: API functions pattern (in black) and exceptions (in gray).

worthless messages, and highest automation level, usually without any limitations.

3.3 Error Management

All library functions return an integer value, wherein zero means success and a different value is the number of the occurred error. When an error occurs in a library function, it registers the respective description in dedicated variables. After the function returns, the application can consult error information using three library functions: getError, getErrorDetail and getOnvifErrorCodes.

Errors can occur at different levels: HTTP, SOAP, ONVIF and library itself. Therefore, error values are divided in four separate ranges:

- {1} for all ONVIF errors, which do not have an actual value and are coded in text;
- [2, 99] for SOAP errors (being 48 the current maximum);
- [100, 899] for HTTP errors (keeping their original value, currently limited to 599); and,
- [900, 999] for library specific errors.

getError and getErrorDetail functions can be used to obtain descriptions of all errors except ONVIF. The former function returns the reason for error occurrence, and the latter obtains additional information, if available.

For example, when the application invokes a function with invalid input arguments, it returns a library error code and getError retrieves the generic message "Invalid argument passed to library function". Detailed error descriptions are available for functions' specific errors, namely regarding their particular arguments. For instance, if setNTP_LL receives an incorrect server address in its 5th argument (see figure 5), for example in the 1st string, then getErrorDetails obtains "The 1st NTP server address is not a valid IPv4, IPv6 or name".

Whenever possible, getError also provides semantically rich error descriptions, though still generic. For example, when the application invokes any HL function without first calling setHLDeviceAndUser, it gets a library error and getError retrieves the message "setHLDeviceAndUser must be called before HL functions". In such cases getErrorDetail does not offer any further information.

When ONVIF errors occur, all three functions can be used to obtain the text received in the command response message. In this case, getError, getErrorDetail and getOnvifErrorCodes return respectively the Reason, Detail, and ErrorCodes of ONVIF message, see (ONVIF, 2012).

3.4 API Rationale

The two library layers have a pattern of characteristics and relationships that are common across almost all services/functions, see figure 6: (1) LL functions wrap stubs in a 1:1 relationship, (2) LL functions have the names of HL functions suffixed with "_LL", and (3) both provide the same base functionalities, with LL functions adding only input parameters for service IP address, username and password. For different reasons, the exceptions are all Device Discovery (DD) service functions, Basic Notification functions of Event service, and profile-related functions of Media service. (The latter two to be addressed in a future paper due to their complexity and space required.)

The free<getOperOutArgs> functions are provided to help the application developer to release non-contiguous memory allocated by HL and LL functions that obtain data whose size is not fixed (or known a priori). The respective output arguments are pointers to structures that, in turn, contain pointers to other structures. The getCapabilities (both HL and LL) functions are examples of them,

```
typedef struct Capabilities_Info
{
     AnalyticsC *Analytics;
     DeviceC *Device;
     EventC *Events;
     ImagingC *Imaging;
     MediaC *Media;
     PTZC *PTZ;
} Capabilities;

int getCapabilities(Capabilities** cap);

int freeCapabilities(Capabilities* cap);
```

Figure 7: Example of a structure with independently (de)allocated memory blocks.

for which there is a freeCapabilities function, shown in figure 7, to free the Capabilities structure. The structure has six (possibly NULL) pointers to memory blocks be independently de-allocated, each of which with its own (sub-)pointers to further independent blocks (not described in the figure). By offering free<getOperOutArgs> functions the library frees the developer from the burden of traversing the tree of pointers specific to each different structure.
The setHLDeviceAndUser and three error functions (described in previous section) belong to the Library Specific (LS) functions group, once they do not execute any specific ONVIF operation but rather set HL variables to be used by HL functions, and consult the last occurred error information, respectively. This group is included in the UMOC_Core module, which also contains the HL and error variables.

4 DEVICE DISCOVERY

ONVIF adopts WS-Discovery 1.0 (with a few extensions) to search for devices in local or remote networks. gSOAP supports WS-Discovery, offering a plugin that provides functions to send all service messages (Probe and Resolve for clients; Hello, Bye, ProbeMatch and ResolveMatch for servers). The plugin API also includes a function to listen to (and receive) any discovery messages and declares handlers for each message. The library implements the discovery of devices: it sends Probes, listens for discovery messages, collects ProbeMatches, and finally returns only the most relevant data of matches. Resolve messages are not needed because ONVIF requires devices to have a transport address and therefore it must be included in the XAddrs of ProbeMatches (according to both ONVIF and WS-Discovery).

```
Discovery

int find(DeviceTypesCode dtc, char* scopes[],
              int *numDevices, DeviceInfo** list);
int findNVTs(NvtScope code, char* locations[],
              char* hardware[], char* names[], char* other[],
              int *numDevices, DeviceInfo** list);
int freeDeviceList(int numDevices, DeviceInfo* list);
```

Figure 8: Discovery Module API.

Devices can be found according to the following search parameters:

- *Types* (the device implements);
- *Scopes* (the device is in), allowing devices to be organized into logical groups;
- *Matching rule*, defining how scopes are matched.

ONVIF specific discovery behaviour is achieved by using its device types and scopes. It defines "tds:Device" generic type for all devices, and "dn:NetworkVideoTransmitter" type for NVTs. ONVIF defines standard scopes with the syntax "onvif://www.onvif.org/<category>/<value>", and five categories: "Profile", "location", "hardware", "name" and "type". Each category is followed by a value, which can be constrained to belong to a predefined set. Device owners can define other (specific) scopes not subject to this syntax. ONVIF defines RFC 3986 as the mandatory matching rule for all devices.

The library provides two functions to search for devices, shown in figure 8, and function freeDeviceList to release memory allocated by both of them. As illustrated in figure 6, Device Discovery service functions have all the same level (neither LL nor HL).

The find function searches for all ONVIF devices, and has as input parameters: a code to select among any subset of the 4 device types, and an optional list of scopes to narrow down the results. The 1st parameter is a bit field that can be conveniently set up by making an OR statement of enumerated constants that are provided. Figure 9 shows an example search for devices that are either NVT or NVS, and are located (physically, if properly configured) in the campus of Azurém. The library also provides a constant ("ALL_TYPES") to allow more practical search for devices of any type. The list of scopes is an array of pointers to strings, that must be terminated with an empty string or a NULL pointer, a solution that was preferred over parsing some URI delimiting scheme.

The findNVTs function searches exclusively for NVT devices, having as input parameters: a code to select types of NVTs (according, to the "type" category of standard NVT scopes), and, optionally,

```
char* scope[] =
{"onvif://www.onvif.org/location/Azurem",
    NULL};
int numDevices = 0;
DeviceInfo* listDevices = NULL;
resp = find(NVT_TYPE || NVS_TYPE,
            scope,
            &numDevices, &listDevices);
```

Figure 9: Example to discover NVT and NVS devices located in Azurém campus.

```
char* location[] = {"Azurem",
                    "building/B", NULL};
int numDevices = 0;
DeviceInfo* listDevices = NULL;
resp = findNVTs(NONE,
        location, NULL, NULL, NULL,
        &numDevices, &listDevices);
```

Figure 10: Example to discover all NVT devices located in building B of Azurém campus.

values for the standard scope categories “location”, “hardware” and “name”, and a list of user-specific scopes (that can be used in the same manner as the 2nd parameter of find function). The 1st parameter is a bit field that (similarly to find) can be set up using a set of enumerated constants that are provided: “none”, “video_encoder”, “ptz”, “audio_encoder”, and “video_analytics”. For example, in figure 10, the function is used to search for any NVTs that are located in the building named “B” of Azurém campus. Note that the “locations” parameter only needs the value part of “location” category scopes. This is so because it is a standard scope category, which also applies to the “hardware” and “names” parameters.
Both functions have two output parameters: the first returns the number of devices that are found, and the second is an array that provides for each device a subset of ProbeMatch message data: type, address of device management service, and scopes (divided by same categories as the 2nd to 5th input parameters of findNVTs).

It should be noted that the library uses the only matching rule that ensures full compatibility with ONVIF devices: “http://schemas.xmlsoap.org/ws/ 2005/04/ discovery/rfc3986”.

5 RESULTS

The library supports the following features: device discovery, consulting of device capabilities and information, configuration of date/time, NTP, DNS and Dynamic DNS settings, management of user accounts, consulting and actuation of physical outputs, setting of image parameters, subscription and reception of events, control of PTZ unit, media profiles management and requesting snapshot and stream URIs.
We have developed a NVT client application that demonstrates the usage of the complete API. This application has been used to test the library using several IP cameras with ONVIF support, from different manufacturers. All library features were tested successfully, although we faced minor problems in cameras’ implementation of ONVIF. These problems are considered as normal, since ONVIF specifications are recent and evolving. Consequently, manufacturers that do not participate in the specifications are still making progresses to implement flawlessly all ONVIF services.
The library has already been delivered to the industry partner and is being easily integrated, replacing legacy code. The library offers functionalities designed to be generically applicable, and the only request we had was the addition of a function that addresses specific needs of the partner: provide resolutions of all codecs supported by a device (whether or not included in VideoEncoderConfigurations of existing profiles), and all existing streaming URIs.
This library enables an easier development of controllers for automation systems, wherein a controller is a client of a group of NVTs. The industry partner is starting to make their controllers this way. Gradually they are integrating NVTs that explore ONVIF capabilities to command alarms and other security systems based on video analysis techniques.

6 CONCLUSIONS

This paper introduces a C library supporting the development of ONVIF NVT clients. The goal of the library is to help developers to handle the complexity of ONVIF specifications, by providing a simpler API and higher-level functionalities, without affecting flexibility. We have explained the library design and how it takes advantage of several aspects of the specification and the integration of features to simplify the API. We demonstrate how the library reduces the complexity by exemplifying how it supports the implementation of operations and functionalities that present challenges to developers.

This work has been validated by an industry partner that is using the library to develop its products. Future work includes adding support for multithreading and other ONVIF services besides NVT.

REFERENCES

Engelen, R. and Gallivan, K., 2002. The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks. In *2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid2002)*, Berlin, Germany, pp. 128-135.

IETF, June 1999. *RFC 2617, HTTP Authentication: Basic and Digest Access Authentication.* [Online]. Available: http://www.ietf.org/rfc/rfc2617.txt.

J. Beatty, et al, April 2005. *XMLSOAP, Web Services Dynamic Discovery (WS-Discovery).* [Online]. Available: http://specs.xmlsoap.org/ws/2005/04/discovery/ws-discovery.pdf.

Lopes, S. F., Silva, S., Mendes, J., Metrolho, J. C. and Duque, D., February 2013. Development of a library for clients of ONVIF video cameras: challenges and solutions. In *Proceedings of the IEEE International Conference on Industrial Technology (ICIT'13)*, IEEE Industrial Electronics Society.

OASIS, October 2006. *Web Services Base Notification 1.3 (WS-BaseNotification).* [Online]. Available: http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf.

OASIS, October 2006. *Web Services Topics 1.3 (WS-Topics).* [Online]. Available: http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf.

OASIS, May 2012. *Web Services Security: SOAP Message Security 1.1 (WS-Security).* [Online]. Available: http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf.

OASIS, May 2012. *Web Services Security UsernameToken Profile 1.1.1.* [Online]. Available: http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf.

ONVIF, December 2012. *ONVIF™ Core Specification, Version 2.2.1.* [Online]. Available: http://www.onvif.org/specs/DocMap.html.

PSIA, August 2011. *PSIA Service Model, Version 1.2.* [Online]. Available: http://www.psialliance.org/documents.html.

Senst, T., Patzold, M., et al, September 2001. On building decentralized wide-area surveillance networks based on ONVIF. In *8th IEEE International Conference on Advanced Video and Signal-Based Surveillance (AVSS 2011)*, pp.420-423.

Synesis, March 2013. *ONVIF Device Manager (Onvifdm) v2.2.231.* [Online]. Available: http://sourceforge.net/projects/onvifdm.

Yi-Hsing Tsai, Jung-Kuang Hsu, Yun-Ei Wu, Wei-Feng Huang, June 2011. Distributed Multimedia Content Processing in ONVIF Surveillance System. In *International Conference on Future Computer Sciences and Application (ICFCSA 2011)*, pp.70-73.

W3C, April 2012. *XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*. [Online]. Available: http://www.w3.org/TR/xmlschema11-2.