MPSI TECHNOLOGIES

WhizniumSBE

Machine-to-Machine Communication in WhizniumSBE Projects

White Paper

August 17, 2018
updated September 19, 2018

Alexander Wirthmüller
aw@mpsitech.com

MPSI Technologies GmbH

FOR INTERNAL USE

1 Introduction

The Whiznium product line provides embedded software developers with innovative tools that make use of automated code generation for the development of feature-rich applications, e.g. in IoT, computer vision and robotics.

The Whiznium Service Builder's Edition (SBE) targets hardware control applications that are based on Embedded Linux powered boards such as the Raspberry Pi 3, Minnowboard Turbot or ZedBoard. Hardware can range from sensors, cameras and servo drives to complete FPGA-based sub-systems. Based on a fine-grained model definition, WhizniumSBE generates C++ code for an application-specific multi-threaded main executable in build-ready fashion, along with a SQL database, web-based user interface (UI) and numerous connectivity options.

The range of capabilities covered by WhizniumSBE and by its sister tool for FPGA system development, WhizniumDBE, is illustrated in Figure 1.

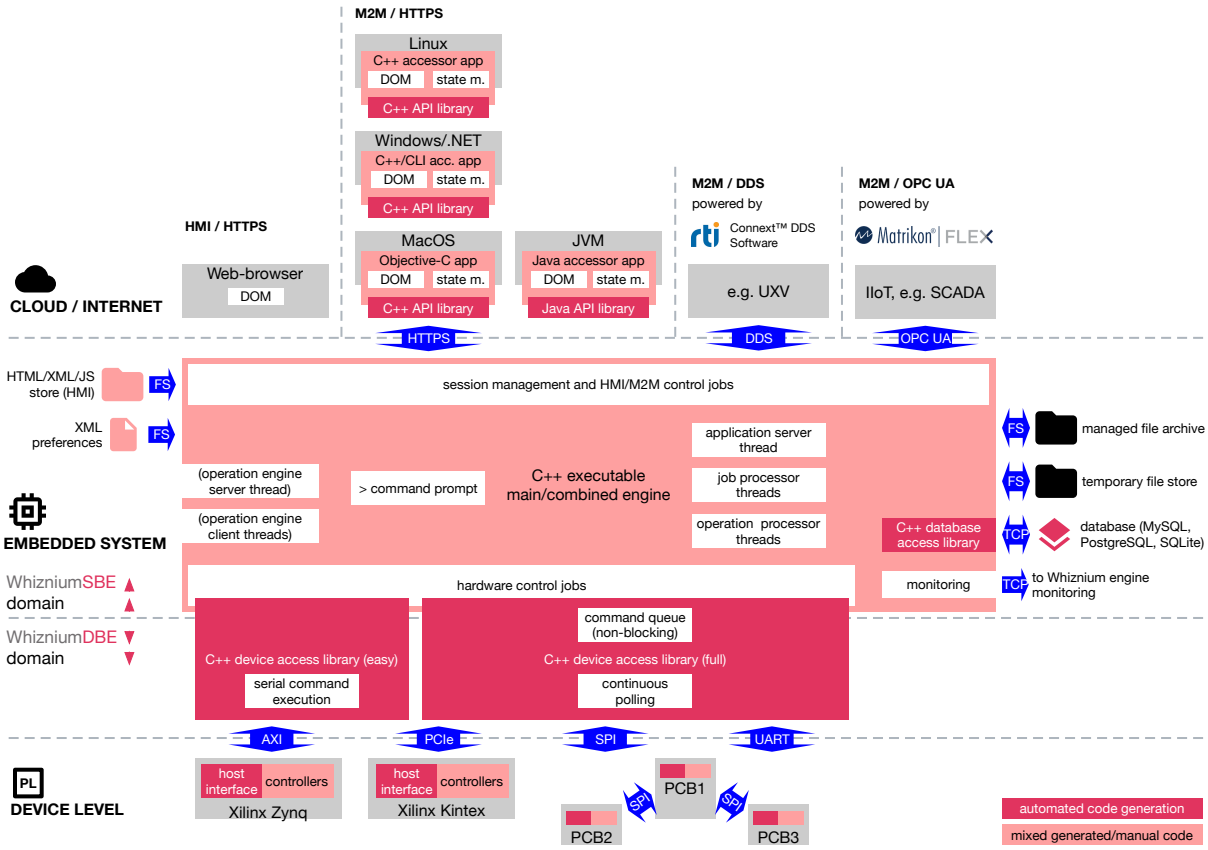

Figure 1: Schematic view of device level, embedded system and cloud/internet parts of a Whiznium-powered application along with the corresponding source code features.

This document focusses on Machine-to-Machine (M2M) communication, i.e. the connectivity options which allow WhizniumSBE-powered applications to interact with other machines. Scenarios include, but are not limited to, the transfer of life machine health data into the cloud, or the real-time provisioning of pre-processed vision information within a smart robotics system.

M2M communication for WhizniumSBE can be subdivided into two categories: on one hand there is the exchange of generic XML data blocks via HTTPS which closely follows the web-based UI or Human-Machine-Interface (HMI). To simplify the process of developing client-side applications, WhizniumSBE provides project-specific C++ and Java API libraries, along with supporting code generation for C++/CLI/Objective-C and Java.

On top of the generic option, the developer can choose to embed an industry-standard OPC UA server and/or a DDS publisher. The OPC UA server functionality is provided by means of the Matrikon® Flex OPC UA SDK, whereas RTI® Connext™ DDS Professional Software is used to ensure standard-compliant DDS connectivity. OPC UA and DDS bypass the structure given by the HMI and rather provide low-level access to hardware-related methods and variables.

Both M2M communication categories have in common that detailed access rights can be attributed to various user roles.

In the following chapters, the Multi-Spectral Detector will be introduced as an example for a complex WhizniumSBE-powered project. Its features and source code will then be used to shine a light on the mechanisms that make M2M communication in WhizniumSBE projects work. The document concludes with a comparison chart for generic vs. OPC UA vs. DDS connectivity.

2 The Multi-Spectral Detector Project

The embedded system used to describe M2M communication is one variant of MPSI Technologies' modular vision demonstrator. This demonstrator, conceived to highlight the full breadth of Whiznium's capabilities, can be re-configured e.g. to include low-cost USB vs. high-end GigE cameras. Also the embedded hardware can be re-arranged from an all-programmable Xilinx Zynq SoC (ARM cores and FPGA hard-wired in the same package) to separate high-performance Intel Atom quad-core and Xilinx Kintex7 FPGA boards.

The actual configuration chosen here is depicted in Figure 2.

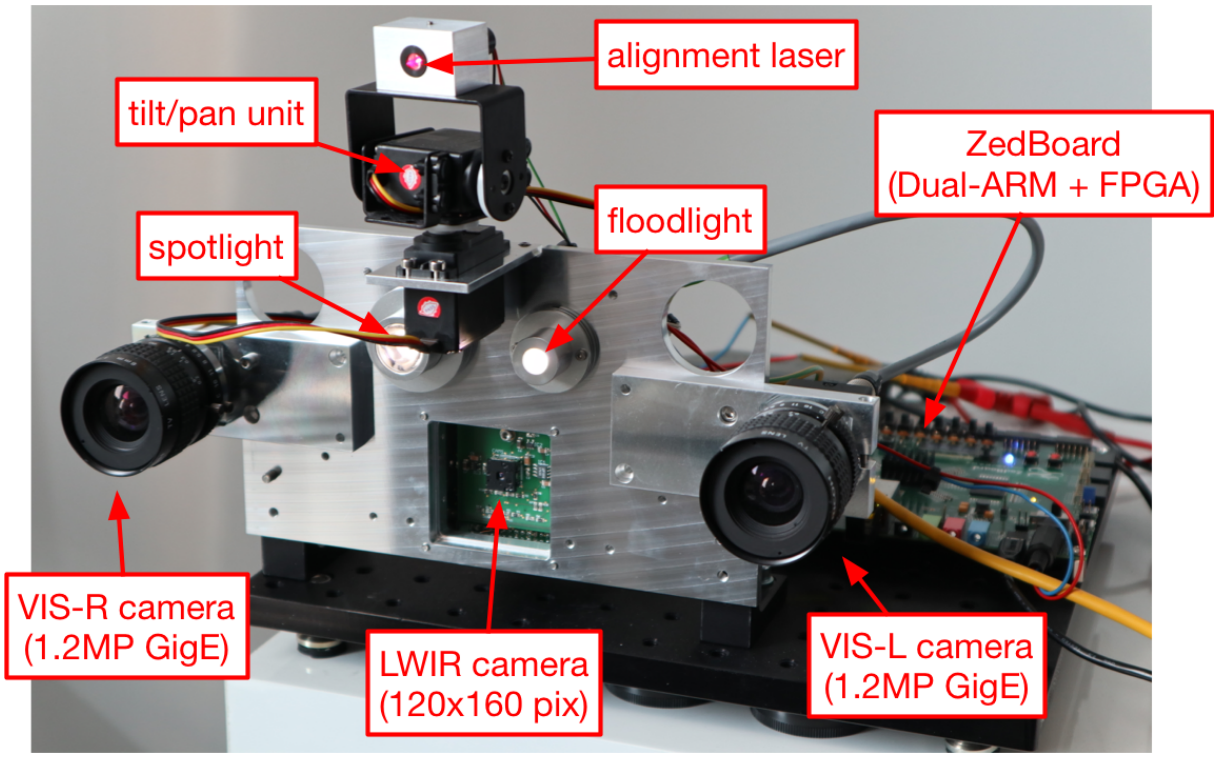

Figure 2: picture of the vision demonstrator.

In the following sub-chapters, a brief description of each relevant hardware handling / processing feature is given ; the job identifiers (see chapter 3.2) in parentheses denote the C++ class which is responsible for handling the respective feature.

2.1 Tilt sensor (JobMsdcAcqAdxl)

The tilt sensor is an Analog Devices ADXL345 3-axis accelerometer which is surface-mounted onto the detector's mainboard. Its readout via SPI is handled in the FPGA sub-system. The sensor serves as the digital equivalent of a water bubble.

2.2 Thermal imager (JobMsdcAcqLwir)

A FLIR Lepton3 thermal imaging core based on microbolometer technology for long-wave infrared (LWIR) light is used to obtain 120 x 160 pixel thermal images at a rate of 9Hz. Control and readout is accomplished via I2C and SPI interfaces, respectively. Both

functionalities are implemented in the FPGA sub-system, including on-FPGA buffering and streaming of frames.

An exemplary scene is shown in Figure 3.

Figure 3: The same scene as LWIR image (left) and as visible light image (right). The most prominent feature in the infrared is the hand due to its thermal signature, whereas the brightest feature in the visible range is the red alignment laser spot.

2.3 Visible light cameras (JobMsdcAcqVisl/Visr)

Machine vision is enabled by dual FLIR BlackFly Gigabit Ethernet cameras which provide 1280 x 960 pixel RGB images at frame rates of up to 52fps. FPGA-generated trigger signals allow for the time-synchronous acquisition from the left (VIS-L) and right (VIS-R) cameras.

The screenshot in Figure 4 shows two simultaneous web-based UI sessions, streaming from VIS-L and VIS-R, respectively.

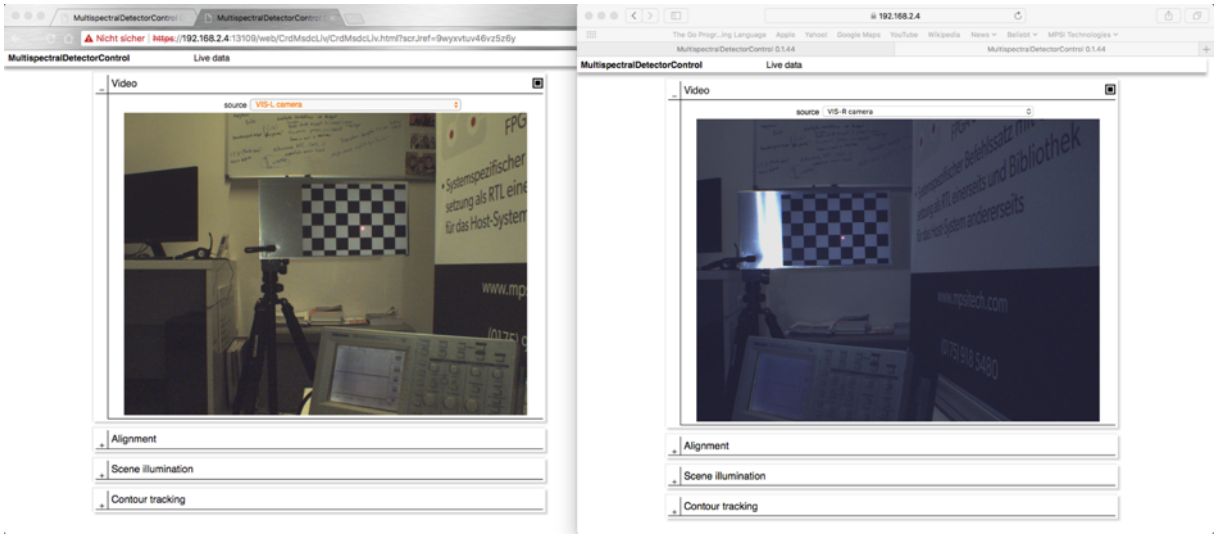

Figure 4: VIS-L and VIS-R views of the same scene.

2.4 Alignment laser (JobMsdcActAlign)

The system features a red 635nm alignment laser (eye-safe, sub 1mW class), the optical output of which can be modulated continuously with an analog input signal. This signal is generated by a Maxim MAX5711 DAC which in turn is controlled from the FPGA sub-system via SPI.

The laser provides a visual feature with unique fingerprint to be detected by the VIS-L/R cameras. To this end, laser modulation is synchronized with the VIS-L/R hardware trigger signals, resulting in the exemplary time-series shown in Figure 5.

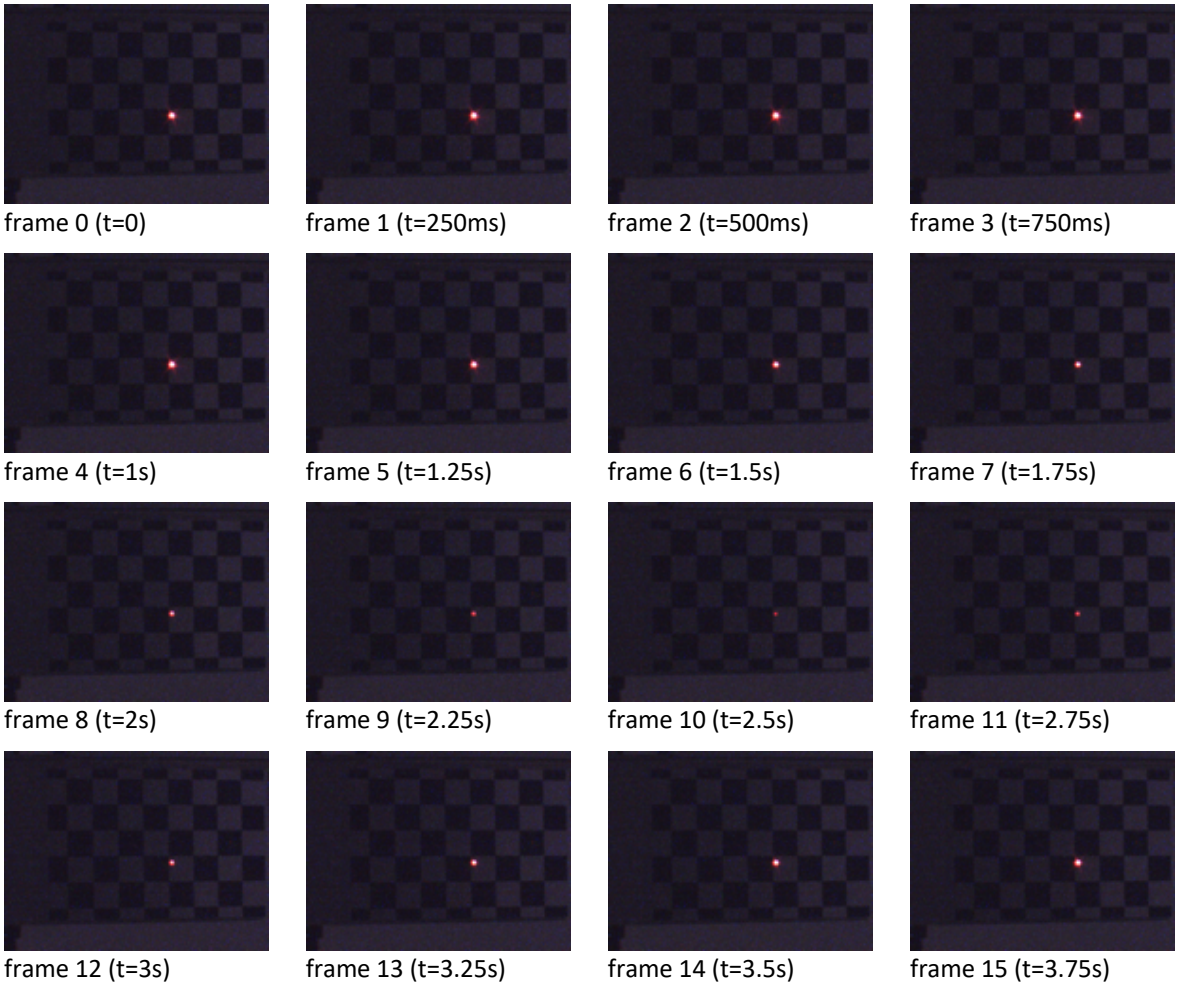

Figure 5: Time series of the VIS-L camera synchronized to the alignment laser running a sinusoidal sequence with a period of 4s. Peak intensity at t=500ms.

2.5 High power LED's (JobMsdcActLed)

A spotlight (2x 15° opening angle) and a floodlight (2x 60° opening angle) are driven by each one Linear LT3474 step-down converter in constant-current mode. The apparent light intensity is regulated with 100Hz PWM signals (1% to 100% duty-cycle) which are generated by the FPGA.

The corresponding web-based UI controls are shown in Figure 6.

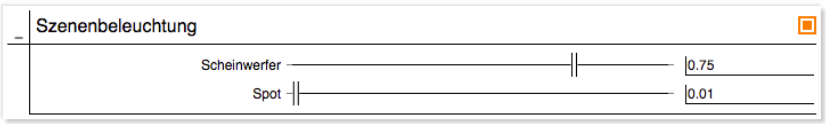

Figure 6: Web-based UI panel for controlling the LED intensity for a German-speaking user.

2.6 Tilt-pan unit (`JobMsdcActServo`)

For advanced functionality such as feature tracking, the alignment laser can be rotated around two axes. The hardware implementation uses two standard RC servos which use a FPGA-generated PWM signal to set their respective angles.

Figure 7 shows the web-based UI for manual control of this feature.

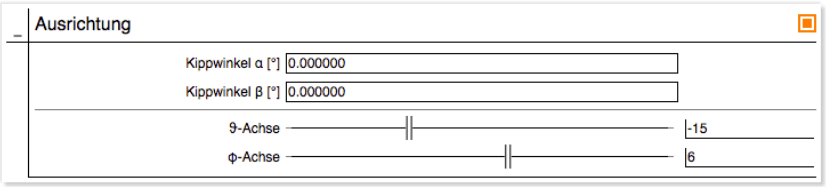

Figure 7: Web-based UI panel for manually controlling the tilt-pan unit. The tilt angle read-out is visible as well.

2.7 3D reconstruction (`JobMsdcPrcStereo`)

WhizniumSBE applications are multi-threaded and well-suited for running multiple concurrent tasks. Event-driven activation of certain (post-)processing jobs, in this case the combination of time-synchronous VIS-L and VIS-R frames to a depth map as they become available, can be modelled in detail, resulting in robust, powerful source code.

The stereo vision feature / generation of a depth map is currently under revision. It makes use of standard algorithms provided by the OpenCV framework.

2.8 Contour tracking (`JobMsdcPrcTrack`)

Another planned feature is the recognition of a “same-color” shape within the view of the VIS-L camera which is subsequently tracked by the red alignment laser. This feature implies combining `JobMsdcAcqVisl`, `JobMsdcActAlign` and `JobMsdcActServo` into a closed feedback loop.

3 WhizniumSBE Project Architecture

3.1 Project components

The development process with WhizniumSBE results in the compile-/ deploy-ready source code components which are highlighted in bold in Figure 8. The naming conventions of WhizniumSBE require a 4-letter abbreviation for each project, `Msdc` was chosen for the detector project.

`_mdl/msdc`	model files ; information to be processed by WhizniumSBE
`msdccmbd`	C++ source files for main executable (“combined engine”) ; auto-generated with manual insertion points
`msdccmbd/Msdccmbd_exe.cpp`	command line and entry point `int main()`
`msdccmbd/MsdccmbdAppsrv.cpp`	HTTPS server for generic communication
`msdccmbd/MsdccmbdDdspub.cpp`	DDS publisher
`msdccmbd/MsdccmbdUasrv.cpp`	OPC UA server
`_ini/msdc`	IDL input file for RTI® Connext™ DDS Code Generator
`_rls/dbsmsdc_*` `_rls/msdccmbd_*`	shell scripts and make files to perform builds on several platforms
`webappmsdc`	JS/HTML/XML files for the web-based UI ; auto-generated with manual insertion points

`_ini/dbsmsdc`
`_ini/msdccmbd_*`
`_ini/msdccmbd`
	SQL script to establish the SQLite database, platform-specific preferences file, XML file to populate the SQLite database (initial fill)

`apimsdc`	C++ source files for the API library
`dbsmsdc`	C++ source files for database access (by table / vector)
`japimsdc`	Java source files for the API library

Figure 8: Project folder overview, by order in which components are needed.

All further discussion is related to the main/combined engine code, to be found in the `msdccmbd` folder.

Link to code: https://github.com/mpsitech/MultiSpectralDetectorControl

3.2 Functionality selection

WhizniumSBE model specification is done using eight text-based model files in total, each representing one aspect of the model, e.g. database, user interface or deployment. Connectivity options to be implemented in code on the other hand are part of each project version’s standard functionality selection, as shown in Figure 9.

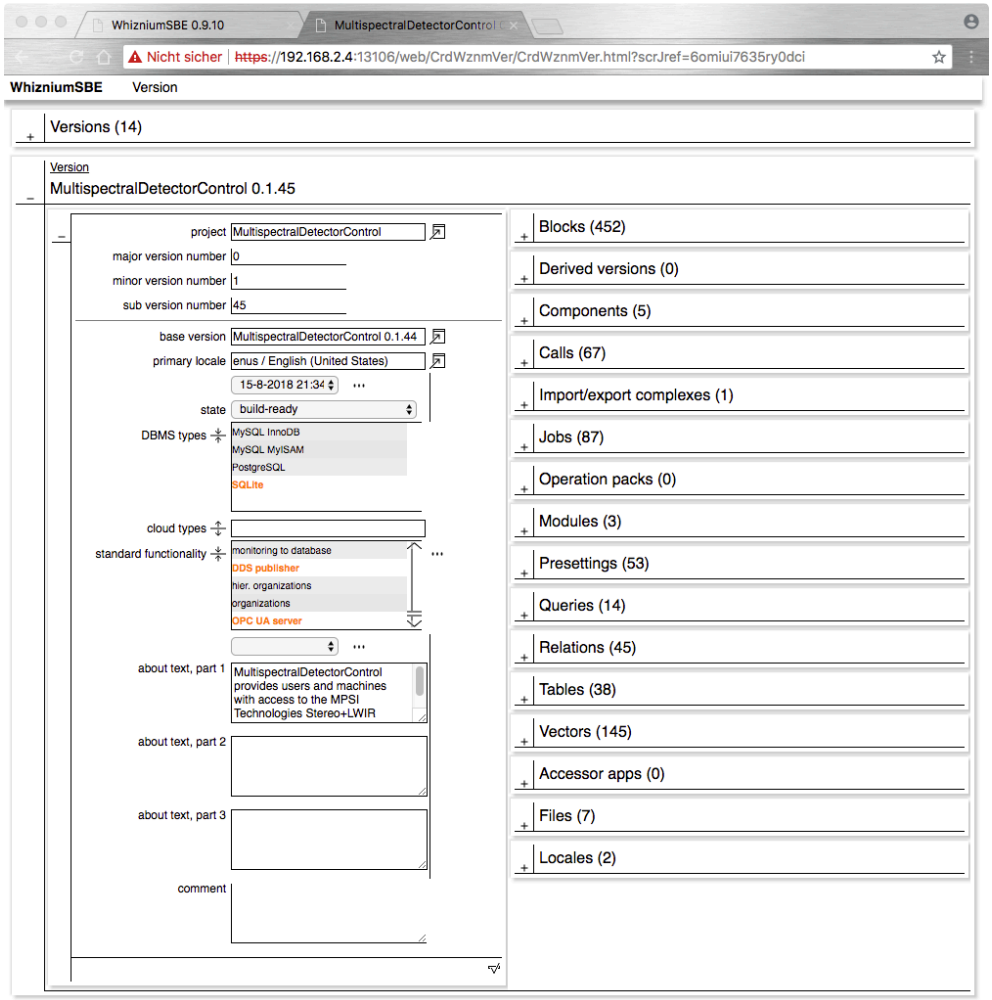

Figure 9: `Msdc` version 0.1.45 overview including DDS publisher and OPC UA server selected as standard functionality.

3.3 Preferences file

Complex embedded systems come with a great deal of parameters ; in WhizniumSBE, these can be attributed to specific hardware control functionality and their corresponding C++ classes (jobs). The auto-generated code takes care of automatically loading and storing all settings using a common XML preferences file.

WhizniumSBE's auto-generation feature for web-based UI's makes it particularly simple to establish a human-readable and -writable interface to settings data. For `Msdc`, this is illustrated in Figure 10.

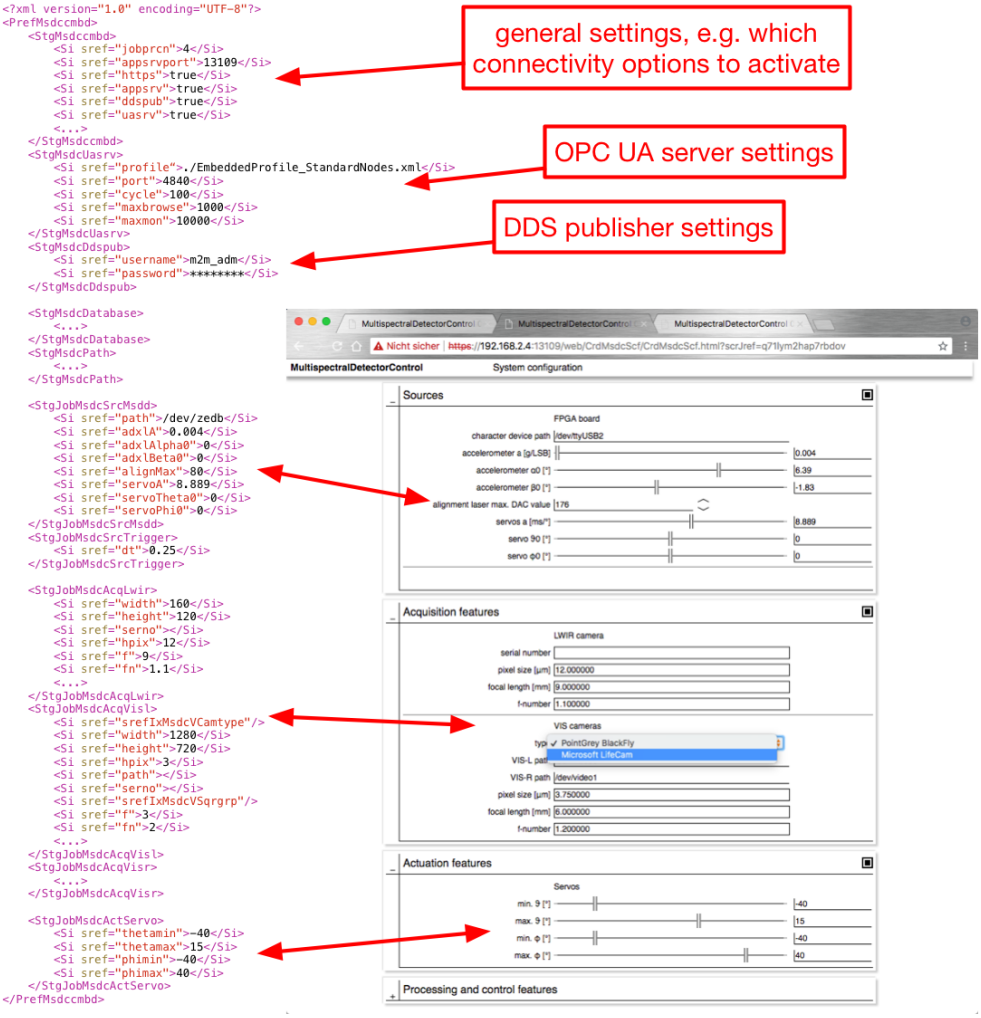

Figure 10: XML preferences file (left) along with web-based UI for job-specific settings (right).

3.4 The job tree

At run-time, each WhizniumSBE-powered project stores session state data in a hierarchical structure of objects, the job tree. In terms of code, each super-job `#include`'s its respective sub-jobs. A single root job `RootMsdc` manages the establishment of HMI and M2M session jobs, while the HMI/M2M session jobs `SessMsdc` and `M2msessMsdc` provide their sub-jobs with information about feature access rights.

All further jobs are either responsible for UI elements (cards - `CrdMsdc…`, panels - `PnlMsdc…`, database queries - `QryMsdc…` and dialogs - `DlgMsdc…`) or for hardware control (`JobMsdcAcq…/Act…/Src…`).

3.5 Master/slave jobs

In embedded systems, it is crucial that at any given time only one programmatical element, in this case a job, is in control of a specific hardware feature. For instance, it is not desirable to have the tracking algorithm `JobMsdcPrcTrack` trying to position the alignment laser spot (via `JobMsdcActServo`), while simultaneously a human operator is providing conflicting input via the web-based UI. Sensor data on the other hand may have multiple

recipients - a good example is the VIS-L acquisition job `JobMsdcAcqVisl`: image data can be provided to several HMI and M2M sessions for external processing, while `Msdc`-internally, VIS-L images are processed by the stereo (`JobMsdcPrcStereo`) and contour tracking (`JobMsdcPrcTrack`) algorithms.

WhizniumSBE provides the option to equip a job with master/slave functionality. This feature ensures non-conflicting control and multi-party access of hardware. Mutex-protected and seamless transitions of a classes' master control from one object to another is implemented within the auto-generated source code fabric. Master/slave jobs share common (in C++, `static`) data and application-internal calls (see below) ensure that updates by the master job are passed on to all respective slave jobs of the same class.

3.6 Calls and call listeners

Jobs within the job tree can communicate with one another by triggering and listening to calls. Calls may carry a limited amount of data as invocation and return arguments. A central feature in each WhizniumSBE-powered application ensures the matching of triggered calls to registered call listeners.

For example, the web-based UI's navigation card job `CrdMsdcNav` triggers a number of `CallMsdcCrdActive` calls to find out which navigation targets (cards) to show to / hide from the user. Based on user group/user access rights, these calls are answered by the session job `SessMsdc` up the job tree hierarchy with a view/edit/execute return argument.

Once the user chooses a navigation target to open as a new web-browser tab, e.g. a new "System configuration" card, `CrdMsdcNav` triggers a `CallMsdcCrdOpen` call to which again `SessMsdc` would reply with the new card job's reference, once the corresponding card job `CrdMsdcScf` is established successfully. Both call scenarios are highlighted in the job tree shown in Figure 11.

```
+ RootMsdc (1)
      ...
      + SessMsdc (21)
            CallMsdcCrdActive (tree)
            CallMsdcCrdClose (tree)
            CallMsdcCrdOpen (tree)
            CallMsdcLog (tree)
            CallMsdcRecaccess (tree)
            CallMsdcRefPreSet (tree)
            + CrdMsdcNav (22, dcol)
                  CallMsdcDlgClose (imm)
                  - PnlMsdcNavHeadbar (23)
                  - PnlMsdcNavPre (24)
                  - PnlMsdcNavAdmin (25)
                        CallMsdcHusrRunvMod.crdUsrEq (all)
                  - PnlMsdcNavOpr (26)
                        CallMsdcHusrRunvMod.crdUsrEq (all)
            + CrdMsdcScf (27, dcol)
                  CallMsdcDlgClose (imm)
                  + PnlMsdcScfSource (28)
                        ...
                  + PnlMsdcScfAcquis (31)
                        CallMsdcMastslvChg (imm)
                        + JobMsdcAcqAdxl/M (32)
                              ...
                              - JobMsdcSrcMsdd/S (33)
                                    ...
                              - JobMsdcSrcTrigger/S (34)
                                    ...
```

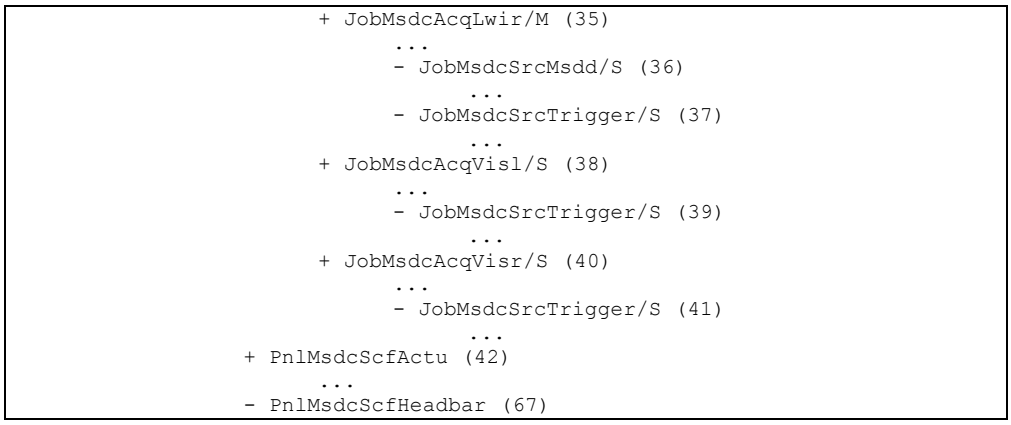

Figure 11: Partial job tree for a HMI session with navigation card and system configuration card opened. Call listeners in gray.

Calls are also used to notify the OPC UA server and DDS publisher of updated job variables.

3.7 Methods and variables

Hardware control jobs are shielded from external users in the web-based UI / generic connectivity scenario where all access is handled via UI jobs higher up the job tree, and XML data blocks are employed for HTTPS communication. However, M2M sessions used for OPC UA and DDS allow low-level access to all methods and variables specified in the specified in the application model.

The set of `Msdc` methods, along with their invocation and return parameters is listed in Figure 12 ; the set of variables is listed in Figure 13. For the latter, both shared and instance-specific data can be made visible to the outside.

```
JobMsdcAcqLwir:
        (boolean success) = setOutput(vecsref ixMsdcVSqrgrp)
        (boolean success) = start()
        (boolean success) = stop()
JobMsdcAcqVisl:
        (boolean success) = setFocus(float focus)
        (boolean success) = setOutput(vecsref ixMsdcVCamres, vecsref ixMsdcVSqrgrp,
                                      boolean grayscale)
        (boolean success) = setTint(float Tint)
        (boolean success) = start()
        (boolean success) = stop()
JobMsdcAcqVisr:
        (boolean success) = setFocus(float focus)
        (boolean success) = setOutput(vecsref ixMsdcVCamres, vecsref ixMsdcVSqrgrp,
                                      boolean grayscale)
        (boolean success) = setTint(float Tint)
        (boolean success) = start()
        (boolean success) = stop()
JobMsdcActAlign:
        (boolean success) = setWave(vecsref ixVFunction, utinyint N, floatvec seq)
        (boolean success) = start()
        (boolean success) = stop()
JobMsdcActLed:
        (boolean success) = setFlood(float flood)
        (boolean success) = setSpot(float spot)
JobMsdcActServo:
        (boolean success) = setPhi(float phi)
        (boolean success) = setTheta(float theta)
```

Figure 12: Multi-Spectral Detector Control methods by job.

```
JobMsdcAcqAdxl:
        {float alpha, float beta}
JobMsdcAcqLwir:
        {uint seqno, double t, usmallintvec gray16}
JobMsdcAcqVisl:
        {uint seqno, double t, utinyintvec rgbx8, utinyintvec gray8}
JobMsdcAcqVisr:
        {uint seqno, double t, utinyintvec rgbx8, utinyintvec gray8}
JobMsdcActLed:
        {float flood, float spot}
JobMsdcActServo:
        {float theta, float phi}
```

Figure 13: Multi-Spectral Detector Control variables by job.

3.9 Access rights management

Regardless of the connectivity option chosen, no access to WhizniumSBE-powered applications is provided to outside stakeholders without first establishing a session. Sessions are opened either using a username/password combination or a known X.509 certificate. The auto-generated database holds information about user groups and users, along with dedicated feature access rights at arbitrary granularity.

Features for web-based HMI and generic M2M sessions comprise all cards, whereas OPC UA and DDS M2M sessions use methods and variables as features for which credentials can be administered. Examples are shown in Figure 14.

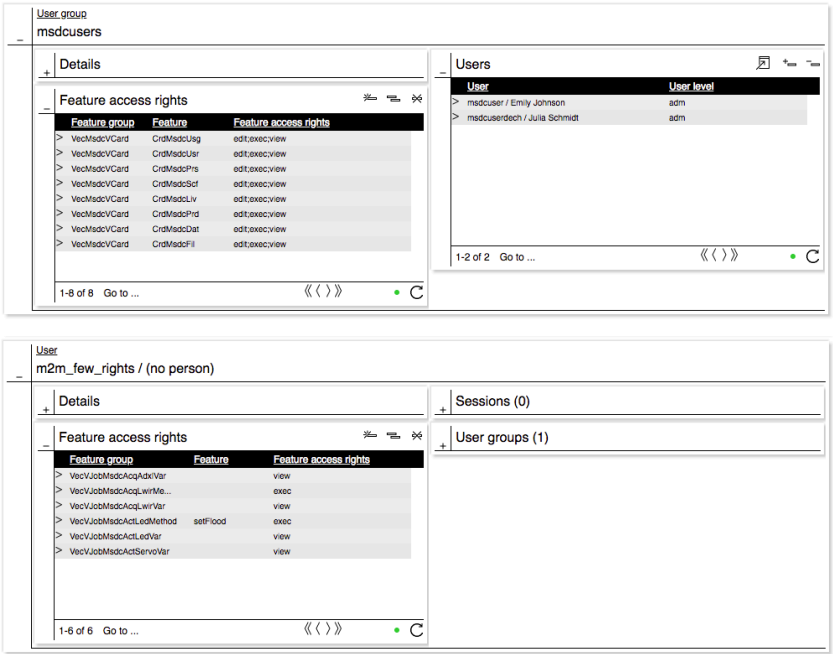

Figure 14: Definition of card (HMI) access rights by user group on top, definition of job method/variable (M2M) access rights for a specific user below.

4 Generic HTTPS/XML M2M Communication

All connectivity options that are based on the HMI or web-based user interface (UI) rely on exchanging the same type of XML blocks that a web browser would use in POST requests via HTTP/1.1. No direct hardware control job access is possible, as only root, session, card, panel and dialog jobs exchange dispatches (see 4.2).

4.1 XML data blocks

All data blocks fall into exactly one category of those listed in Figure 15. Serializers/deserializers from/to C++ objects are auto-generated by WhizniumSBE in accordance with the “direction of travel” (engine-app, app-engine, or both).

ContIacMsdcLivAlign	interactive content - written and updated by engine and app. This example: slider positions representing the servo positions on the “Live data” - “Alignment” panel.
ContInfMsdcLivAlign	informative content - written and updated by engine only. This example: tilt angle positions on the “Live data” - “Alignment” panel.
FeedFPupTyp	feed - written and updated by engine and app ; provides lists of indexed identifier / title / comment items. This example: entries for a pop-up button (Pup) based on a WhizniumSBE “type” vector .
StatAppMsdcLivAlign	app state - written once by engine, updated by app. This example: panel expanded/collapsed view state for the “Live data” - “Alignment” panel.
StatShrMsdcScfActu	shared state - written and updated by engine. This example: among others min/max bounds for sliders on the “System configuration” - “Actuation features” panel.
StgIacMsdcUsrList	interactive settings - written and updated by engine and app. This example: table column widths of the “Users” list panel.
TagMsdcLivAlign	tags - written once by engine ; written in accordance with the user’s locale/language. This example: control captions for the “Live data” - “Alignment” panel.

Figure 15: XML data block types with examples. The respective prefix in bold.

4.2 Dispatches

Dispatches are the XML entities that are transmitted between engine and app. They may contain a number of the XML data structured mentioned above. In addition, they carry a unique reference of the job they originate from / are addressed to. The job reference is passed in scrambled form, so as not to disclose details of the job tree.

Dispatches emitted by the server (main executable) are named `DpchEng…` whereas dispatches emitted by the client (web browser or accessor app) are named `DpchApp…`. For example, the dispatch `DpchEngMsdcLivVideoLive` delivers frames to the “Live data” -

"Video" panel and the dispatch `DpchAppMsdcLivServoData` is responsible for requesting a change of servo position from the "Alignment" panel.

4.3 Dispatch collectors and long polling

While some app-engine interactions are triggered by the client (UI / app) and warrant an immediate response by the engine in the request-reply scheme, there are many cases in which the server wants to communicate new information to the client on its own initiative. This behavior is not accounted for in the HTTP/1.1 protocol (this changes with HTTP/2.0, but HTTP/2.0 is not implemented in WhizniumSBE - yet).

Workarounds to emulate server-initiated communications include continuous polling by the client, however this method is detrimental in terms of bandwidth usage. In WhizniumSBE, the problem is solved by having the client emit a "notify" request which is answered by the server only once new engine dispatches become available for transmission - else the request times out and a new one is initiated. This method also is called long polling. A typical cycle time for long polling is 15 seconds.

Using long polling implies the engine-internal storage of engine dispatches to be transmitted. In web-browsers, one connection per tab or card is maintained - this is reflected in the job tree by attaching a dispatch collector object to each card job which collects all dispatches from down the job tree (panels, database queries, dialogs) until a new opportunity for communication becomes available. Any generic M2M solution on the other hand can work with a single dispatch collector which is attached to the session job rather than to a card job.

4.4 C++ and Java API libraries

The C++ and Java API libraries are collections of XML data block and XML dispatch serializer/deserializer objects. Source code file names and class names are the same as in the engine.

4.5 Accessor apps

To establish meaningful workflows of apps that access a WhizniumSBE-powered tool, not only the API library is required. Rather, in analogy to the web-browser's Document Object Model (DOM), some state data arriving via dispatches has to be stored and re-used client-side, and also the HMI-like behavior of "click that button" - "wait for this reaction" - "click the next button" - … has to be taken into account. Whiznium helps write the corresponding code based on a state machine that reflects the expected client workflow(s), including platform-specific networking. The accessor app feature is available for all important C++ flavors and Java.

5 OPC UA Powered By Matrikon® Flex OPC UA SDK

OPC UA is a widely adopted communication standard in industrial automation. It is employed to ensure interoperability between devices from different vendors and covers everything from simple sensors to enterprise IT systems.

To equip a WhizniumSBE project with OPC UA server functionality, a customer has to purchase an additional Matrikon® Flex OPC UA SDK license. WhizniumSBE uses the project model description to generate OPC UA server source code which builds on the Matrikon® Flex OPC UA SDK C++ API.

No OPC UA specific modelling is required, as WhizniumSBE methods and variables are mapped 1:1 to a corresponding OPC UA address space, as illustrated in Figure 16: here, every WhizniumSBE job becomes a folder as organizational unit. The illustration also is evidence for a 1:1 mapping of sessions and user access rights between WhizniumSBE and OPC UA.

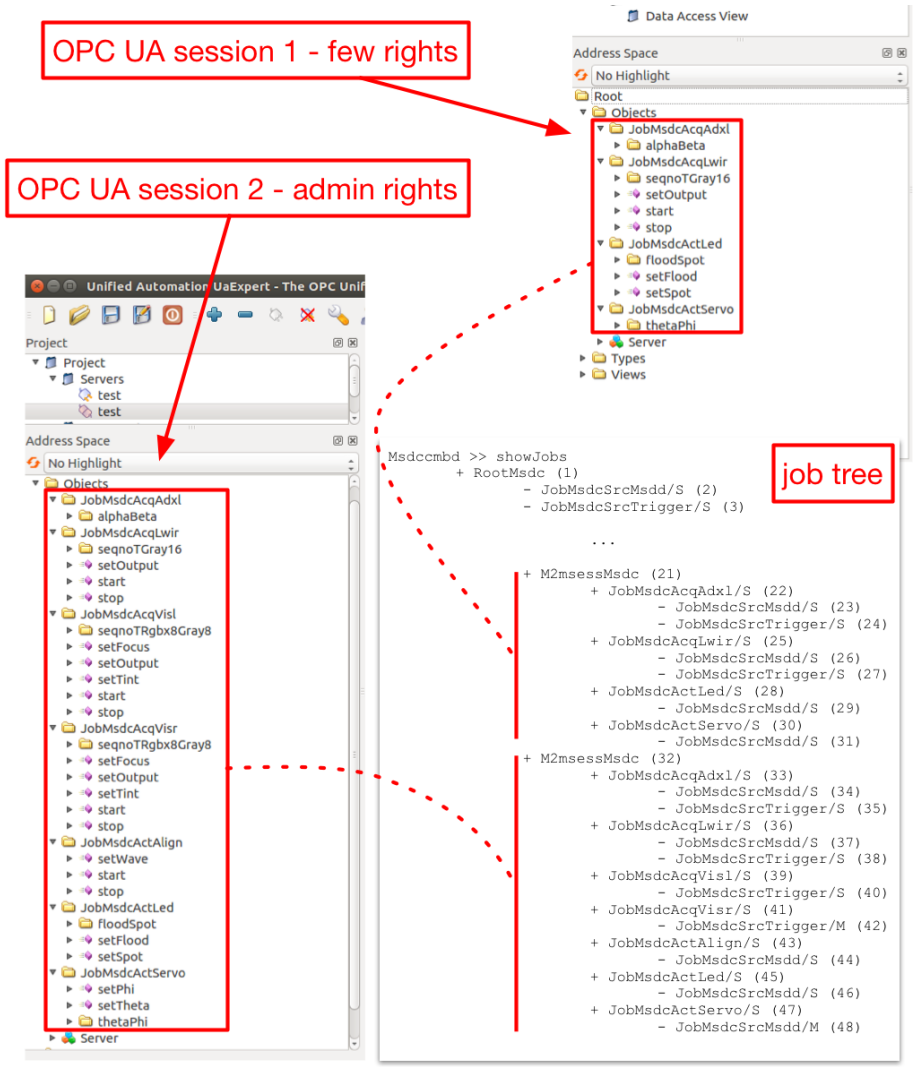

Figure 16: OPC UA address space for two different sessions and their job tree counterpart.

6 DDS Powered By RTI® Connext™ DDS Professional Software

DDS relies on the principle of a “databus” and it is used to share time- and mission-critical data between numerous “participants” in connected systems. DDS is not a classical client-server architecture, and the complexity of actual transportation of data between nodes is hidden away from the user.

To equip a WhizniumSBE project with DDS publisher functionality, a customer has to purchase an additional RTI® Connext™ Professional license. WhizniumSBE uses the project model description to generate IDL code that is in turn interpreted by RTI’s code generator. Additional WhizniumSBE-generated source code establishes request/repliers for each method and DDS publishers for each variable, both based on the RTI® Connext™ Professinal “modern C++” API.

No DDS specific modelling is required as WhizniumSBE methods and variables are mapped 1:1 to corresponding DDS topics. Figure 17 shows the traffic on the databus the moment `Msdccmbd` starts up. Currently only one WhizniumSBE session with dedicated user access rights is launched on startup of the DDS publisher.

```
RTI Connext DDS Spy built with DDS version: 5.3.1 (Core: 1.8a.00, C: 1.8a.00, C++: 1.8a.00)
Copyright 2012 Real-Time Innovations, Inc.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
rtiddsspy is listening for data, press CTRL+C to stop it.

source_timestamp   Info  Src HostId  topic               type
-----------------  ----  ----------  ------------------  ------------------
1534370378.496586  W +N  C0A80204    JobMsdcAcqLwir.set  DdsJobMsdcAcqLwir:
                                 ...  OutputReply         :setOutput_reply
1534370378.498309  W +N  C0A80204    JobMsdcAcqLwir.sta  DdsJobMsdcAcqLwir:
                                 ...  rtReply             :start_reply
1534370378.499605  W +N  C0A80204    JobMsdcAcqLwir.sto  DdsJobMsdcAcqLwir:
                                 ...  pReply              :stop_reply
1534370378.497619  R +N  C0A80204    JobMsdcAcqLwir.set  DdsJobMsdcAcqLwir:
                                 ...  OutputRequest       :setOutput_req
1534370378.498973  R +N  C0A80204    JobMsdcAcqLwir.sta  DdsJobMsdcAcqLwir:
                                 ...  rtRequest           :start_req
1534370378.500266  R +N  C0A80204    JobMsdcAcqLwir.sto  DdsJobMsdcAcqLwir:
                                 ...  pRequest            :stop_req
1534370378.501321  R +N  C0A80204    JobMsdcAcqVisl.set  DdsJobMsdcAcqVisl:
                                 ...  FocusRequest        :setFocus_req
1534370378.501923  R +N  C0A80204    JobMsdcAcqVisl.set  DdsJobMsdcAcqVisl:
                                 ...  OutputRequest       :setOutput_req
1534370378.500881  W +N  C0A80204    JobMsdcAcqVisl.set  DdsJobMsdcAcqVisl:
                                 ...  FocusReply          :setFocus_reply
1534370378.501587  W +N  C0A80204    JobMsdcAcqVisl.set  DdsJobMsdcAcqVisl:
                                 ...  OutputReply         :setOutput_reply
1534370378.502407  W +N  C0A80204    JobMsdcAcqVisl.set  DdsJobMsdcAcqVisl:
                                 ...  TintReply           :setTint_reply
1534370378.503025  W +N  C0A80204    JobMsdcAcqVisl.sta  DdsJobMsdcAcqVisl:
                                 ...  rtReply             :start_reply
1534370378.503998  W +N  C0A80204    JobMsdcAcqVisl.sto  DdsJobMsdcAcqVisl:
                                 ...  pReply              :stop_reply
1534370378.502719  R +N  C0A80204    JobMsdcAcqVisl.set  DdsJobMsdcAcqVisl:
                                 ...  TintRequest         :setTint_req
1534370378.503327  R +N  C0A80204    JobMsdcAcqVisl.sta  DdsJobMsdcAcqVisl:
                                 ...  rtRequest           :start_req
1534370378.504812  R +N  C0A80204    JobMsdcAcqVisl.sto  DdsJobMsdcAcqVisl:
                                 ...  pRequest            :stop_req
1534370378.505715  W +N  C0A80204    JobMsdcAcqVisr.set  DdsJobMsdcAcqVisr:
                                 ...  FocusReply          :setFocus_reply
1534370378.506460  R +N  C0A80204    JobMsdcAcqVisr.set  DdsJobMsdcAcqVisr:
                                 ...  FocusRequest        :setFocus_req
1534370378.507210  W +N  C0A80204    JobMsdcAcqVisr.set  DdsJobMsdcAcqVisr:
                                 ...  OutputReply         :setOutput_reply
```

```
1534370378.508057  R +N  C0A80204    JobMsdcAcqVisr.set  DdsJobMsdcAcqVisr:
                                ...  OutputRequest       :setOutput_req
1534370378.508910  W +N  C0A80204    JobMsdcAcqVisr.set  DdsJobMsdcAcqVisr:
                                ...  TintReply           :setTint_reply
1534370378.509663  R +N  C0A80204    JobMsdcAcqVisr.set  DdsJobMsdcAcqVisr:
                                ...  TintRequest         :setTint_req
1534370378.510515  W +N  C0A80204    JobMsdcAcqVisr.sta  DdsJobMsdcAcqVisr:
                                ...  rtReply             :start_reply
1534370378.511250  R +N  C0A80204    JobMsdcAcqVisr.sta  DdsJobMsdcAcqVisr:
                                ...  rtRequest           :start_req
1534370378.511986  W +N  C0A80204    JobMsdcAcqVisr.sto  DdsJobMsdcAcqVisr:
                                ...  pReply              :stop_reply
1534370378.512490  R +N  C0A80204    JobMsdcAcqVisr.sto  DdsJobMsdcAcqVisr:
                                ...  pRequest            :stop_req
1534370378.512852  W +N  C0A80204    JobMsdcActAlign.se  DdsJobMsdcActAlign
                                ...  tWaveReply          ::setWave_reply
1534370378.513731  W +N  C0A80204    JobMsdcActAlign.st  DdsJobMsdcActAlign
                                ...  artReply            ::start_reply
1534370378.513375  R +N  C0A80204    JobMsdcActAlign.se  DdsJobMsdcActAlign
                                ...  tWaveRequest        ::setWave_req
1534370378.514084  R +N  C0A80204    JobMsdcActAlign.st  DdsJobMsdcActAlign
                                ...  artRequest          ::start_req
1534370378.514954  R +N  C0A80204    JobMsdcActAlign.st  DdsJobMsdcActAlign
                                ...  opRequest           ::stop_req
1534370378.514500  W +N  C0A80204    JobMsdcActAlign.st  DdsJobMsdcActAlign
                                ...  opReply             ::stop_reply
1534370378.515503  W +N  C0A80204    JobMsdcActLed.setF  DdsJobMsdcActLed::
                                ...  loodReply          setFlood_reply
1534370378.516085  R +N  C0A80204    JobMsdcActLed.setF  DdsJobMsdcActLed::
                                ...  loodRequest        setFlood_req
1534370378.516719  W +N  C0A80204    JobMsdcActLed.setS  DdsJobMsdcActLed::
                                ...  potReply           setSpot_reply
1534370378.517320  R +N  C0A80204    JobMsdcActLed.setS  DdsJobMsdcActLed::
                                ...  potRequest         setSpot_req
1534370378.517784  W +N  C0A80204    JobMsdcActServo.se  DdsJobMsdcActServo
                                ...  tPhiReply          ::setPhi_reply
1534370378.518239  R +N  C0A80204    JobMsdcActServo.se  DdsJobMsdcActServo
                                ...  tPhiRequest        ::setPhi_req
1534370378.518765  W +N  C0A80204    JobMsdcActServo.se  DdsJobMsdcActServo
                                ...  tThetaReply        ::setTheta_reply
1534370378.519470  W +N  C0A80204    JobMsdcAcqAdxl.alp  DdsJobMsdcAcqAdxl:
                                ...  haBeta              :alphaBeta
1534370378.519125  R +N  C0A80204    JobMsdcActServo.se  DdsJobMsdcActServo
                                ...  tThetaRequest      ::setTheta_req
1534370378.520219  W +N  C0A80204    JobMsdcActLed.floo  DdsJobMsdcActLed::
                                ...  dSpot              floodSpot
1534370378.520543  W +N  C0A80204    JobMsdcActServo.th  DdsJobMsdcActServo
                                ...  etaPhi              ::thetaPhi
```

Figure 17: RTI® Connext™ DDS Spy output on `Msdccmbd` launch.

7 Connectivity Comparison Chart

Feature	**Generic**	**OPC UA**	**DDS**
Structure	card/panel/dialog (UI element) based	M2M session mapped to OPC UA address space	M2M session mapped to DDS topics
Transfer protocol	HTTPS	OPC UA via TCP	DDS via UDP/TCP
Transfer content	textual/XML ; binary data Base64 encoded	binary	binary
Access rights	card (UI element) based	method/variable based	method/variable based
Sessions	user name / password	user name / password, X.509 certificate	N/A ; future: RTI® Connext™ DDS Secure software
Variable update (READ ONLY!)	polling, long polling	polling only ; future: OPC UA pub/sub	DDS publish/subscribe
Method invocation	simulated click in web-based UI equivalent	OPC UA method handler	DDS request/reply

Figure 18: Comparison of concepts and features for generic vs. OPC UA vs. DDS connectivity options.

Appendix A: Multi-Spectral Detector FPGA-Based Sub-System

The detector's FPGA sub-system is implemented as a WhizniumDBE project "Multi-Spectral Detector Device" `(Msdd)` in "easy" mode, implying serial/blocking communication. For increased hardware flexibility, two functionally equivalent units are available: ZedBoard `(Zedb)` - Zynq via internal AXI bus, and Basys3 `(Bss3)` via USB-UART bridge.

Within the host system software, the `JobMsdcSrcMsdd` class handles all interaction with the FPGA sub-system. The implemented controllers along with their commands and buffer transfers are listed in Figure 19.

```
adxl: accelerometer read-out (ADXL345 via SPI)
        (smallint ax) = getAx()
        (smallint ay) = getAy()
        (smallint az) = getAz()
align: alignment laser control (MAX5385 DAC via SPI)
        () = setSeq(vblob seq)
led: high power LED control (LT3474 PWM)
        () = setTon15(utinyint ton15)
        () = setTon60(utinyint ton60)
lwirif: LWIR camera control (Lepton3 via I2C)
        () = setRng(_bool rng)
lwiracq: LWIR camera acquisition (Lepton3 via SPI)
       <- abufLwiracqToHostif (38kB)
       <- bbufLwiracqToHostif (38kB)
        () = setRng(_bool rng)
        (tix tixVBufstate, uint tkst, usmallint min, usmallint max) = getInfo()
servo: servo control (HS422 PWM)
        () = setTheta(smallint theta)
        () = setPhi(smallint phi)
state: state monitor
        (tix tixVZedbState) = get()
tkclksrc: 10kHz clock source
        (uint tkst) = getTkst()
        () = setTkst(uint tkst)
trigger: trigger source
        () = setRng(_bool rng, _bool btnNotTfrm)
        () = setTdlyLwir(usmallint tdlyLwir)
        () = setTdlyVisr(usmallint tdlyVisr)
        () = setTfrm(usmallint Tfrm)
vgaacq: VGA camera acquisition (ucam via UART)
       <- abufVgaacqToHostif (38kB)
       <- bbufVgaacqToHostif (38kB)
        () = setRng(_bool rng)
        (tix tixVBufstate, uint tkst) = getInfo()
```

Figure 19: Multi-spectral Detector Device command set and available buffer transfers by controller

Link to code: https://github.com/mpsitech/MultiSpectralDetectorDevice

`_mdl/msdd`	model files ; information to be processed by WhizniumDBE
devmsdd	C++ source files for the device access library
`_rls/devmsdc_*`	shell scripts and make files to perform builds on several platforms
msdd/bss3 **msdd/zedb**	VHDL source files for Basys3 (USB-UART bridge) and ZedBoard (AXI interconnect)

Figure 20: Project folder overview.

HIGH-END EXAMPLE ACTIVE COMPONENTS

END - DEVICES	**accelerometer** ADXL345 via SPI	**15°/60° high-power LED's** LT3474 constant current + PWM	low-resolution camera µCam-III via SPI	**1mW modulated red laser** MAX5385 via SPI	dual HD webcams LifeCam via USB
	status LED's	**tilt-pan unit** angle by PWM	low-resolution thermal imager Lepton3 via SPI and I2C	**high-resolution thermal imager** Tau2 via LVDS	**dual 1.2MP cameras** BlackFly via Ethernet

FPGA	ZedBoard Zynq Programmable Logic Xilinx Zynq XC7Z020 28nm Silicon 85k logic cells 4900 Kb BlockRAM				**Mercury KX1 FPGA Board** Xilinx Kintex-7 XC7K160T 28nm Silicon 162k logic cells 11700 Kb BlockRAM		**Gigabit Ethernet Switch**
INTERFACE	AXI		USB-UART	**PCIe**	SPI		Active USB Hub

EMBEDDED COMPUTER	**Avnet ZedBoard** ARM Cortex-A9 dual 677MHz 512MB DDR3	**RaspberryPi3** ARM Cortex-A53 quad 1.2GHz 1GB LPDDR2	**gumstix Overo** ARM Cortex-A8 800MHz 512MB LPDDR	**Minnowboard Turbot** Intel Atom E3826 dual 1.46GHz 2GB LPDDR3	**Intel Galileo** Intel Quark X1000 400MHz 256MB DDR3