SSHMessageHandler

Design description document

Version 1.0

Document log

Date	Version	Summary of changes	Author
2005-10-19	1.0	Draft version	Wojciech Buczak
2005-10-21	1.0	Revision	Jan Stowisek
2005-11-11	1.0	Corrections after revision	Wojciech Buczak

Table of contents

1. Introduction

1.1 Document scope

This document introduces the SSHMessageHandler – a dedicated equipment control component for TIM. The component's internal structure is presented, the communication protocols are specified, and all relevant fields of the SSH process XML configuration are defined.

1.2 Definitions, Acronyms and Abbreviations

TIM – ***T****echnical* ***I****nfrastructure* ***M****onitoring* for the LHC project. Please refer to http://ts-project-tim.web.cern.ch/ts-project-tim for detailed information.

DAQ – *Data Acquisition Process* – a standalone process handling communication between different (hardware and software) data sources and TIM system's business layer components.

EMH – *Equipment Message Handler* – component of the TIM DAQ layer, implementing specific protocol to the type of device(s) or software it is talking to in the in the HL.

SSHMH - SSHMessageHandler

HL – *Hardware Layer* – the so-called hardware layer represents all possible devices that all TIM.DAQs communicate with, utilizing various types of EMH components. For TIM, the HL does not always mean low level devices, such as PLC controllers, but all possible types of data sources

DT – TIM *DataTag* [1][2]

CT – TIM *CommandTag*[1][2]

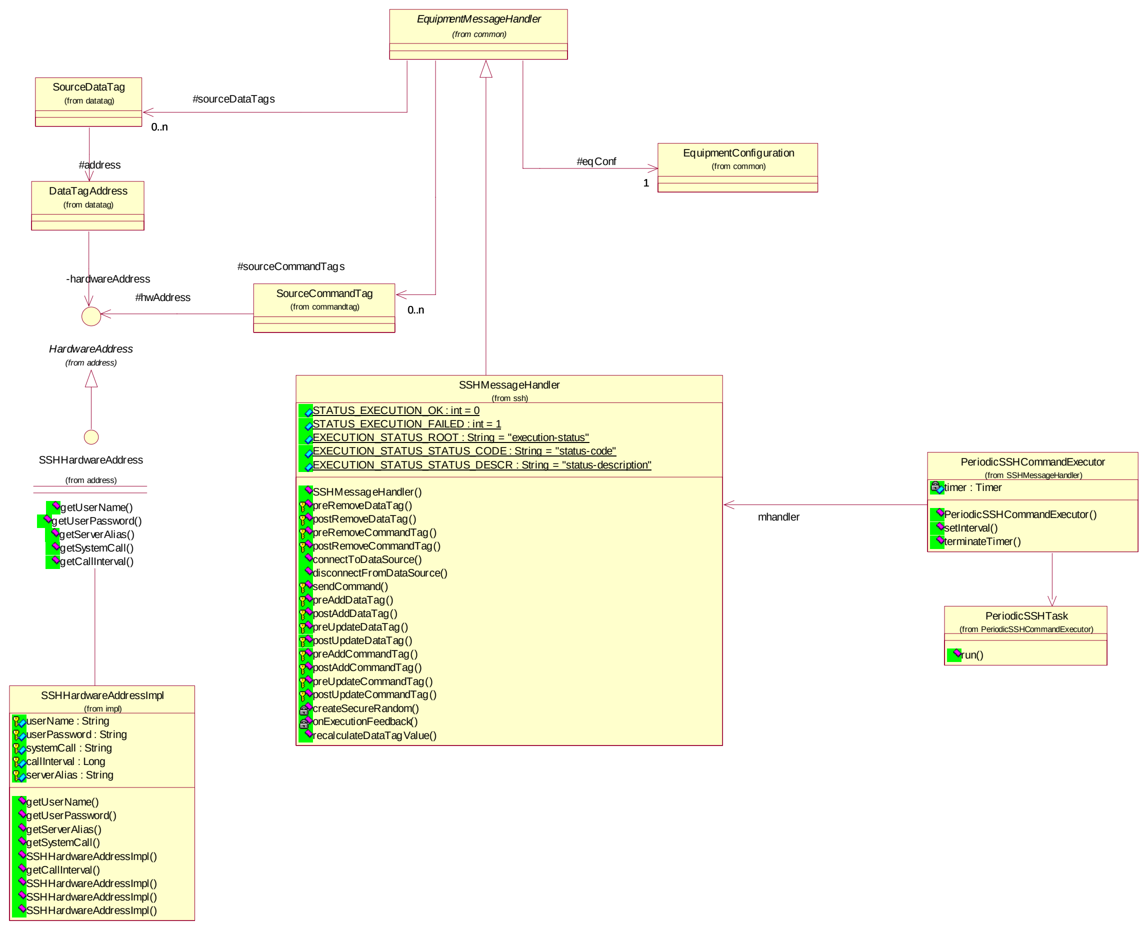
1.3 Document overview

The document is divided into sections describing different aspects of the SSHMH design.
It is not the issue of this document to present neither the TIM DAQ detailed architecture nor the EMH implementation details. For this the reader is strongly encouraged to refer to [1] and [2].

2. SSHMessageHandler generic description

The SSHMessageHandler (SSHMH) is a dedicated TIM EMH component, used for communication with various HW components as well as some of the TIM business-layer systems (e.g. JMS brokers, application server) using SSH v.2 protocol. The handler handles both - DTs and CTs.
However, unlike other EMHs, SSHMH does not treat the DTs in an “event driven” mode, but periodically recalculates the values of the supervised DTs. The value is recalculated on basis of the execution feedback status of the ssh system call associated with the considered tag. Hence, the DTs of the SSHMH are in fact a kind of commands executed in regular intervals. The execution intervals are defined individually for each DT and this information is wrapped inside DT hardware address blocks.

2.1 Class diagram



Like other TIM equipment components, the SSHMH implements the standard EMH interface. All classes and interfaces for SSH hardware address representation are provided by the *tim-shared* toolkit library.

The detailed explanation of the structure of TIM DAQ process XML configuration is described in [2]. This document explains all specific configuration blocks to the SSHMH only.

2.2 The equipment address element

An SSHMH equipment address is a string, build of attribute-value pairs, separated by the semicolons. The sequence in which attributes appear in the string is not significant.

The list of the possible attributes is defined as follows:

- **default_server_alias** – specifies the default host for the handler's DTs/CTs execution. The default server alias's value should have the following format: `machine[:port]` Unless different port is specified, the 22 is taken by default.

- **default_user _name** – the default user name for SSH authorisation
- **default_user_password** – the default user's password

- **default_ssh_key** – the location of the file containing RSA private key used for the handler's SSH authentication. If specified, the handler uses the private/public keys authentication mechanism by default. It is possible to use the $HOME (or ${HOME}) environment variable inside the default_ssh_key. The handler validates its value locally, i.e. against the system the DAQ is started on.

- **default_key_passphrase** – if the private key requires a passphrase, it should be set at this point.

Some examples of the equipment addresses are presented below:

Ex.1)

```
default_server_alias=TIM_DAQTEST01:22;default_user_name=timtest;
default_user_password=password;
```

Ex2)

```
default_server_alias=TIM_DAQTEST01:22; default_ssh_key =
$HOME/.ssh/authorized_key.priv
```

Ex3)

```
default_server_alias=TIM_DAQTEST01:22; default_ssh_key =
${HOME}/.ssh/authorized_key.priv; default_user_name = timtest;
default_ssh_passphrase = passphrase
```

Ex4)

```
default_server_alias=TIM_DAQTEST01:22; default_ssh_key =
${HOME}/.ssh/authorized_key.priv;default_ssh_passphrase = passphrase
```

Comments to the above examples:

Presence of the default_ssh_key in an information for the handler, that the SSH public/private key-pair authentication mechanism should be used instead of the one involving the user-name and password. Thus, although placing attributes: default_ssh_key and default_user_password in one address it is not formally incorrect (and will not cause any errors) it is redundant, as the handler will never use the second one. The handler may, however use the default_user_name for the SSH private/public keys authentication, if it is not specified inside the CT/DT hardware address block.

In some cases, the fields of the SSHEMH equipment address can be overloaded by the corresponding attributes of the DT/CT hardware address block. This situation will happened for example, when one SSH DAQ process supervises DTs or CTs that need to execute their system-calls on different hosts.

2.3 DT & CT hardware address configuration blocks

All DTs and CTs must have hardware addresses of type: `SHHardwareAddress`.

SSHHardwareAddresses are instantiated at DAQ process start-up from XMLblocks that have the following format:

```
<HardwareAddress class=
    "ch.cern.tim.shared.datatag.address.impl.SSHHardwareAddressImpl">
    <!—not obligatory, if the default server-alias is specified inside the handler's
    equipment address -->
    <server-alias>host:22</server-alias>
    <user-name>user</user-name>
    <user-password>password</user-password>
    <ssh-key>${HOME}/.ssh/key_priv</ssh-key>
   <key-passphrase>passphrase</key_passphrase >
    <call-interval>60</call-interval>
    <call-delay>30</call-delay>
    <protocol>simple-io</simple-io>
    <system-call>daqprocess/bin/testControlProcess</system-call>
</HardwareAddress>
```

The XML SSHHardwareAdderess definition presented above includes all possible attributes the address can have. In practice, however, not all need to be specified and only two fields are mandatory.

The mandatory fields are:

- **system-call** – specifies the SSH command (a binary program or a shell script) to execute on a remote machine after the handler logs into it successfully.

For example, having the system-call set to: `daqprocess/bin/tesControlProcess`, the handler takes the following action: `ssh timtest@tcrpl4 daqprocess/bin/testControlProcess`

All required command line arguments for the command should be put into the system call too. For example, the CT responsible for starting up the TIM DAQ process, named: P_DIP01 should be defined in the following way:

```
<system-call>
        daqprocess/bin/daqprocess.sh start P_DIP01
</system-call>
```

The system-call defined as above, causes the handler to take the following action:

```
ssh {user}@{machine} daqprocess/bin/daqprocess.sh start DIP-001
```

One need to remember, that unless the SSH command put inside the system-call is a standard UNIX shell command, or is included in the $PATH on the remote machine, it should be declared with the absolute path.

- **protocol** – specifies the communication protocol between the handler and the HL. The protocol field must be set: `xml` or `simple-io`. Both protocols are described more precisely in chapter 3.

The remaining, optional fields of the SSH hw. address are:

- **server-alias** – like the default_server_alias from the equipment address, it specifies the remote machine on which the system-call should be executed. If specified, it overloads the default one.

- **user-name** – if specified it overloads its default equivalent from the generic equipment hardware address.

- user-password - if specified it overloads its default equivalent from the generic equipment hardware address.

- ssh-key- if specified it overloads its default one from the equipment hardware address.

- ssh-passhprase- if specified it overloads its default one from the equipment hardware address.

- user-password - if specified it overloads its default equivalent from the generic equipment hardware address

- call-interval – concerns only DTs. It specifies the time interval (in seconds) for periodic DT value recalculation. All DTs supervised by the SSHEMH are treated in the same way, i.e. their values are periodically reset by executing SSH system-calls, which should result with obtaining current values for the related tags.

- call-delay – concerns only DTs. It specifies the time (in seconds), how long the first execution of a DT's system-call should be postponed, after the handler's initialisation. After the thirst execution is done, the handler continues to periodically execute system-call, according to call-interval scheduling parameter.

2.4 SSHEMH authentication mechanisms

When logging on a remote machine, the SSHEMH can authenticate itself in two possible ways:

- **authenticating using private-public RSA key pair**
The handler tries the private-public key pair authentication method if the **ssh_key** is set inside the DT/CT hardware address, or the default one is found inside the handler's equipment address. In that case, the private-public key authentication method becomes the main authentication method. The server-alias is taken from the CT (or DT), or from the equipment address (if not found inside the DT/CT hw. address)

The handler tries to establish the user name in the following order:
1. it searches the DT/CT hardware address
2. it searches the equipment address
3. it looks up the $USER environment variable

The handler fixes the hostname in the similar way, but as the first step the $HOST variable is taken.

If neither the CT hardware address contain the ssh_key nor the default one can be found in the equipment hardware address, the handler tries to authenticate itself using the **user_name** & **user_password** pair.

- **authenticating using user-name & password**
The SSHMH tries to authenticate itself using the user name and password specified in the XML configuration obtained from the TIM business logic layer. To do so, the password needs to be specified either inside the hardware address configuration block of a DT (CT) or the default one inside the equipment

address. If available, the user_name and user_password of the tag's hardware address has higher priority, than the defaults from the equipment hardware address. If specified neither in the tag's hw. address block nor inside the eq. address, the handler set's the user_name by looking up $USER env. variable. Similarly, the server-alias is taken from the CT (or DT), or from the equipment address (if not found inside the DT/CT hw. address. Unless found in both places, the server_alias is initialised with $HOST env. variable.

The authentication algorithm can be described by the following meta-code:

```
String s_alias = null;
String user = null;
String password = null;
String key = null;
String key_pass = null;

if (server_alias is specified inside the DT/CT hw. address) {
   // take the server alias from the DT/CT's eq. address
   s_alias = server-alias;
}
else (default_server_alias is specified inside the handler's eq. address){
  // take the server alias from the handler's eq. address
  s_alias = default_server_alias;
}
else {
  s_alias = $HOST;
}

if (ssh_key is specified inside the DT/CT hw. address) {
   // take the key from the DT/CT's eq. address
   key = ssh_key;
}
else (default_ssh_key is specified inside the handler's eq. address) {
  // take the default key from the handler's eq. address
  key = default_ssh_key;
}

if (key_passphrase is specified inside the DT/CT hw. address) {
   key_pass = ssh_key;
}
else (default_key_passphrase is specified inside the handler's eq. address) {
  key_pass = default_key_passphrase;
}

if (user_name is specified inside the DT/CT hw. address) {
   // take the username alias from the DT/CT's eq. address
   user = user-name;
}
else (default_user_name is specified inside the handler's eq. address) {
  // take the username from the handler's eq. address
  user = default_user_name;
}
else {
  user = $USER;
}

if (user_password is specified inside the DT/CT hw. address) {
   // take the password from the DT/CT's eq. address
   password = user-password;
}
else {
  // take the server alias from the handler's eq. address
  password = default_user_password;
}

if  (key != null) {

   // private-public authentication

   if (key_pass != null) {
       authenticate_using_key(s_alias, user, ssh_key, ssh_pass);
   }
   else {
       authenticate_using_key(s_alias, user, ssh_key);
  }
}
else {
  if  (password != null)
     authenticate(s_alias, user, password);
  else
     ERROR;
}
```

2.5 Example SSH DAQ configuration

An example XML configuration of a DAQ process equipped with the SSMMH component is presented below :

```
<ProcessConfiguration process-id="4020" type="initialise">
  <jms-user> <!-- jms_user_name --> </jms-user>
  <jms-password> <!—jms_user_passwd --></jms-password>
  <jms-qcf-jndi-name>jms/process/factories/QCF</jms-qcf-jndi-name>
  <jms-queue-jndi-name>jms/process/destinations/queues/processmessage/P_SSHCTRL01
   </jms-queue-jndi-name>
  <jms-listener-topic>tim.process.tcrpl9.cern.ch.P_SSHCTRL01</jms-listener-topic>
  <alive-tag-id>100410</alive-tag-id>
  <alive-interval>60000</alive-interval>
  <max-message-size>100</max-message-size>
```

```
<max-message-delay>1000</max-message-delay>
<EquipmentUnits>
  <EquipmentUnit id="1" name="E_SSH_SSHCTRL01">
    <handler-class-name>ch.cern.tim.driver.ssh.SSHMessageHandler
    </handler-class-name>
    <commfault-tag-id>32047</commfault-tag-id>
    <commfault-tag-value>false</commfault-tag-value>
    <alive-tag-id>42841</alive-tag-id>
    <alive-interval>120000</alive-interval>
    <address>default_server_alias=TIM_TEST_DAQ01:22;
      default_user_name=timtest;default_ssh_key=
      ${HOME}/.ssh/key_priv; default_key_passphrase=passphrase
    </address>
    <DataTags>
     <DataTag id="1" name="TIMDAQ1:FREE_MEMORY_BLOCK" control="false">
        <data-type>Boolean</data-type>
        <DataTagAddress>
            <time-to-live>9999999</time-to-live>
            <priority>7</priority>
            <guaranteed-delivery>false</guaranteed-delivery>
            <HardwareAddress class=
            "ch.cern.tim.shared.datatag.address.impl.SSHHardwareAddressImpl">
                  <!— checks the current size of a free memomy block (in kB) -->
                  <system-call>
                        cat /proc/meminfo | grep MemFree | awk '{ print $2}'
                  </system-call>
                  <protocol>simple-io</protocol>
                   <!— reset the value every 30 sec. -->
                  <call-interval>30</call-interval>
                   <!— schedule the first value calculation 30 sec. after start-up -->
                  <call-delay>30</call-delay>
            </HardwareAddress>
       </DataTagAddress>
     </DataTag>
   </DataTags>
   <CommandTags>
     <CommandTag id="1" name="P_DIP01_start">
       <source-timeout>4000</source-timeout>
       <source-retries>0</source-retries>
       <HardwareAddress class=
            "ch.cern.tim.shared.datatag.address.impl.SSHHardwareAddressImpl"
            <system-call>
                 daqprocess.sh start P_DIP01
            </system-call>
            <protocol>xml</protocol>
        </HardwareAddress>
     </CommandTag>
     <CommandTag id="2" name="P_DIP01_stop">
       <source-timeout>4000</source-timeout>
       <source-retries>0</source-retries>
```

```
        <HardwareAddress class=
             "ch.cern.tim.shared.datatag.address.impl.SSHHardwareAddressImpl"
             <system-call>
                 daqprocess.sh stop P_DIP01
             </system-call>
             <protocol>xml</protocol>
         </HardwareAddress>
      </CommandTag>
    </CommandTags>
   </EquipmentUnit>
  </EquipmentUnits>
</ProcessConfiguration>
```

3. SSHMH – HL communication protocols: XML vs. simple-io

3.1 The basic protocol (simple IO)

The basic variant of SSHMH – HL communication will be applied mainly for DTs.
As SSHEMH's DTs are foreseen to be used mainly for gathering control information, about an overal state of the system, (such as CPU load, memory usage etc...) utilising standard UNIX shell programs. In that case the handler expects to receive feedbacks on standard output, but at the same time, it monitors the process termination codes. Thus, the system-call should always terminate with an exit code 0 – indicating success or -**1** indicating an error.

E.g, the following DT's system call:

```
<protocol>simple-io</protocol>
<system-call>
        cat /proc/meminfo | grep MemFree | awk '{ print $2}'
</system-call>
```

could be used for monitoring the size of a free memory block on a remote machine.
The handler automatically tries to match the type of the value it receives with the predefined type of the tag and (if required) does the necessary conversion, or invalidates the tag.
The tag is always invalidated in case an error code is caught on system-call exit.

3.2 The XML-based protocol

The XML-based protocol may find its application in more complicated cases. If the XML protocol is configured for a DT or CT, the handler expects to receive an XML feedback on standard IO, after executing it.

The format of the message is defined by the DTD, described below:

```
        <!DOCTYPE execution [
           <!ELEMENT execution-status (status-code, status-description)>
           <!ELEMENT status-code (#PCDATA)>
           <!ELEMENT status-description (#PCDATA)>
           <!ELEMENT value (#PCDATA)>
              <!ATTLIST value type (java.lang.Integer | java.lang.Boolean |
                     java.lang.Long | java.lang.Float | java.lang.Double |
                     java.lang.String)  #REQUIRED>
           ]>
```

The element **execution-code** is mandatory and should contain an integer value of value **0** or **-1**.
Value **0** indicates status: SUCCESS, while **-1** indicates: FAILURE.

The **status-description** element (not obligatory) carries additional information, such as e.g. explanation of the FAILURE execution status. If this concerns a DT, this information is used for tag invalidation, while with CTs this string is wrapped inside the command report.

In both cases it is transmitted up to the client layer, so that the client can understand what happens in a clear way.

The status description is ignored every time the invoked shell process exits with state: SUCCESS.

The **value** element is obligatory only if using xml protocol with DTs. It wraps a tag value obtained by executing the tag's system-call. The value element should always have the **type** attribute, to inform the handler about the data type of the value. The possible types are defined by the DTD presented above.

An example of the xml-based system-call is presented below:

```
<protocol>xml</protocol>
<system-call>
   daqprocess/bin/daqprocess.sh start DIP-001
</system-call>
```

3.3 Example XML feedback messages

Example 1: an XML feedback of the DAQ process start-up script, after unsuccessful attempt to start-up process P_DIP01

```
 <?xml version="1.0"?>
 <execution-status>
    <status-code>-1</status-code>
    <status-description>
          The process P_DIP01 seems to be running, stop it first
    </status-description>
  </execution-status>
```

Example 2: an XML feedback of the start-up script, after successfully starting up process

P_DIP01

```
<?xml version="1.0"?>
<execution-status>
    <status-code>0</status-code>
</execution-status>
```

Example 3: an XML feedback of the TIM JMS broker script watchdog script, used for checking states of the process queues:

```
<?xml version="1.0"?>
<execution-status>
     <status-code>-1</status-code>
     <status-description>The JMS queue tim.sys.processmessage.P_APIMMD06 is
        full. Contact the system administrator
     </status-description>
</execution-status>
```

Example 3: an XML feedback of the system-call associated to a DT keeping the current processor load (percentage) on a remote machine:

```
<?xml version="1.0"?>
<execution-status>
     <status-code>0</status-code>
      <value type="java.lang.Integer">67</value>
</execution-status>
```

4. Example use-cases for the SSH DAQs

The SSH DAQs can utilise a set of existing UNIX tools for controlling the overall system state.

4.1 TIM operational servers monitoring issues

4.1.1 Controling CPU issues with vmstat

vmstat - reports virtual memory statistics of process, virtual memory, disk, trap, and CPU activity.

On multicpu systems, vmstat averages the number of CPUs into the output. For per-process statistics .Without options, vmstat displays a one-line summary of the virtual memory activity since the system was booted.

Basic synctax is: ***vmstat <options> interval count***

option - let you specify the type of information needed such as paging -p , cache -c ,.interrupt -i etc. if no option is specified information about process , memory , paging , disk ,interrupts & cpu is displayed .

interval - is time period in seconds between two samples . vmstat 4 will give data at each 4 seconds interval.

count - is the number of times the data is needed . vmstat 4 5 will give data at 4 seconds interval 5 times.

Following columns has to be watched to determine if there is any cpu issue

1. Processes in the run queue (`procs r`)

   An example system-call for obtaining this parameter could look as follows:

   ```
   <system-call>vmstat 1 2 | tail -1 | awk '{print $1}' </system-call>
   ```

2. User time (`cpu us`)

   An example system-call for obtaining this parameter could look as follows:

   ```
   <system-call>vmstat 1 2 | tail -1 | awk '{print $13}' </system-call>
   ```

3. System time (`cpu sy`)

   An example system-call for obtaining this parameter could look as follows:

   ```
   <system-call>vmstat 1 2 | tail -1 | awk '{print $14}' </system-call>
   ```

4. Idle time (`cpu id`)

   An example system-call for obtaining this parameter could look as follows:

   ```
   <system-call>vmstat 1 2 | tail -1 | awk '{print $15}' </system-call>
   ```

```
     procs      cpu
     r b w    us sy  id
     0 0 0    4  14  82
     0 0 1    3  35  62
     0 0 1    3  33  64
     0 0 1    1  21  78
```

Problem symptoms:

1. If the number of processes in run queue (`procs r`) are consistently greater than the number of CPUs on the system it will slow down system as there are more processes then available CPUs.

2. if this number is more than four times the number of available CPUs in the system then system is facing shortage of cpu power and will greatly slow down the processes on the system.

3. If the idle time (`cpu id`) is consistently 0 and if the system time (`cpu sy`) is double the user time (`cpu us`) system is facing shortage of CPU resources.

Resolution:

Resolution to these kinds of issues involves tuning of application procedures to make efficient use of cpu and as a last resort increasing the cpu power or adding more cpu to the system.

Having defined the control tags corresponding to the information described above, TIM could also provide simple rules for composed boolean states to monitor the overall state of its operational machines.

4.1.2 Controling memory issues

…

5. References

[1] TIMDAQDesignDescription - TIM DAQ module rchitecture presentation and implementation detiles.

[2] EMHDevelopersGuide – introduction to TIM EMH components development