SDK introduction & integration

During the project, we integrated 3 different SDKs to wmap_platform build system. Work started on top of wmap_platform master branch, commit **4c6ba3**. All commits can be found in branch **feature-ci/OPCUA**.

The cmake build system of wmap_platform was under work during the project, so we didn't do any rebases from master during the project. The build system related changes in the branch are most likely outdated by now and must be refactored when rebased to latest master in future.

All the commits went through pre-commit and very light weight review process, they are in PoC level and not all of the comments given by WMAP developers were addressed during the project.

The source codes for all of the SDKs can be found in **http://im.bitbucket.wartsila.com/users/stu008/repos/opcua_poc/browse**. Ask permission to the repository from Tunis, Staffan.

OPC-UA abstraction layer in wmap_platform

An abstraction layer was implemented to ease the integration work of the different SDKs to wmap_platform. The abstraction layer sources are located in sys/comm/eth/opcua. opcua_al.h lists the functions that the SDK layer must implement.

The abstraction layer handles:

- Initializing the CPS records
- Creating tasks for server and client and calling the respective initialization functions
- Looping through all DC codes and registering them to the server address space
- Handling periodical scheduler operations such as writing for the client and providing monitored items for the server.

The used SDK in wmap_platform can be selected in sys/cmake/externals.cmake by changing the value of OPCUA_STACK variable. The prebuilt libraries for the SDKs are stored in sys/lib/opcua.

OPC-UA related file locations under wmap_platform

Path	Description
sys/lib/matrikonflex	Prebuilt Matrikon FLEX SDK libraries.
sys/lib/mbedtls	Prebuilt mbed TLS library. Needed by open62541 SDK.
sys/lib/opcuahp	Prebuilt UA high performance SDK libraries.
sys/lib/open62541	Prebuilt open62541 SDK libraries.
sys/code/comm/eth/opcua	Source codes for the main OPC-UA implementation.
sys/code/comm/eth/opcua/stacks	Abstraction layer implementations for the different SDKs.
sys/cmake/install_files/ns0.bin	Namespace 0 definition used by UAHP SDK.
sys/cmake/install_files/hpsdk2048.*	Certificate used by UAHP and open62541 SDKs.
sys/cmake/install_files/EmbeddedProfile_StandardNodes.xml	Namespace 0 definition used by Matrikon Flex SDK.
sys//cmake/install_files/application_rsa_sha256*	Certificate used by Matrikon Flex SDK.

open62541

open62541 is an open source OPC-UA SDK, the source code is available in https://github.com/open62541/open62541. In the PoC we forked the master branch from commit 75dab1 to https://github.com/lkaino/open62541 (owned by Jaakko Laurikainen). The github repository contains all the changes we made to the stack, the same changes were also included in the Bitbucket repo as patches which can be applied to the cloned repository. We chose not to include the source of the SDK in the Bitbucket because we would have lost the commit history and later rebasing from the mainline would not be possible.

The SDK supports running the server and clients in their own separate threads. It uses regular malloc to dynamically allocate memory during runtime,

Integration to wmap_platform

open62541 can be integrated in two ways: amalgamated single source file or as a statically linked library. We tested both methods and ended up using the static library as it is not necessary to rebuild the SDK every time the wmap_platform is built. The mbedtls library is also statically linked to the open62541 library, which might not be the best way in future if we want to use the mbedtls for other purposes also.

The integration work started by creating an architecture for Xenomai (https://github.com/lkaino/open62541/commit/a452cd13d4e6899b7115bfa969d435d027264b2d). This was not necessary step, we could have used the basic Linux architecture, but this allowed us to modify e.g. the timestamping methods in the stack. In future we could implement for example the Xenomai mutexes in the architecture.

Two build scripts were added to the root folder for building the SDK library for HW and virtual modules https://github.com/lkaino/open62541/commit/aacd4f0762a02d2fb75915305d1ebd4477d5ab4f.

To use the certificates and encrypted communication, the SDK requires the mbedtls library to be available during building. As we have not built this library for the modules before, we included it in the Bitbucket repo and created build scripts for it to build it for both HW and virtual modules. The mbed TLS does not build out of the box for virtual targets, so we found and backported a PR from the mainline branch to the version we are using: https://github.com/ARMmbed/mbedtls/pull/1986.

Building the SDK

1. navigate to mbedtls-2.14.0 folder in the Bitbucket repo and run commands *./build_unic2_hw* and *./build_unic2_virtual*.
2. follow the instructions in stacks/open62541/HowTo.txt.

Unified Automation High Performance OPC UA SDK

The SDK is delivered in plain source code and contains implementation for both server and client. The version used is **1.2.1-203**.

The SDK is quite new, not all features have been implemented or finalized. It seems to be made for small embedded applications, as the whole SDK, both server and clients run, on single thread. The single threaded approach is not good for prioritization between the server and clients, basically it is not possible.

The SDK implements it's own memory allocator, which allocates memory from a statically allocated area of memory, which size can be configured compile time. It implements built-in memory leak detections. Using a custom allocator is safer in theory, depending on how well the allocator has been implemented and tested. At least it should prevent the SDK from eating up all the RAM in the process.

The SDK uses OpenSSL for encryption, which we already have available on the modules.

Integration to wmap_platform

UA HP SDK required much more work to integrate, changes to the build system were necessary for it be built and run on UNIC targets. The work started by creating target configurations and toolchain definitions for unic2 hw and virtual targets (http://im.bitbucket.wartsila.com/users/stu008/repos/opcua_poc/commits/e82268eed273053bad739e24005e26ef6af4ceae).

The target configurations are located in the root folder, called *target_config_unic2_hw* and *target_config_unic2_virtual*. These files define the build and destination folders, the toolchain file locations and CMake build options. Any additional build options like memory tracing need to be enabled in these files. Currently we use options CONFIG_MULTITHREADED and CONFIG_THREADSAFE. Contrary to the name of the options, these do not allow the use of separate threads, but implements a locking mechanism in the memory allocator. This allows the user to implement some parts like "encoder" to be implemented in a separate process which would be run on a separate CPU core. Later though, by our request, UA implemented a "fix" to the server side which allows us to produce the data for the monitored items in a separate threads when these compilation options are defined.

The toolchain definitions are located in *toolchains/unic2/toolchain-unic2-hw.cmake* and *toolchains/unic2/toolchain-unic2-virtual.cmake*. These files use the toolchain definition from wmap_platform, and assume that the wmap_platform is located in path /media/projects/wmap_platform.

By default the SDK produces a library called libutil.so, which has same name with the standard library https://www.daemon-systems.org/man/libutil.3.html. This was problematic in HW modules where the standard library was dynamically linked in instead of the SDK one. There might be other ways to solve this, but we ended up changing the name of the library in the SDK to libuautil.so.

Similarly to open62541, the SDK contains platform source files which can be tailored for the OS and platform. We modified the existing linux platform where we implemented UNIC versions of the time getters: http://im.bitbucket.wartsila.com/users/stu008/repos/opcua_poc/commits/30a734f7379d69fa0f11b9476fe55f7a79c50f0f.

Later when implementing the client we realized that the SDK had multiple bugs when running on our PowerPC big-endian system. The SDK implementation has taken the big-endian systems into account, but it seems that they haven't tested it on big-endian systems at all. We managed to fix the bugs we found in http://im.bitbucket.wartsila.com/users/stu008/repos/opcua_poc/commits/06e463846b18883d372b3bca13a951fb0e7fb5da.

All in all the integration was a lot of work, with the build system and the implementation requiring fixes by us.

Building the SDK

1. Navigate to sdk folder.
2. Issue commands *./build.sh -t unic2_hw -T -s* and *./build.sh -t unic2_virtual -T -s*, where
    a. -t defines the target
    b. -T disables the unit testing
    c. -s strips the output libraries. If you ever need to investigate a crash inside the SDK, rebuild the SDK without the -s parameter. Then load the libs in gdb using *set solib-search-path /media/projects/opcua_poc/stacks/uasdkhpbundle-src-linux-v1.2.1-203/sdk/distPowerPC/lib*. After that run command*thread apply all bt*.
3. Copy the compiled libraries to wmap_platform

a. *cp distPowerPC/lib/*.so /media/projects/wmap_platform/sys/lib/opcuahp/hw*
b. *cp distx86/lib/*.so /media/projects/wmap_platform/sys/lib/opcuahp/virtual*

Matrikon Flex SDK

Sources

Matrikon stack is built from configuration based on multi threaded C++ Linux example. Compilation uses makepp and the build scripts and make files are located in stacks/flex_R410_1/Examples/Linux/MAKEPP_stack/opcua_server_library.

There are two versions of sources (Matrikon stack + platform): using default threading (phtreads) and another using Xenomai. Matching versions of the stack and platform must be build together. In addition to threads also mutexes, condition variables and timer functions are separate.

Default threading

This build uses Matrikon default cpp11 threading and mutexes.

Stack sources are in BitBucket repository http://im.bitbucket.wartsila.com/users/stu008/repos/opcua_poc/browse (clone link ssh://git@im.bitbucket.wartsila.com:7999/~stu008/opcua_poc.git) in the master branch. Matrikon stack sources are in folder stacks/flex_R410_1/Examples/Linux/MAKEPP_stack.

Platform sources are in Gerrit feature branch feature-ci/OPCUA of wmap_platform.

Xenomai threading

This build replaces cpp11 threads, locking and timers with matching UNIC functionality.

Stack sources are in BitBucket repository http://im.bitbucket.wartsila.com/users/stu008/repos/opcua_poc/browse (clone link ssh://git@im.bitbucket.wartsila.com:7999/~stu008/opcua_poc.git) in the branch feature/matrikon_unic_threads.

Platform sources are in Gerrit review https://fis-rd-dev.accdom.for.int/gerrit/#/c/29558/, branched from feature-ci/OPCUA.

In Xenomai build following classes are removed from the stack library and are instead implemented in the platform side (under sys/code/comm/eth/opcua/stacks/matrikonflex/unic_impl):
- async_operation_t: std::mutex replaced with UNIC_OS_MUTEX.
- lock_t: std::mutex replaced with UNIC_OS_MUTEX.
- system_calls: collection of functions, replaced time functions UASDK_millisecond and UASDK_datetime.
- thread_pool_t: replaced threads, mutexes and condition variables with UNIC versions.
- thread_t: replaced std::thread with UNIC_OS_TASK.
- timer_provider_t: replaced implementation using std::thread and std::chrono with version using Thread_t and UASDK_millisecond.

New header has been created for each class (in the unic folder).

Configuration

Build configuration settings can be changed in the configuration header uasdk_custom_build_config.h. This header is located in both the stack repository and the platform repository and the contents should match:

- Stack header: stacks/flex_R410_1/Examples/Linux/MAKEPP_stack/SDK_Configuration/uasdk_custom_build_config.h
- Platform header: sys/code/comm/eth/opcua/stacks/matrikonflex/uasdk_custom_build_config.h

With hardware modules the setting UASDK_USE_SYSTEM_HEAP should be set to 0, otherwise the stack will consume all memory at startup.

Note that the endian setting is different for virtual and hardware modules (big endian for virtual and little endian for hardware).

Matrikon stack has to be selected as the used OPC-UA stack in externals.cmake file:

```
set(OPCUA_STACK matrikon)
```

Additional non-compiled files that are required during runtime are put to folder sys/cmake/install_files:
- application_rsa_sha256_key.pem: private key for the server certificate. This will be copied to app/etc/pki/DefaultApplicationGroup/own/certs/.
- application_rsa_sha256.der: certificate for the server. This will be copied to app/etc/pki/DefaultApplicationGroup/own/private.
- EmbeddedProfile_StandardNodes.xml: profile XML file

Server will create the certificate automatically if the certificate / key files are present.

Building

Stack must be build and copied to platform folder before building the platform.

Requirements:
- makepp tool (make replacement)
- openssl development headers

Stack headers are copied from the stack's header folder stacks/flex_R410_1/SDK_Source to platform header folder sys/code/comm/eth/opcua/stacks/matrikonflex/server. If changes are made to the headers, the new versions need to be copied manually.

Hardware modules

Execute stacks/flex_R410_1/Examples/Linux/MAKEPP_stack/opcua_server_library/build_hw.sh and copy output from .build_hw folder to sys/lib/matrikonflex/hw/libopcua.so.

Virtual modules

Execute stacks/flex_R410_1/Examples/Linux/MAKEPP_stack/opcua_server_library/build_virtual.sh and copy output from .build_virtual folder to sys/lib/matrikonflex/virtual/libopcua.so.

Installation

The Matrikon stack shared library is too large to fit into the /app folder on the module, instead, it is installed to folder /data/lib and that folder is also added to RPATH so that the shared library is found during runtime. Installation script clears this folder during installation prior to copying the new files.

Certification folder PKI is set to /app/etc/pki, though this might cause issued with incorrectly detected application configuration and better place might be under /data/lib or some other location.

Customization

Customizing the behavior of the server is done by implementing classes that implemented certain interfaces (e.g. INodeValueAttributeReaderWriter_t to implemented value reading and writing) and is pretty clear to use.

Implementing custom target for the stack (e.g. custom threading) was tedious and requires lot of manual work with make files and copying the files. This could be alleviated e.g. by compiling the stack as part of the platform compilation though compiling the stack takes a while.

Evaluation

Some notes about the server. Client was not tested.

Test environment:

- Max monitored items 5000 / 1000
- Max browse refs per node 1000 / 100
- Server cyclic rate 50
- With secure (Basic256Sha256, no user authentication) and non secure connection
    - Security provided by openssl library.
- 32MB memory in the custom memory pool
- 10 / 2 threads