10/3/2010

WS-ARC service configuration manual

Martin Skou Andersen*

*skou@nbi.dk

Contents

1 Introduction

The Web Service (WS) Advanced Resource Connector (ARC) provides a great deal of services which all share a common configuration interface. One or more services can be specified in a configuration and the `arched` daemon should then be used for starting the service(s). The configuration should be specified in a file using either the relatively simple INI format, or the more complex XML format.

INI configuration is the recommended way of configuring services for people who are new to ARC or unfamiliar with WS-ARC configuration. However it is limited in the sense that it relies on profiles. A profile is a configuration template setup for one or more services, and common use case profiles exist for most services and are installed per default. The limitation lies in the fact that only services and options which are specified in the profile can be configured in INI configuration, however the installed profiles should give enough flexibility for common use cases. For a detailed description on INI configuration see section 2. Profiles are further discussed in section 4.

XML configuration is the core format of WS-ARC configuration – it is used internally – and thus it gives full flexibility when configuring services. A detailed description of XML configuration can be found in section 3.

The `arched` daemon, which is used to start WS-ARC services according to the configuration is briefly discussed in section 5.

1.1 How to read this manual

This manual is divided into the three independent sections INI configuration, XML configuration and Profiles, and these can be read independent of each other. Additionally the `arched` section describe the command with the same name.

It should be noted that words typeset with the `typewriter` font should be typed as written.

2 INI configuration

As mentioned in the introduction, INI configuration directly relies on profiles, and a profile is a configuration template setup for one or more services. So when creating a INI configuration file a choice of profile must be made. Only one profile can be used in a given INI configuration. A standard WS-ARC installation comes with a set of profiles and the list in appendix A gives a short description of these. The profiles will per default be installed into the `/usr/share/arc/profiles` folder, where it is also possible to find an example INI configuration file for each of the profiles.

A INI configuration file basically consist of sections, attribute-value pairs and maybe comments and blank lines. In the following code-listing the syntax of an INI configuration file is shown:

```
# A comment
[ common ]
profile = <profilename> or <path-to-profile>

<attribute> = <value>
...
...

# Another comment
[ <section-name> ]
<attribute> = <value>
...
...
[ <another-section> ]
<attribute> = <value>
...
...
```

A section is specified by putting the name of the section inside brackets ([and]), where the left bracket ([) must be the first non white space character on the line. Any white space character after the left bracket and before the right will not be part of the section name. Every attribute-value pair belongs to the section which has be been defined previously. If the file starts without declaring a section, then following attributes will belong to the common section. Multiple sections with identical names are different sections, except for the common section, which is special in this respect and if multiple common sections exist these will be considered as one.

The `profile` attribute is also special, as it is used to indicate which profile the INI configuration uses. Its value should either be the name of a WS-ARC profile (no `xml` suffix), see appendix A or it should be the absolute path to a profile (with `xml` suffix). The `profile` attribute must be located in the common section. If multiple `profile` attributes exist only the first attribute will be used.

Attribute-value pairs are defined by specifying the name of the attribute followed by an equal sign and then the value the attribute should take. It is the first equal sign which separates the attribute from the value. If a line, not defining a section, does not contain an equal then it will be ignored and if it contains multiple equal signs then all but first equal sign will be part of the value. Any white space before and after the attribute and value will be ignored. Only attributes which have been defined in the selected profile will be consideren by `arched`. For each profile in WS-ARC a corresponding INI configuration file exist which defines sensible values and at the time list all configurable attributes. When it comes to multiple attributes with identical names defined in the same section, it is the selected profile which determines if multiple attributes is allowed and will be interpreted or not. It is possible that attributes from one section can override attributes from another section, however this behaviour is completely determined by the profile.

Comments in the INI configuration file are specified by letting the first non white space character be the sharp character (#), and these will be ignored by `arched`.

2.1 Examples

2.1.1 Echo service

```
profile = /usr/share/arc/profiles/EchoService.xml

pidfile = /tmp/arched.pid
logfile = /tmp/arched.log
loglevel = INFO

port = 60000

cacert = /etc/grid-security/certificates
host_cert = /etc/grid-security/hostcert.pem
host_key = /etc/grid-security/hostkey.pem

[ echo ]
prefix = {{
suffix = }}
```

2.1.2 Other service

2.2 Limitations

Configuring WS-ARC services in INI format may pose some limitations, since this configuration format depend on profiles. If one wants to configure a service setup which is not covered by any of the available profiles, it is simply not possible to configure that service setup in INI format and one has to use the XML format described in section 3. It should however be noted that it is easy to switch from INI to XML format since the `arched` command provide a way to convert a INI configuration into XML. See section 5 for further details.

3 XML configuration

3.1 Overview

XML configuration in WS-ARC provides full flexibility for configuring services, and to construct an adequate configuration one need to address the following subjects. If not done already, a choice on which service(s) being setup need to be made and a short discussion of the available WS-ARC services is given in section 3.2. Upon the choice of service(s) the proper message chain need to be configured and a discussion about message chains is given in section 3.3. Most people also want some kind of security setup capable of allowing/denying certain users, which can be done with security handlers and is discussed in section 3.4. Common configuration options like specifying plugins required for instance by the message chain, and configuring logging, is discussed in section 3.6. In section 3.7 example XML configuration files are given.

3.2 Service configuration

The following services are part of WS-ARC and specific configuration details for these are given in the respective references below:

A-REX - configuration manual in "The ARC Computational Job Management Module - A-REX" [4],

Chelonia storage multi-service - "Chelonia Adminstrator's Manual" [8],

ISIS distributed P2P information service - "ARC Information System" [5],

Charon authorization service - currently described in "Security Framework of ARC NOX" [14],

HOPI lightweight HTTP webserver - "The HOPI Manual" [7] and also partially described in "Chelonia Administrator's Manual" [8].

3.3 Message Chain

To configure WS-ARC services, a basic knowledge about how these are communicating with clients or other services is needed. WS-ARC uses a chain of message components (MCCs) which serves different levels of functionality. In one end it accepts messages from an interface and in the other a service processes them and produce a response. Two types of MCCs exist, and they differ in that the first type should be able to listen on some interface for messages, while the other simply receives messages from another MCC. The MCC then process the message and pass it on to a single or multiple other MCCs or services. MCCs should be configured using the XML element `Component`, where the `name` attribute specifies the MCC to use. The following MCCs are part of WS-ARC:

`tcp.service` Listens on the TCP interface, see section 3.3.1,

`tls.service` Adds encryption layer to communication, see section 3.3.2,

`http.service` Processes HTTP messages, see section 3.3.4,

`gsi.service` Processes GSI messages as implemented in Globus toolkit [3], see section 3.3.3,

`soap.service` Processes SOAP messages, see section 3.3.5.

Additionally a special MCC, the Plexer, exists which is able to pass messages to different components simply based on the endpoint the messages was sent to, and it should be configured using the `Plexer` element, see section 3.3.6. In the end of the chain a service should be specified, using the `Service` element, where the `name` attribute indicates the given service.

A working configuration should then link a set of MCCs and services, starting with a MCC that receives messages from some interface. The MCCs and services should be uniquely identifiable, by setting the `id` attributes of these elements. A MCC links to another or several MCCs or services using the `next` element, where the `id` attribute specifies the `id` of that MCC or service. Some MCCs support multiple `next` elements

with textual label inside. Those are used by such MCCs to perform internal routing of processed message. But most MCCs do not do any routing and hence require single `next` without label.

The message chain should be contained in the root element `ArcConfig`, which is shown in the conceptual configuration example below:

```
...
<Component name="mcc1.service" id="mcc1">
  <next id="mcc2"/>
</Component>
<Component name="mcc2.service" id="mcc2">
  <next id="service"/>
</Component>
<Service name="service" id="service"/>
...
```

For readability the message chain can be grouped into a `Chain` element in one or more levels.

More technical information about available MCCs and the messages their process may be found in the Hosting Environment technical manual [2].

3.3.1 TCP MCC

The TCP message component is named `tcp.service` and should be the first component in a chain. It is used to setup a server socket which will listen for incoming connections. The socket can be configured using the `Listen` element, which can occur multiple times and under each element the following child elements can be specified:

`Interface` This element specifies the interface which socket should listen on. If this element is not specified then the socket will listen on all available interfaces.

`Port` The port which the socket should bind to should be specified with this element. If no port is specified an error will be reported and this `Listen` element will be ignored.

`Version` The IP version can be specified by this element, and by default both version 4 and 6 will be used. It is possible to specify version 4 or 6 only.

`NoDelay` By default Nagle's algorithm is used for sending data, however if the `NoDelay` element is set to `true` data will be sent when ready. This is mainly used for optimizing connections for communication patterns which involve a lot of very short messages exchanged by service and client.

`Timeout` Specifies how much time, in seconds, data sending or receiving may take. If time is exceeded MCC reports error and in most cases connection is dropped. The default value is 60 seconds.

It is also possible to put a limit on number of incoming connections, and how these should be handled if the limit is reached. The limit can only be put on all configured interfaces, and not per interface. The `Limit` element specifies the limit on incoming connections, and the `drop` attribute specifies whether new connections, above the limit, will be put on hold (default) or if they should be dropped, which is specified by setting the attribute to `true`.

An example of a TCP MCC configuration using the described elements is shown below:

```
...
<Component name="tcp.service" id="tcp">
  <Listen>
    <Interface>w.x.y.z</Interface>
    <Port>44444</Port>
    <Version>4</Version>
    <NoDelay>true</NoDelay>
    <Timeout>120</Timeout>
  </Listen>
  <Listen>
```

```
      <Interface>a:b:c:d:e:f:g:h</Interface>
      <Port>55555</Port>
      <Version>6</Version>
      <NoDelay>false</NoDelay>
      <Timeout>20</Timeout>
    </Listen>
    <Limit drop="true">30</Limit>
  </Component>
  ...
```

3.3.2 TLS MCC

A security layer can be configured with the TLS MCC, named `tls.service`, and it should be put on top of a TCP MCC. By default it is able to understand the SSLv2 (hadshake only), SSLv3 and TLSv1 protocols, however the `Handshake` element can be used to specify another behaviour. The supported values for the `Handshake` element are `TLS` (default) and `SSLv3`, where specifying the latter will configure a MCC which only understand the SSLv3 protocol. If other values than the mentioned ones are given, then `TLS` will be used.

For the TLS MCC to function, it should have access to a host key-certificate pair. The key should be the private one, and it should not be password protected. The path to the key-certificate pair can be specified by using the `KeyPath` and `CertificatePath` elements. If the `KeyPath` element is not specified the default path `/etc/grid-security/hostkey.pem` will be used, and likewise if the `CertificatePath` element is not specified the default path `/etc/grid-security/hostcert.pem` will be used. The supported key and certificate formats are PEM and DER.

When establishing a connection with a client, the client certificate are per default verified against the Certificate Authority (CA) certificates, known to the TLS MCC. The CA certificates used for verification can be specified with either the `CACertificatesDir` or the `CACertificatePath` element, or both elements can be used. The `CACertificatesDir` element should point to a directory containing one or more CA certificates (in PEM format). The files should each contain exactly one certificate, and should be named by the hash value of the CA subject name. CA certificates provided by NorduGrid uses this option. The `CACertificatePath` element should point to a file containing one or more CA certificates (in PEM format) identified by:

```
    -----BEGIN CERTIFICATE-----
    ... (CA certificate in base64 encoding) ...
    -----END CERTIFICATE-----
```

sequences. If both elements have been specified, CA certificates will first be looked up in the file pointed to by the `CACertificatePath` element and then in the directory pointed to by the `CACertificatesDir` element. If none of the two elements have been specified, then the `CACertificatesDir` element will be initialised to the `/etc/grid-security/certificates` location. Client certificate verification can also be skipped, which is done by setting the `ClientAuthn` element to `false`.

```
  ...
  <Component name="tls.service" id="tls">
    <KeyPath></KeyPath>
    <CertificatePath></CertificatePath>
    <CACertificatePath></CACertificatePath>
    <CACertificatesDir></CACertificatesDir>
    <ClientAuthn></ClientAuthn>
    <Handshake>SSLv3</Handshake>
  </Component>
  ...
```

If element `CACertificatesDir` have attribute `PolicyGlobus` set to "true" additional verification of client credentials is performed. Client credentials are restricted to be signed by specific Certification Authorities

as specified in additional configuration files. Those files are stored in the location specified by the element `CACertificatesDir` and have names made of the hash of the subject of CA certificate followed by `.signing_policy` suffix. Information about format of those files may be found at http://dev.globus.org/wiki/Signing_Polic

3.3.3 GSI MCC

At its purpose MCC is similar to TLS described in section 3.3.2. It provides security layer using Globus implementation of GSS-API and corresponding communication protocol.

Its configuration element `KeyPath` and `CertificatePath` are identical to those of TLS MCC. This MCC is implemented using libraries of Globus Toolkit and its functionality and requirements are better to be described by descrition of Globus gssapi library.

Use of this MCC is not recommended due to non-standard origin of underlying protocol.

3.3.4 HTTP MCC

The HTTP protocol is supported by the HTTP MCC named `http.service`, and it should either be put on top of the TCP or TLS MCC. A working configuration of this MCC is simple, since it should only use the `next` element. Differently from most other MCCs this one may have multiple `next` with content being name of HTTP method in uppercase. Depending on method client requested message may be routed to different components. Most services of WS-ARC support GET and PUT for data communication and POST for SOAP. Some also support HEAD.

An example of a HTTP MCC configuration is shown below.

```
...
<Component name="http.service">
  <next id="component1">GET</next>
  <next id="component2">POST</next>
  <next id="component3">PUT</next>
</Component>
...
```

3.3.5 SOAP MCC

SOAP is supported by the SOAP MCC, named `soap.service`. It parses and processes incoming messages as SOAP messages. This component should be put right after the HTTP MCC in the message chain. The component is needed by most of the services in WS-ARC. The only option which should be configured for the SOAP MCC is to which component or service the valid SOAP messages should be directed to, and it is done with the `next` element as for the other components.

3.3.6 Plexer

The Plexer is a special MCC which filters messages based on their URL endpoints. It supports multiple `next` elements with content being regular expressions (POSIX extended) used to filter incoming messages. The message is forwarded to the component or service specified by the `id` attribute of the corresponding `next` element which mathed the endpoint URL of the message. At the first match the message will be forwarded to the respective component or service and no more expressions will be tried. The `next` elements are tried in order they are specified in configuration. If the message did not match any of the regular expressions then it is discarded and an error is returned to the client. An example of a Plexer configuration is shown below.

```
...
<Plexer>
  <next>^/arex$</next>
  <next>^/[eE][cC][hH][oO]$</next>
</Plexer>
...
```

3.4 Security Handlers

Security Handlers are components which extract security related properties of the message, collect then and use for making authorization decisions.

The Security Handlers are assigned MCCs and are arranged in queues. Each queue is processing specific messages. Usually MCCs implement two queues named "incoming" and "outgoing" for the messages going through chain to service and returning back respectively.

The Security Handlers in each queue are applied sequentially and if any produces negative result processing stops are error is raised.

The Security Handlers are added to MCCs through `SecHandler` configuration element put inside `Component` or `Service` elements. The `name` attribute specifies which plugin to be used. And queue is specified by `event` attribute. Each configured instance can be optionally assigned identifier through `id` attribute. Tne content of `SecHandler` is specific for every kind of plugin. Example is below:

```
...
<Component name="mcc1.service" id="mcc1">
  <next id="mcc2"/>
  <SecHandler name="handler1" event="incoming" id="id1">
  </SecHandler>
</Component>
...
```

For more technical information about security-related attributes of the message and the Security Handlers please see [14].

3.4.1 ARC Authorization

This Security Handler applies set of Policy Decision Point (PDP) element to the message and then produces decision result using combining algorithm specified in `action` attribute of `SecHandler` element. For desciption of PDP elements see section 3.5. The plugin name of this Secirity hadler is `arc.authz` and it resides in `arcshc` loadable module.

Available combining algorithms are:

breakOnDeny - processing stops when PDP returns Deny decision. The SecHandler produces negative result. This is a default algorith.

breakOnAllow - processing stopr when PDP return Allow decision. The SecHandler produces positive result.

breakAlways - processing stops after first PDP. Produced result is used as result of SecHandler.

breakNever - all PDPs are applied. Result of last PDP is used as result of SecHandler.

Here is an example of of ARC Authorization Security Handler configuration.

```
...
  <SecHandler name="arc.authz" event="incoming" id="authz">
    <PDP name="arc.pdp">
      ...
    </PDP>
    <PDP name="xacml.pdp">
      ...
    </PDP>
  </SecHandler>
...
```

3.4.2 Identity Mapping

The Identity Mapping plugin applies set of associated PDPs to the message being processed and depending on result assigns local account to the message. Its plugin name is `identity.map` and it resides in `identitymap` loading module.

This Security Handler works in a similar way to ARC Authorization (see section 3.4.1). But instead of passing results of PDPs as final processing result it applies local account mapping algorithm associated with particular PDP.

Configuration is also similar with only difference being that inside each `PDP` configuration element there is additional element representing mapping algorithm used for producing local account name. Available are following mapping algorithms:

`LocalName` - maps to specified account name.

```
...
    <PDP name="arc.pdp">
      <LocalName>nobody</LocalName>
      ...
    </PDP>
...
```

`LocalList` - uses subject of X.509 certificate presented by client to perform mapping. Element content provides path to local file containing pairs of subjects and account names. One pair per line separated by blank space.

```
...
    <PDP name="arc.pdp">
      <LocalList>/etc/grid-security/grid-mapfile</LocalList>
      ...
    </PDP>
...
```

`LocalSimplePool` - manages dynamic mapping of X.509 client subject to a set of local accont names. Content of element point to directory containing file named `pool`. This file contains list of local account available for mapping one per line. This directory is also used fro storing currently active mappings.

```
...
    <PDP name="arc.pdp">
      <LocalSimplePool>/etc/grid-security/mappool</LocalSimplePool>
      ...
    </PDP>
...
```

3.4.3 Delegation Collector

This plugin processes chain of X.509 certificates and extracts embedded policies. Policies are attached to the message and may be later used by Delegation PDP described in section 3.5.7. The plugin name of this Security Handler is `delegation.collector` and it resides in `mcctls` loadable module. It has no configuration parameters.

3.4.4 UsernameToken

The UsernameToken Security Handler processes information in the SOAP Header stored according to "Web Services Security UsernameToken Profile 1.0" [10]. The plugin name of this Security Handler is `usernametoken.handler` and it resides in `arcshc` loadable module.

The configuration element `Process` defines type of processing to be performed. Possible values are:

extract - parse and process elements in SOAP Header. This option shoudl be used for service side configuration.

generate - create UsernameToken. This option is only used for client side.

Only another configuration element relevant for service side processing is `PasswordSource`. It specifies path to local file containing list of usernames and passwords (one pair per line) which are accepted by plugin. If there is no pair in file matching one SOAP Header this Security Handler will raise an error.

3.4.5 X.509 Token

The X.509 Token Security Handler processes information in the SOAP Header stored according to "Web Services Security: X.509 Token Profile 1.0" [11]. The plugin name of this Security Handler is `x509token.handler` and it resides in `arcshc` loadable module.

The configuration element `Process` defines type of processing to be performed. Possible values are:

extract - parse and process elements in SOAP Header. This option shoudl be used for service side configuration.

generate - create X.509 Token. This option is only used for client side.

In `extract` mode this plugin process X.509 Token from SOAP Header and validates contained credetials against locally stored CA certificates. Following configuration elements specify location of trusted CA certificates:

CACertificatePath - path to file containing CA certificate.

CACertificatesDir - path to directory containing files with CA certificates.

If provided credentials failed verification this Security Handler will raise an error.

3.4.6 SAML Token

The SAML Token Security Handler processes information in the SOAP Header stored according to "Web Services Security: SAML Token Profile 1.1" [9]. The plugin name of this Security Handler is `samltoken.handler` and it resides in `arcshc` loadable module.

The configuration of this plugin is identical to previously described X.509 Token Security Handler. It supports configuration elements `Process`, `CACertificatePath` and `CACertificatesDir`.

The information collected by this plugin may then be used in ARC Policy Decision Point and Service Invoker Policy Decision Point plugins.

3.4.7 SAML Single Sign-On

3.5 Policy Decision Points

The Policy Decision Point component is responsible for applying particular policy or any kind of authorization evaluation algorithm to properties of the message and providing authorization decision which then used by other components - usually by the Security Handlers implementing functionality of Policy Enforcement Point.

3.5.1 Allow

This PDP always returns positive result. Its plugin name is `allow.pdp` and it resides in `arcshc` loadable module.

3.5.2 Deny

This PDP always returns negative result. Its plugin name is `deny.pdp` and it resides in `arcshc` loadable module.

3.5.3 Simple List

This PDP matches X.509 subject of client certificate to list of predefined values. Its plugin name is `simplelist.pdp` and it resides in `arcshc` loadable module.

Multiple configuration elements `DN` specify accepted subjects. Also attribute `location` may be used to point to a file containing list of subjects, one per line. If client's subject is not listed neither in `DN` nor in `location` PDP returns negative result. Otherwise result is positive. Below is configuration example for simple authorization by grid-mapfile:

```
...
    <PDP name="simplelist.pdp" location="/etc/grid-security/grid-mapfile">
      <DN>/O=Grid/O=NorduGrid/CN=Service Admin</DN>
    </PDP>
...
```

3.5.4 ARC

This plugin applies authorization policy written in ARC Policy language to attributes of the message. Its plugin name is `arc.pdp` and it resides in `arcshc` loadable module. For more information about ARC Policy language please see "Security Framework of ARC NOX" [13]. That docment also describes security attributes produced by different components of ARC which can be included in policies.

The configuration element `PolicyStore` specifies locations of policy. Its `Location` subelement provides path to file storing policy. Also policy can be directly embedded into configuration by using `Policy` configuration element.

The `PolicyCombiningAlg` element specifies how results from multiple policies are combined. For detailed explanation please refer to [13].

It is also possible to limit number of message attributes which are matched to policy using `Filter` configuration element. This may be needed to achieve better evaluation performance. The multiple subelements `Select` and `Reject` allow to either use only specific attributes or remove specific attributes. If neither `Select` not `Reject` are specified then all attribites are used.

Example of simple ARC PDP reading policy from external file is below:

```
...
    <PDP name="arc.pdp">
      <PolicyStore>
        <Location>/etc/grid-security/policy.xml</Location>
      </PolicyStore>
    </PDP>
...
```

3.5.5 GACL

This PDP shares configuration parameters and provides functionality similar to ARC PDP described in the section 3.5.4. Its plugin name is `gacl.pdp` and it resides in the same `arcshc` loadable module.

This plugin accepts policies written in GACL instead of ARC language. For more information about GACL prease check "The Gridsite Web" [6] and "GACL Mini-Howto" [1].

3.5.6 XACML

This plugin's configuration and functionality are similar to previously described ARC and GACL PDPs. Its plugin name is `xacml.pdp` and it resides in the same `arcshc` loadable module.

This plugin accepts policies written in subset of XACML language [12]. Currently XACML functionality is not complete yet. For more information see [13].

3.5.7 Delegation

This plugin applies authorization policy written in ARC Policy language and obtained from cahin of X.509 Proxy Certificates of client to attributes of the message. Its plugin name is `delegation.pdp` and it resides in `arcshc` loadable module.

Functionality wise it is very similar to ARC PDP described in section 3.5.4 but uses policy internally obtained from message. Hence it has only `Filter` configuration element.

For detailed explanation how policy document is embedded into X.509 Proxy Certificate please refer to [13].

3.5.8 Service Invoker

This plugin comunicates with possibly remote authorization service by passing it authorization request and getting back authorization decision. Its plugin name is `pdpservice.invoker` and it resides in `arcshc` loadable module.

The configuration element `Filter` is similar to those in other PDPs.

To specify service to be contacted `ServiceEndpoint` element is used. It contains contact URL of service. Service is expected to accept SOAP over HTTP(S) messages conforming to protocol defined in `TransferProtocol` configuration element. Possible options are `arc` for ARC own protocol and `saml` for "SAML 2.0 Profile of the XACML 2.0" kind of exchange. The format of payload - authorization request and response - is controlled using `RequestFormat` element. Implemented are `xacml` and `arc` options for XACML conforming or ARC specific formats respectively.

For establishing TLS communication channel to the service PDP needs to apply X.509 credentials. For that configuration elements `CertificatePath` and `KeyPath` are used. Alternatively if PDP uses X.509 Proxy Certificate path to file containing proxy may be specified in `ProxyPath`.

For verifying credentials presented by service trusted CA certificates are specified through `CACertificatePath` and `CACertificatesDir` for single file and directory containing multiple files respectively.

3.6 General Daemon Configuration

The following options configures the `arched` daemon, and thus applies to all services. Some of the options can also be set using command line arguments, and if used these will override the configuration file options. See section 5 for a description of the `arched` daemon.

Under the `Server` element the file to store the PID in, can be specified using the `PidFile` element, if not specified the PID will be stored in the file `/var/run/arched.pid`. This options can also be set using a command line argument. If the `Foreground` element is present under the `Server` element, `arched` will run in foreground and thus the `PidFile` element will be ignored. It is also posible to specify that `arched` should run in foreground on the command line, however it is not posible to specify that it should run as a daemon, if configured to run in foreground in the configuration file. Also under the `Server` the user and group owning the `arched` process can be specified, which is accomplished with the `User` and `Group` elements. These can also be set on the command line.

Logging options should be specified under the `Logger` element under the `Server` element. The `File` element specifies which file to write log messages to and if not present log messages will be written to the `/var/log/arched.log` file. Each log message is associated with a log level which indicates the serverity of the message. The following log levels exist in WS-ARC, ordered by serverity: `FATAL`, `ERROR`, `WARNING`, `INFO`, `VERBOSE` and `DEBUG`. With the `Level` element the level of messages being reported can be specified. Log messages with a higher or same level as specified by the `Level` element will be written to the log file. The

default log level is `WARNING`.
By default the log file is not limited in size, neither will any log rotation be done. The `Maxsize` and `Backups` elements can be used to change this behaviour. The `Maxsize` element sets the maximum file size of the log file in bytes, while the `Backups` element specifies the number of files used for log rotation. When the size of the log file exceeds the limit specified by the `Maxsize` element the log file is renamed to `<log-file>.1`. If this file exist it is renamed to `<log-file>.2` and so forth up to the number specified by `Backups`, which means that the last file will be deleted. The log file size is only allowed to exceed the specified limit by the size of one log message. If the `Maxsize` element have been specified and no `Backups` element is specified then log rotation will not be carried out, thus the log file is truncated when the limit is exceeded.

Plugins which should be used later in the configuration should be specified under the `Plugins` element where the `Name` element specifies the name of the plugin. The `Path` element under the `ModuleManager` element specifies where to locate the plugins.

The following example show the use of the above configuration options:

```
<?xml version="1.0"?>
<ArcConfig>
  <Server>
    <PidFile>/tmp/arched.pid</PidFile>
    <User></User>
    <Group></Group>
    <Logger>
      <File>/tmp/arched.log</File>
      <Level>WARNING</Level>
      <MaxSize>1000000</MaxSize>
      <Backups>5</Backups>
    </Logger>
  </Server>
  <ModuleManager>
    <Path>/usr/lib/arc</Path>
  </ModuleManager>
  <Plugins>
    <Name>plugin1</Name>
    <Name>plugin2</Name>
    ...
  </Plugins>
  ...
</ArcConfig>
```

3.7 Examples

3.7.1 Echo Service

```
<?xml version="1.0"?>
<cfg:ArcConfig xmlns="http://www.nordugrid.org/schemas/loader/2009/08"
               xmlns:cfg="http://www.nordugrid.org/schemas/arcconfig/2009/08"
               xmlns:tcp="http://www.nordugrid.org/schemas/tcp/2009/08"
               xmlns:tls="http://www.nordugrid.org/schemas/tls/2009/08"
               xmlns:echo="http://www.nordugrid.org/schemas/echo/2009/08"
               xmlns:infosys="http://www.nordugrid.org/schemas/infosys/2009/08"
                  >
  <cfg:Server>
    <cfg:PidFile>/tmp/arched.pid</cfg:PidFile>
    <cfg:Logger>
      <cfg:File>/var/log/arched.log</cfg:File>
      <cfg:Level>WARNING</cfg:Level>
    </cfg:Logger>
  </cfg:Server>
  <ModuleManager>
    <Path>/usr/lib/arc</Path>
```

```
    </ModuleManager>
    <Plugins>
      <Name>mcctls</Name>
      <Name>mcchttp</Name>
      <Name>mccsoap</Name>
      <Name>mcctcp</Name>
    </Plugins>
    <Chain>
      <Component name="tcp.service" id="tcp">
        <next id="http"/>
        <tcp:Listen>
          <tcp:Interface>0.0.0.0</tcp:Interface>
          <tcp:Version>4</tcp:Version>
        </tcp:Listen>
      </Component>
      <Component name="http.service" id="http">
        <next id="soap">POST</next>
        <next id="plexer">GET</next>
      </Component>
      <Component name="soap.service" id="soap">
        <next id="plexer"/>
      </Component>
      <Plexer id="plexer">
        <next id="echo">/Echo</next>
      </Plexer>
      <Service name="echo" id="echo">
        <echo:prefix>[</echo:prefix>
        <echo:suffix>]</echo:suffix>
      </Service>
    </Chain>
  </cfg:ArcConfig>
```

4 Profiles

In WS-ARC the concept of profiles have been introduced. A profile is a complete XML configuration template, for a specific service setup and it is used when configuring services in INI format. The core of WS-ARC uses an XML struture for service initialization, which makes the XML format the apparent choice for configuration. However the XML language is not simple and easy to use, which is why WS-ARC utilizes profiles, namely to provide a configuration format (the INI format) easy to use.

To write a profile, some knowledge on the INI (see section 2) and XML configuration formats (see section 3) is needed. Not necessaray to say, knowledge on the desired service setup is also needed. A XML configuration file is only a profile if it defines a option mapping from INI format to XML, thus making it possible to utilise the XML configuration through the INI format.

Two XML attributes have been introduced to be able to create a mapping, and these are `inisections` and `initag`. XML elements which should be configurable in INI format must specify these attributes. The `initag` attribute defines the name of the tag which represent the respective XML element in the INI configuration file, and the `inisections` attribute is used to specify, as a space separated list, which sections the tag can occur in. The list is ordered, in the sense that the listed sections is searched, in the specified order, for the tag specified by the `initag` attribute, and if found in a section no further sections will be searched. Only leaf XML elements (elements with no child elements) should be mapped, mapping non leaf element result in a undefined behaviour. A profile example of a XML element mapping is shown in the listing below:

ElementMapping.xml

```
...
<Server>
  ...
  <Logger>
```

```
    <File inisections="special common"
          initag="logfile">/var/log/arched.log</File>
    ...
  </Logger>
</Server>
...
```

And this example profile can then be utilised in a INI configuration file as:

ElementMapping.ini

```
profile=ElementMapping.xml
logfile=/etc/arc/arched.log
...
[ special ]
logfile=/var/arc/arched.log
...
```

And the resulting XML configuration will look like:

```
...
<Server>
  ...
  <Logger>
    <File>/var/arc/arched.log</File>
    ...
  </Logger>
</Server>
...
```

To be able to map XML attributes the use of a new XML element `AttributeRepresentation` have been introduced. The element should have three attributes set, the two attributes `inisections` and `initag` used as described above, and a new attribute `id` which specifies which attribute of the parent element is being mapped. A profile example of a XML attribute is shown in the listing below:

AttributeMapping.xml

```
...
<Component name="tcp.service">
  ...
  <Limit drop="false">
    <AttributeRepresentation id="drop" inisections="common" initag="drop"/>
    100
  </Limit>
</Component>
...
```

And this example profile can then be utilised in a INI configuration file as:

AttributeMapping.ini

```
profile=AttributeMappingExample.xml
drop=true
...
```

And the resulting XML configuration will look like:

```
...
<Component name="tcp.service">
  ...
  <Limit drop="true">100</Limit>
```

```
</Component>
...
```

5 The arched daemon

The `arched` daemon is the daemon which is used to start every service in WS-ARC. It will load a configuration file, and start the services specified herein.

arched [OPTION...]

(ARC 0.9)

`-f, --foreground`		run daemon in foreground
`-c, --xml-config`	*path*	full path of XML configuration file
`-i, --ini-config`	*path*	fill path of INI configuration file
`-d, --config-dump`		dump generated XML configuration
`-p, --pid-file`	*path*	full path of PID file
`-u, --user`	*user*	user name
`-g, --group`	*group*	group name
`-s, --schema`	*path*	full path of XML schema file
`-h, --help`		Show help options

If no configuration file is specified, `arched` first looks for the `/etc/arc/server.ini` INI configuration file if it does not exist then the `/etc/arc/server.xml` XML configuration file is tried. If none of these is found `arched` will abort with an error and exit code 1.

A INI configuration file can be specified with the `-i` option, while the `-c` option makes it possible to specify a XML configuration file. If both options are given, the INI configuration will be chosen.

`arched` also has the capability of printing the XML configuration which will be generated from a given INI configuration. By specifying the `-d` option along with the `-i` option and the path to the INI configuration `arched` will transform the INI configuration to XML and dump the output to standard out and exit.

A List of standard profiles in WS-ARC

A.1 A-REX

ComputingElementWithFork MINIMAL configuration template for a secure standalone Computing Element (CE) with a fork queue.

InsecureComputingElement

ComputingElementWithMapfile

ComputingElementWithMapfileAndPBSBackend

ComputingElement_DNlist

ComputingElement_VOMS

A.2 Chelonia

CheloniaAllCentralizedAHash

CheloniaAllCentralizedAHashWithISIS

CheloniaAllReplicatedAHash

CheloniaAllReplicatedAHashGatewayVOMSWithISIS

CheloniaAllReplicatedAHashWithISIS

CheloniaReplicatedAHash

CheloniaShepherdWithHopi

InsecureNonDistributedStorageElement

NonDistributedStorageElement_DNlist

NonDistributedStorageElement_VOMS

A.3 ISIS

InsecureP2PIIS

InsecureStandaloneIIS

P2PIIS

StandaloneIIS

A.4 Charon

Charon

A.5 Hopi

Hopi

HopiInsecure

HopiWithPlexer

HopiWithPlexerInsecure

A.6 Combined services

InsecureComputingAndStorageElement

ComputingAndStorageElement_DNlist

ComputingAndStorageElement_VOMS

A.7 Testing

Echo

EchoInsecure

EchoPython

EchoPythonInsecure

EchoSAMLToken

EchoUsernameToken

EchoX509Token

InsecureHopiService

B Profile attributes naming convention

Since profiles defines a mapping of XML elements or attributes to INI tags, they control the naming of the tags. So for consistency tags in different profiles reprensenting the same XML element or attribute should not be named differently. A naming convention list for the XML elements and attributes described in section 3 is presented below. Note that there are only conventions for leaf XML elements (element with no children).

XML element/attribute	INI tag
/Server/PidFile	pidfile
/Server/User	user
/Server/Group	group
/Server/Logger/File	logfile
/Server/Logger/Level	loglevel
/Server/Logger/Maxsize	logmaxsize
/Server/Logger/Backups	logbackups
TCP MCC	
/Component/Listen/Interface	interface
/Component/Listen/Port	port
/Component/Listen/Version	ipversion
/Component/Listen/NoDelay	nodelay
/Component/Listen/Timeout	timeout
/Component/Limit	limit
/Component/Limit.drop	drop
TLS MCC	
/Component/KeyPath	x509_user_key
/Component/CertificatePath	x509_user_cert
/Component/CACertificatePath	x509_cacert_path
/Component/CACertificatesDir	x509_cacert_dir
/Component/CACertificatesDir.PolicyGlobus	policyglobus
/Component/ClientAuthn	clientauthn
/Component/VOMSCertTrustDNChain/VOMSCertTrustDN	
/Component/VOMSCertTrustDNChain/VOMSCertTrustRegex	
/Component/VOMSCertTrustDNChainsLocation	
/Component/Handshake	handshake

C XML Schemas

C.1 General Daemon Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.nordugrid.org/schemas/arcconfig/2009/08"
    xmlns="http://www.nordugrid.org/schemas/arcconfig/2009/08"
    xmlns:loader="http://www.nordugrid.org/schemas/loader/2009/08"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">

<xsd:import namespace="http://www.nordugrid.org/schemas/loader/2009/08" schemaLocation="
    loader.xsd"/>
<!-- Root element -->
<xsd:element name="ArcConfig">
    <xsd:complexType>
        <xsd:annotation>
            <xsd:documentation>
            This is the top level element of ARC configuration document.
            </xsd:documentation>
```

```
            </xsd:annotation>
            <xsd:sequence>
                <xsd:element ref="Server" minOccurs="0" maxOccurs="1"/>
                <xsd:element ref="loader:ModuleManager" minOccurs="0" maxOccurs="1"/>
                <xsd:element ref="loader:Plugins" minOccurs="0" maxOccurs="1"/>
                <xsd:element ref="loader:Component" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="loader:Chain" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="loader:Plexer" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="loader:Service" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="loader:SecHandler" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <!-- Server -->
    <xsd:element name="Server">
        <xsd:complexType>
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                Server specific configuration options.
                </xsd:documentation>
            </xsd:annotation>
            <xsd:all>
                <xsd:element name="PidFile" type="xsd:string" minOccurs="0" maxOccurs="1"
                    default="/var/run/arched.pid">
                    <xsd:annotation>
                        <xsd:documentation xml:lang="en">
                        Path to pid file.
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="Foreground" type="xsd:boolean" minOccurs="0" maxOccurs="1
                    " default="false">
                    <xsd:annotation>
                        <xsd:documentation xml:lang="en">
                        Indicates whether the server runs in foreground or daemon mode. If
                            it runs in foreground the log goes to standard error as well.
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="User" type="xsd:string" minOccurs="0" maxOccurs="1">
                    <xsd:annotation>
                        <xsd:documentation xml:lang="en">
                        arched will run under the specified user. If not specified the
                            effective user will not be changed.
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="Group" type="xsd:string" minOccurs="0" maxOccurs="1">
                    <xsd:annotation>
                        <xsd:documentation xml:lang="en">
                        arched will run under the specified group. If not specified the
                            effective group will not be changed.
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element ref="Logger" minOccurs="0" maxOccurs="1"/>
            </xsd:all>
        </xsd:complexType>
    </xsd:element>

    <!-- Logger -->
    <xsd:simpleType name="LoggerLevel_Type">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
            The enumeration lists the supported log levels.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="FATAL"/>
            <xsd:enumeration value="ERROR"/>
            <xsd:enumeration value="WARNING"/>
            <xsd:enumeration value="INFO"/>
```

```
            <xsd:enumeration value="VERBOSE"/>
            <xsd:enumeration value="DEBUG"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:element name="Logger">
        <xsd:complexType>
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                Logger configuration element.
                </xsd:documentation>
            </xsd:annotation>
            <xsd:all>
                <xsd:element name="File" type="xsd:string" minOccurs="0" maxOccurs="1"
                    default="/var/log/arched.log">
                    <xsd:annotation>
                        <xsd:documentation xml:lang="en">
                        This element configures the file which should be used for storing
                            log messages.
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="Level" type="LoggerLevel_Type" minOccurs="0" maxOccurs="1
                    " default="WARNING">
                    <xsd:annotation>
                        <xsd:documentation xml:lang="en">
                        Specifies the log level of the messages which should be logged.
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="Maxsize" type="xsd:nonNegativeInteger" minOccurs="0"
                    maxOccurs="1">
                    <xsd:annotation>
                        <xsd:documentation xml:lang="en">
                        Defines the maximal size of the logfile in bytes.
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
                <xsd:element name="Backups" type="xsd:nonNegativeInteger" minOccurs="0"
                    maxOccurs="1">
                    <xsd:annotation>
                        <xsd:documentation xml:lang="en">
                        Defines the maximal number of logfiles used in log rotation.
                        </xsd:documentation>
                    </xsd:annotation>
                </xsd:element>
            </xsd:all>
        </xsd:complexType>
    </xsd:element>
    </xsd:schema>
```

C.2 Loader Schema

```
    <?xml version="1.0" encoding="UTF-8"?>
    <xsd:schema
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.nordugrid.org/schemas/loader/2009/08"
        xmlns="http://www.nordugrid.org/schemas/loader/2009/08"
        xmlns:ini="http://www.nordugrid.org/schemas/ini/2009/08"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified">

    <!-- ModuleManager -->
    <xsd:element name="ModuleManager">
        <xsd:complexType>
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                This element specifies parameters needed to successfully load plugins.
                    Currently it allows to specify filesystem paths to directories where
                    plugin libraries are located.
```

```
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="Path" minOccurs="0" maxOccurs="unbounded" default="?">
                <xsd:annotation>
                    <xsd:documentation xml:lang="en">
                    Specify filesystem paths to directories where plugin libraries are
                        located.
                    </xsd:documentation>
                    <xsd:appinfo>
                        <ini:tag>modulepath</ini:tag>
                    </xsd:appinfo>
                </xsd:annotation>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<!-- Plugins -->
<xsd:element name="Plugins">
    <xsd:complexType>
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
             This element defines shared library which contains plugins to be used. It
                 is supposed to be used if name of library is not same as name of plugin
                  and hence can't be located automatically.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="Name" minOccurs="0" maxOccurs="unbounded">
                <xsd:annotation>
                    <xsd:documentation xml:lang="en">
                    Specify the name of the plugin.
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<!-- Chain -->
<xsd:element name="Chain">
    <xsd:complexType>
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
            This element is not required and does not affect chains directly. It's
                purpose is to group multiple components logically mostly for readability
                 purpose.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element ref="Chain" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="Component" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="Plexer" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="Service" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="SecHandler" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<!-- Component -->
<xsd:element name="next">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:attribute name="id" type="xsd:string" use="optional"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>

<xsd:element name="Component">
    <xsd:complexType>
```

```
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                This element defines MCC plugin. Required attribute 'name' specifies name of
                     plugin as defined in MCC description. Required attribute 'id' assigns
                    identifier which is used to refer to this element from others. Sub-
                    elements 'next' refer to next components in a chain through their
                    attribute 'id' and their content represent assigned component-specific
                    label. If attribute 'id' is missing all 'next' refer to next component
                    in document. If 'next' is missing one label-less 'next' is assigned
                    automatically. Presence of attribute 'entry' exposes this MCC through
                    Loader class interface to external code. That is meant to be used in
                    code which cretes chains dynamically like client utilities. Rest
                    elements define component-specific configuration.
                </xsd:documentation>
            </xsd:annotation>
            <xsd:sequence>
                <xsd:element ref="next" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="SecHandler" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:any namespace="##other" processContents="strict" minOccurs="0"
                    maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:string" use="required"/>
            <xsd:attribute name="id" type="xsd:string" use="required"/>
            <xsd:attribute name="entry" type="xsd:string" use="optional"/>
            <xsd:anyAttribute namespace="##other" processContents="strict"/>
        </xsd:complexType>
    </xsd:element>

    <!-- Plexer -->
    <xsd:element name="Plexer">
        <xsd:complexType>
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                 This element is a Plexer. Optional attribute 'id' assigns identifier which
                     is used to refer to this element from others. If not specified it will
                     be assigned to "plexer" automatically. Sub-elements 'next' refer to
                     next components in a chain and their content represent requested
                     endpoints. In Plexer content of element 'next' represents Regular
                     Expression pattern. For every incoming message path part of message's
                     endpoint is matched pattern.In case of ordinary service element 'next'
                     may look like
                   <next>^/service$</next>
                If service is also responsible for whole subtree then simple solution is
                   <next>^/service</next>
                But more safer solution would be to use 2 elements
                   <next>^/service$</next>
                   <next>^/service/</next>
                Unmatched part of message endpoint is propagated with incoming message in
                attribute PLEXER:EXTENSION and may be used by service to determine response.
                </xsd:documentation>
            </xsd:annotation>
            <xsd:sequence>
                <xsd:element ref="next" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:any namespace="##other" processContents="strict" minOccurs="0"
                    maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="id" type="xsd:string" use="optional"/>
            <xsd:anyAttribute namespace="##other" processContents="strict"/>
        </xsd:complexType>
    </xsd:element>

    <!-- Service -->
    <xsd:element name="Service">
        <xsd:complexType>
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                 This element represents a service - last compnent in a chain. Required
                     attribute 'name' specifies name of plugin as defined in Service
                     description. Required attribute 'id' assigns identifier which is used
                     to refer to this element from others. Rest elements define service-
                     specific configuration.
                </xsd:documentation>
            </xsd:annotation>
```

```
            <xsd:sequence>
                <xsd:element ref="SecHandler" minOccurs="0" maxOccurs="unbounded"/>
                <xsd:any namespace="##other" processContents="strict" minOccurs="0"
                    maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:string" use="required"/>
            <xsd:attribute name="id" type="xsd:string" use="required"/>
            <xsd:anyAttribute namespace="##other" processContents="strict"/>
        </xsd:complexType>
    </xsd:element>

    <!-- SecHandler -->
    <xsd:element name="SecHandler">
        <xsd:complexType>
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                 This element specifies security handler plugin to be called at various
                    stages of message processing. Depending on produced result message may
                    be either sent farther through chain or processing would be cancelled.
                    Required attribute 'name' specifies name of plugin. Attibute 'id'
                    creates identifier of SecHandler which may be used to refer to it. If
                    attribute 'refid' is defined then configuration of SecHendler is
                    provided by another element within ArcConfig with corresponding 'id'.
                    Attribute 'event' defines to which queue inside MCC SecHandler to be
                    attached. If it's missing SecHandler is attached to default queue if
                    MCC has such. Names of queues are MCC specific. If not otherwise
                    specified they are 'incoming' and 'outgoing' and are processed for
                    incoming and outgoing messages. There is no default queue by default.
                </xsd:documentation>
            </xsd:annotation>
            <xsd:sequence>
                <xsd:any namespace="##other" processContents="strict" minOccurs="0"
                    maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:string" use="required"/>
            <xsd:attribute name="id" type="xsd:string" use="optional"/>
            <xsd:attribute name="refid" type="xsd:string" use="optional"/>
            <xsd:attribute name="event" type="xsd:string" use="optional"/>
            <xsd:anyAttribute namespace="##other" processContents="strict"/>
        </xsd:complexType>
    </xsd:element>

    </xsd:schema>
```

C.3 TCP MCC Schema

```
    <?xml version="1.0" encoding="UTF-8"?>
    <xsd:schema
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.nordugrid.org/schemas/tcp/2009/08"
        xmlns="http://www.nordugrid.org/schemas/tcp/2009/08"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified">

    <!-- Listen -->
    <xsd:element name="Listen">
        <xsd:complexType>
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                This element defines listening TCP socket. If interface is missing socket is
                     bound to all local interfaces (not supported). There may be multiple
                    Listen elements.
                </xsd:documentation>
            </xsd:annotation>
            <xsd:sequence>
                <xsd:element name="Interface" type="xsd:string" minOccurs="0" maxOccurs="1"
                    default="0.0.0.0">
                    <xsd:annotation>
                        <xsd:documentation xml:lang="en">
                        Listen network interface.
```

```
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="Port" type="xsd:int">
                <xsd:annotation>
                    <xsd:documentation xml:lang="en">
                    Listen network port.
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="Version" minOccurs="0" maxOccurs="1">
                <xsd:simpleType>
                    <xsd:annotation>
                        <xsd:documentation xml:lang="en">
                        This element defines TCP/IP protocol version.
                        If not specified both versions will be used whenether possible.
                        </xsd:documentation>
                    </xsd:annotation>
                    <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="4"/>
                        <xsd:enumeration value="6"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:element>
            <xsd:element name="NoDelay" type="xsd:boolean" minOccurs="0" maxOccurs="1"
                default="false">
                <xsd:annotation>
                    <xsd:documentation xml:lang="en">
                    Apply no delay socket option. If set to true TCP packets will
                    be proceesd as soon as possible without waiting for more data
                    to become available.
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="Timeout" type="xsd:int" minOccurs="0" maxOccurs="1"
                default="60">
                <xsd:annotation>
                    <xsd:documentation xml:lang="en">
                    Timeout for socket level operations
                    </xsd:documentation>
                </xsd:annotation>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<!-- Connect -->
<xsd:element name="Connect">
    <xsd:complexType>
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
            This element defines TCP connection to be established to specified Host at
                specified Port. If LocalPort is defined TCP socket will be bound to this
                 port number (not supported).
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="Host" type="xsd:string"/>
            <xsd:element name="Port" type="xsd:int"/>
            <xsd:element name="Timeout" type="xsd:int"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="Limit">
    <xsd:complexType>
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
            This element defines upper limit for number of simultaneous
            active TCP connections. Only positive numbers are meaningful.
            If attribute "drop" is specified and is set to true then
            connections over specified limit will be dropped. Otherwise
            they will be put on hold.
```

```
            </xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
            <xsd:extension base="xsd:int">
                <xsd:attribute name="drop" type="xsd:boolean" use="optional"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>

</xsd:schema>
```

C.4 TLS MCC Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.nordugrid.org/schemas/tls/2009/08"
    xmlns="http://www.nordugrid.org/schemas/tls/2009/08"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">

<xsd:element name="KeyPath" type="xsd:string" default="/etc/grid-security/hostkey.pem">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
        Location of private key.
        Default is /etc/grid-security/hostkey.pem for service
        and $HOME/.globus/userkey.pem for client.
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>

<xsd:element name="CertificatePath" type="xsd:string" default="/etc/grid-security/
    hostcert.pem">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
        Location of public certificate.
        Default is /etc/grid-security/hostcert.pem for service and
        $HOME/.globus/usercert.pem for client.
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>

<xsd:element name="ProxyPath" type="xsd:string">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
        Location of proxy credentials - includes certificates,
        key and chain of involved certificates. Overwrites
        elements KeyPath and CertificatePath. Default is /tmp/hash{userid}.0 for client,
        and none for service.
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>

<xsd:element name="CACertificatePath" type="xsd:string">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
        Location of certificate of CA. Default is none.
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>

<xsd:element name="CACertificatesDir" default="/etc/grid-security/certificates">
    <xsd:complexType>
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
            Directory containing certificates of accepted CAs.
            Default is /etc/grid-security/certificates
            </xsd:documentation>
        </xsd:annotation>
```

```
            <xsd:simpleContent>
                <xsd:extension base="xsd:string">
                    <xsd:attribute name="PolicyGlobus" type="xsd:boolean" use="optional"
                        default="false"/>
                </xsd:extension>
            </xsd:simpleContent>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="ClientAuthn" type="xsd:boolean" default="true">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
             Whether checking client certificate. Only needed for service side.
            Default is "true"
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <xsd:element name="VOMSCertTrustDNChain">
        <xsd:complexType>
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                The DN list of the trusted voms server credential;
                in the AC part of voms proxy certificate, voms proxy
                certificate comes with the server certificate which is
                used to sign the AC. So when verifying the AC on the
                AC-consuming side (in ARC1, it is the MCCTLS which will
                consumes the AC), the server certificate will be checked
                against a trusted DN list. Only if the DN and issuer's
                DN of server certificate exactly matches the DN list
                in the configuration under TLS component, the AC can be trusted
                </xsd:documentation>
            </xsd:annotation>
            <xsd:sequence>
              <xsd:element name="VOMSCertTrustDN" type="xsd:string" minOccurs="0" maxOccurs
                  ="unbounded"/>
              <xsd:element name="VOMSCertTrustRegex" type="xsd:string" minOccurs="0"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="VOMSCertTrustDNChainsLocation" type="xsd:string">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
            DN list in an external file, which is in the same format as VOMSCertTrustDNChain
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <xsd:element name="Handshake" default="TLS">
        <xsd:simpleType>
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                Type of handshake applied. Default is TLS.
                </xsd:documentation>
            </xsd:annotation>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="TLS"/>
                <xsd:enumeration value="SSLv3"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:element>

    </xsd:schema>
```

C.5 GSI MCC Schema

```
    <?xml version="1.0" encoding="UTF-8"?>
    <xsd:schema
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
    targetNamespace="http://www.nordugrid.org/schemas/gsi/2009/08"
    xmlns="http://www.nordugrid.org/schemas/gsi/2009/08"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">

<xsd:element name="KeyPath" type="xsd:string">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
        Location of private key.
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>

<xsd:element name="CertificatePath" type="xsd:string">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
        Location of public certificate.
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>

<xsd:element name="ProxyPath" type="xsd:string">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
        Location of proxy credentials - includes certificates,
        key and chain of involved certificates. Overwrites
        elements KeyPath and CertificatePath.
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>

</xsd:schema>
```

C.6 ARC Authorization Security Handler Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
   xmlns:authz="http://www.nordugrid.org/schemas/arcauthz/2009/08"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.nordugrid.org/schemas/arcauthz/2009/08"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">

 <!-- This schema defines elements which are accepted by ArcAuthZ
      SecHandler. See mcc.xsd for general information about SecHandler
      elements. ArcAuthZ plugin is expected to be used to evaluate
      authorization decision by combaining multiple PDP plugins.
      Then called it delegates all security processing to specified PDPs.
      Those are called sequentially till positive answer obtained or list
      is exausted. -->

 <xsd:complexType name="PluginsType">
  <!-- This element is used to load PDP plugins. There may be multiple
       or none such element. -->
  <xsd:sequence>
   <!-- Name of library containing PDP plugins -->
   <xsd:element name="Name" type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:element name="Plugins" type="authz:PluginsType"/>

 <xsd:complexType name="PDPType">
  <!-- This element contains configuration of PDP to be used. There may be
       multiple or none such element. This element will be passed to PDP
       plugin during initialization. -->
  <xsd:sequence>
   <xsd:any namespace="##other" processContents="strict" minOccurs="0" maxOccurs="
       unbounded"/>
  </xsd:sequence>
  <!-- Attribute 'name' contains name of plugin as defined in one of
```

```
        loaded libraries. -->
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <!-- Attribute 'id' contains identifier which is used to distinguish
       among plugins. -->
  <xsd:attribute name="id" type="xsd:string" use="optional"/>
  <!-- Attribute 'action' defines behavior after obtaining results from
       each PDP. Possible values and their meanings are:
         breakOnFailure - stop processing in case of negative result.
       Result of SecHandler is negative too. In case of positive
       result continue to next PDP.
         breakOnAllow - stop processing in case of positive result.
       Result of SecHandler is positive too. In case of negative
       result continue to next PDP.
         breakAlways - stop processing immediately and use obtained
       result as result of SecHandler.
         breakNever - continue to next PDP.
       In any case if there is no next PDP in chain last result is
       used as result of SecHandler. Default behavior is breakOnAllow.
  -->
  <xsd:attribute name="action" use="optional" default="breakOnDeny">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="breakOnDeny"/>
        <xsd:enumeration value="breakOnAllow"/>
        <xsd:enumeration value="breakAlways"/>
        <xsd:enumeration value="breakNever"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:anyAttribute namespace="##other" processContents="strict"/>
 </xsd:complexType>
 <xsd:element name="PDP" type="authz:PDPType"/>

</xsd:schema>
```

C.7 UsernameToken Security Handler Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
   xmlns:ut="http://www.nordugrid.org/schemas/usernametokensh/2009/08"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.nordugrid.org/schemas/usernametokensh/2009/08"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">

  <!-- This schema defines elements which are accepted by UsernameToken
       SecHandler. See mcc.xsd for general information about SecHandler
       elements. UsernameTokenSH plugin is expected to be used to collect
       Security Attributes for messages comming to service and to form
       proper Username Token SOAP Header for client outgoing messages. -->

  <xsd:element name="Process" type="ut:ProcessType"/>
  <xsd:simpleType name="ProcessType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
          This element defines either Username Token is extracted
          from SOAP header or generated using other configuration elements.
          Type of the processing of Username Token to
          SOAP message: extract or generate.
          It is needed for both client and service side.
          Default is none.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="extract"/>
      <xsd:enumeration value="generate"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:element name="PasswordEncoding" type="ut:PasswordEncodingType"/>
```

```
    <xsd:simpleType name="PasswordEncodingType">
      <xsd:annotation>
          <xsd:documentation xml:lang="en">
            The encoding type of the password (one part of
            UsernameToken): text or digest.
            only needed for client side.
            Default is none.
          </xsd:documentation>
      </xsd:annotation>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="text"/>
        <xsd:enumeration value="digest"/>
      </xsd:restriction>
    </xsd:simpleType>

    <xsd:element name="Username" type="xsd:string">
      <xsd:annotation>
          <xsd:documentation xml:lang="en">
            The Username element of the token.
            only needed for client side.
            Default is none.
          </xsd:documentation>
      </xsd:annotation>
    </xsd:element>

    <xsd:element name="Password" type="xsd:string">
      <xsd:annotation>
          <xsd:documentation xml:lang="en">
            The Password element of the token.
            only needed for client side.
            Default is none.
          </xsd:documentation>
      </xsd:annotation>
    </xsd:element>

    <!-- External source for password to read from. There should be a file name here. -->
    <xsd:element name="PasswordSource" type="xsd:string">
      <xsd:annotation>
          <xsd:documentation xml:lang="en">
            Location of external source for password to read from.
            The content of the file should be like:
              user1, password1
              user2, password2
            only needed for service side.
            Default is none.
          </xsd:documentation>
      </xsd:annotation>
    </xsd:element>

  </xsd:schema>
```

C.8 X.509 Token Security Handler Schema

```
  <?xml version="1.0" encoding="UTF-8"?>
  <xsd:schema
     xmlns:xt="http://www.nordugrid.org/schemas/x509tokensh/2009/08"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     targetNamespace="http://www.nordugrid.org/schemas/x509tokensh/2009/08"
     elementFormDefault="qualified"
     attributeFormDefault="unqualified">

    <!-- This schema defines elements which are accepted by X509Token
         SecHandler. See mcc.xsd for general information about SecHandler
         elements. X509TokenSH plugin is expected to be used to collect
         Security Attributes for messages comming to service and to form
         proper X509 Token SOAP Header for client outgoing messages. -->

    <xsd:element name="Process" type="xt:ProcessType"/>
    <xsd:simpleType name="ProcessType">
      <xsd:annotation>
```

```
          <xsd:documentation xml:lang="en">
            This element defines either X509 Token is extracted
            from SOAP header or generated using other configuration elements.
            Type of the processing of X509 Token to
            SOAP message: extract or generate .
            Default is none.
          </xsd:documentation>
      </xsd:annotation>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="extract"/>
        <xsd:enumeration value="generate"/>
      </xsd:restriction>
    </xsd:simpleType>

    <xsd:element name="TokenUsage" type="xt:TokenUsageType"/>
    <xsd:simpleType name="TokenUsageType">
      <xsd:annotation>
          <xsd:documentation xml:lang="en">
            The usage of the X509 Token: signature or encryption.
            Default is signature.
          </xsd:documentation>
      </xsd:annotation>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="signature"/>
        <xsd:enumeration value="encryption"/>
      </xsd:restriction>
    </xsd:simpleType>

    <xsd:element name="KeyPath" type="xsd:string">
      <xsd:annotation>
          <xsd:documentation xml:lang="en">
            The location of private key which is used to sign the
            SOAP message, only needed by the client side.
            Default is none.
          </xsd:documentation>
      </xsd:annotation>
    </xsd:element>

    <xsd:element name="CertificatePath" type="xsd:string">
      <xsd:annotation>
          <xsd:documentation xml:lang="en">
            The location of certificate, the certificate is used to be as
            one part of X509 Token, only needed by the client side.
            Default is none.
          </xsd:documentation>
      </xsd:annotation>
    </xsd:element>

    <xsd:element name="CACertificatePath" type="xsd:string">
      <xsd:annotation>
          <xsd:documentation xml:lang="en">
            The location of the file of trusted CA certificate, the
            certificate is used for verifying the signature to SOAP message.
            Only needed by the service side.
            Default is none.
          </xsd:documentation>
      </xsd:annotation>
    </xsd:element>

    <xsd:element name="CACertificatesDir" type="xsd:string" default="/etc/grid-security/
        certificates">
      <xsd:annotation>
          <xsd:documentation xml:lang="en">
            The location of the directory that contains trusted CA certificates,
            the certificates are used for verifying the signature to SOAP message.
            Only needed by the service side.
            Default is "/etc/grid-security/certificates".
          </xsd:documentation>
      </xsd:annotation>
    </xsd:element>

  </xsd:schema>
```

C.9 SAML Token Security Handler Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
   xmlns:xt="http://www.nordugrid.org/schemas/samltokensh/2009/08"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.nordugrid.org/schemas/samltokensh/2009/08"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">

  <!-- This schema defines elements which are accepted by SAMLToken
       SecHandler. See mcc.xsd for general information about SecHandler
       elements. SAMLTokenSH plugin is expected to be used to collect
       Security Attributes for messages comming to service and to form
       proper SAML Token SOAP Header for client's outgoing messages.
       When client needs to contact some 3rd-party authority to get back
       a SAML assertion (compliant to the hold-of-key subject confirmation
       method), by authenticating through TLS; and then uses this SAML
       assertion to protect the SOAP message that will be sent to the
       service side -->

  <xsd:element name="Process" type="xt:ProcessType"/>
  <xsd:simpleType name="ProcessType">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
          This element defines either SAML Token is extracted
          from SOAP header or generated using other configuration elements.
          Type of the processing of SAML Token to
          SOAP message: extract or generate.
          It is needed for both client and service side.
          Default is none.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="extract"/>
      <xsd:enumeration value="generate"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:element name="KeyPath" type="xsd:string">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
          The location of private key which is used to sign the
          SOAP message, only needed by the client side.
          Default is none.
        </xsd:documentation>
    </xsd:annotation>
  </xsd:element>

  <xsd:element name="CertificatePath" type="xsd:string">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
          The location of certificate, the public key parsed from
          certificate is used to be as one part of SAML Token:
          <Assertion><Subject><SubjectConfirmation><SubjectConfirmationData><KeyInfo><
              KeyValue>
            public key
          </KeyValue></KeyInfo></SubjectConfirmationData></SubjectConfirmation></Subject
              ></Assertion>
          Only needed by the client side.
          Default is none.
        </xsd:documentation>
    </xsd:annotation>
  </xsd:element>

  <xsd:element name="CACertificatePath" type="xsd:string">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
          The location of the file of trusted CA certificate, the
          certificate is used for verifying the signature to SOAP message.
          Needed by client and service side.
          Default is none.
        </xsd:documentation>
```

```
      </xsd:annotation>
    </xsd:element>

    <xsd:element name="CACertificatesDir" type="xsd:string" default="/etc/grid-security/
        certificates">
      <xsd:annotation>
          <xsd:documentation xml:lang="en">
            The location of the directory that contains trusted CA certificates,
            the certificates are used for verifying the signature to SOAP message.
            Needed by client and service side.
            Default is "/etc/grid-security/certificates".
          </xsd:documentation>
      </xsd:annotation>
    </xsd:element>

    <xsd:element name="AAService" type="xsd:string">
      <xsd:annotation>
          <xsd:documentation xml:lang="en">
            Endpoint of the attribute authority service.
            AA (attribute authority) service is an external third-party service
            that is used for authenticate the requestor(client) and signing
            SAML Token with requestor's attributes embedded.
            Needed by client side.
            Default is none.
          </xsd:documentation>
      </xsd:annotation>
    </xsd:element>

  </xsd:schema>
```

C.10 ARC PDP Schema

```
  <?xml version="1.0" encoding="UTF-8"?>
  <xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.nordugrid.org/schemas/arcpdp/2009/08"
    xmlns:pdp="http://www.nordugrid.org/schemas/arcpdp/2009/08"
    targetNamespace="http://www.nordugrid.org/schemas/arcpdp/2009/08"
    elementFormDefault="qualified">


      <xsd:element name="Filter">
      <xsd:complexType>
          <xsd:annotation>
              <xsd:documentation xml:lang="en">
                This element defines Security Attributes to select and reject.
                If there are no Select elements all Attributes are used except
                those listed in Reject elements.
              </xsd:documentation>
          </xsd:annotation>
          <xsd:sequence>
              <xsd:element name="Select" type="xsd:string" minOccurs="0" maxOccurs="
                  unbounded"/>
              <xsd:element name="Reject" type="xsd:string" minOccurs="0" maxOccurs="
                  unbounded"/>
          </xsd:sequence>
      </xsd:complexType>
      </xsd:element>

      <xsd:element name="PolicyStore">
      <xsd:complexType>
          <xsd:annotation>
              <xsd:documentation xml:lang="en">
                This element specifies file containing policy document.
                There can be multiple sub elements <Location/>.
              </xsd:documentation>
          </xsd:annotation>
          <xsd:sequence>
              <xsd:element name="Type" type="xsd:string" minOccurs="1" maxOccurs="1"/>
              <xsd:element name="Location" type="xsd:string" minOccurs="1" maxOccurs="1"/>
```

```
            </xsd:sequence>
        </xsd:complexType>
        </xsd:element>

        <xsd:element name="Policy">
        <xsd:complexType>
        </xsd:complexType>
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                  This element contains policy to be processed.
                  There may be nultiple such elements.
                </xsd:documentation>
            </xsd:annotation>
            <xsd:sequence>
                <xsd:any namespace="##other" processContents="strict" minOccurs="0"
                    maxOccurs="unbounded"/>
            </xsd:sequence>
            <xsd:anyAttribute namespace="##other" processContents="strict"/>
        </xsd:complexType>
        </xsd:element>

        <xsd:element name="PolicyCombiningAlg" type="xsd:string">
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                   Combining Algorithm for Policies.
                   For supported names please check documentation. In addition to those
                       there are
                   also few legacy algorithms provided:
                       EvaluatorFailsOnDeny,
                       EvaluatorStopsOnDeny,
                       EvaluatorStopsOnPermit,
                       EvaluatorStopsNever.
                       Unfortunately they behavior is not well defined.
                </xsd:documentation>
            </xsd:annotation>
        </xsd:element>

    </xsd:schema>
```

C.11 XACML PDP Schema

```
    <?xml version="1.0" encoding="UTF-8"?>
    <xsd:schema
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns="http://www.nordugrid.org/schemas/xacmlpdp/2009/08"
      targetNamespace="http://www.nordugrid.org/schemas/xacmlpdp/2009/08"
      elementFormDefault="qualified">

        <xsd:element name="PolicyStore" type="PolicyStoreType">
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                  This element specifies file containing policy document.
                  There could be multiple sub elements <Location/>.
                </xsd:documentation>
            </xsd:annotation>
        </xsd:element>
        <xsd:complexType name="PolicyStoreType">
            <xsd:sequence>
                <xsd:element name="Location" type="LocationType" minOccurs="1" maxOccurs="
                    unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="LocationType">
            <xsd:attribute name="type">
                <xsd:annotation>
                    <xsd:documentation xml:lang="en">
                    This attribute is to specify the type of policy soure which
                    will be used when parsing policy.
                    </xsd:documentation>
                </xsd:annotation>
```

```
            <xsd:simpleType>
                <xsd:restriction base="xsd:string"/>
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>

</xsd:schema>
```

C.12 Delegation PDP Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.nordugrid.org/schemas/delegationpdp/2009/08"
  targetNamespace="http://www.nordugrid.org/schemas/delegationpdp/2009/08"
  elementFormDefault="qualified">

    <xsd:complexType name="FilterType">
        <!-- This element defines Security Attributes to select and reject.
            If there are no Select elements all Attributes are used except
            those listed in Reject elements.  -->
        <xsd:sequence>
            <xsd:element name="Select" type="xsd:string" minOccurs="0" maxOccurs="
                unbounded"/>
            <xsd:element name="Reject" type="xsd:string" minOccurs="0" maxOccurs="
                unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="Filter" type="FilterType"/>

</xsd:schema>
```

C.13 Service Invoker PDP Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.nordugrid.org/schemas/pdpserviceinvoker/2009/08"
  targetNamespace="http://www.nordugrid.org/schemas/pdperviceinvoker/2009/08"
  xmlns:pdp="http://www.nordugrid.org/schemas/pdperviceinvoker/2009/08"
  elementFormDefault="qualified">

    <xsd:complexType name="FilterType">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
              This element defines Security Attributes to select and reject.
              If there are no Select elements all Attributes are used except
              those listed in Reject elements.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="Select" type="xsd:string" minOccurs="0" maxOccurs="
                unbounded"/>
            <xsd:element name="Reject" type="xsd:string" minOccurs="0" maxOccurs="
                unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="Filter" type="pdp:FilterType"/>

    <xsd:element name="ServiceEndpoint" type="xsd:string">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
              This element is to specify endpoint about remote pdpservice.
              It will be configured under the <PDP name="pdpservice.invoker"/>
            </xsd:documentation>
        </xsd:annotation>
```

```
        </xsd:element>

        <xsd:element name="RequestFormat" type="xsd:string" default="arc">
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                 This element is to specified the format of request. Two options
                 are recognized: xacml, arc.
                 Default is "arc".
                </xsd:documentation>
            </xsd:annotation>
        </xsd:element>

        <xsd:element name="TransferProtocol" type="xsd:string" default="arc">
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                  This element is to specified the protocol for transfering request.
                  Default is the arc specific protocol; if "SAML" is specified, then
                  the "SAML2.0 profile of the XACML v2.0" will be used for carrying
                  request.
                  If RequestFormat is specified to XACML, and Transfer is specified to SAML,
                  then this pdpservice invoker is able to interact with third-party pdp
                  service, such as the GLite authorization service.
                  Two options are recognized: saml, arc.
                 Default is "arc".
                </xsd:documentation>
            </xsd:annotation>
        </xsd:element>

        <!--The following information is about credential. Since the pdpserviceinvoker is
        actually a client which will invoke the remote pdpservice, it should has its
            credential
        configuration, but it could also use the credential of the MCCTLS which is in the
            same
        chain as this uplevel service (which is using the pdpserviceinvoker) configuration.
        If the latter situation applies, the following element don't not need to be
            configured
        under the <PDP name="pdpservice.invoker"/>.
        However, the credential of the MCCTLS in the main chain and the credential of the
        pdpservice.invoker should not be coupled, since pdpservice could use tls, but this
            normal
        service (which configures the pdpservice.invoker inside) could not use tls, and vice
             versa.-->


        <xsd:element name="KeyPath" type="xsd:string" default="$HOME/.globus/userkey.pem">
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                   Location of private key.
                   Default is $HOME/.globus/userkey.pem.
                </xsd:documentation>
            </xsd:annotation>
        </xsd:element>

        <xsd:element name="CertificatePath" type="xsd:string" default="$HOME/.globus/
            usercert.pem">
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                 Location of public certificate.
                 Default is $HOME/.globus/usercert.pem.
                </xsd:documentation>
            </xsd:annotation>
        </xsd:element>

        <xsd:element name="ProxyPath" type="xsd:string" default="/tmp/hash{userid}.0">
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                 Location of proxy credentials - includes certificates,
                 key and chain of involved certificates. Overwrites
                 elements KeyPath and CertificatePath. Default is /tmp/hash{userid}.0
                </xsd:documentation>
            </xsd:annotation>
        </xsd:element>
```

```
    <xsd:element name="CACertificatePath" type="xsd:string">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
             Location of certificate of CA. Default is none.
            </xsd:documentation>
        </xsd:annotation>
    </xsd:element>

    <xsd:element name="CACertificatesDir" default="/etc/grid-security/certificates">
            <xsd:annotation>
                <xsd:documentation xml:lang="en">
                 Directory containing certificates of accepted CAs.
                 Default is /etc/grid-security/certificates
                </xsd:documentation>
            </xsd:annotation>
    </xsd:element>

</xsd:schema>
```

C.14 Simple List PDP Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
   xmlns:spdp="http://www.nordugrid.org/schemas/simplelistpdp/2009/08"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.nordugrid.org/schemas/simplelistpdp/2009/08"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">

 <!-- This schema defines configuration elements supported by
      SimpleListPDP PDP plugin. This plugin compares identifier
      found in TLS:PEERDN Message attribute (tipically Distinguished
      Name of client's certificate produced by TLS MCC) to list of
      identities stored in local file. Those are stored one per
      line and may be enclosed in '"' and any amount of space
      characters.
      Alternatively DNs may be directly listed as DN XML elements
      in configuration of PDP. -->

  <xsd:attribute name="location">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
         This attribute is to be used in top (and only) element of
         PDP configuration. It specified full path to file with list
         of identities to be matched.
         Default is none.
        </xsd:documentation>
    </xsd:annotation>
    <xsd:simpleType>
      <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
  </xsd:attribute>

  <xsd:element name="DN" type="xsd:string">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
         the list of dn that is directly specified by
         this pdp.
         Default is none.
        </xsd:documentation>
    </xsd:annotation>
  </xsd:element>

</xsd:schema>
```

References

[1] GACL Mini-Howto. Web site. URL http://www.nordugrid.org/documents/gacl_mini_howto.html.

[2] D. Cameron et al. *The Hosting Environment of the Advanced Resource Connector middleware*. URL http://www.nordugrid.org/documents/ARCHED_article.pdf. NORDUGRID-TECH-19.

[3] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997. Available at: http://www.globus.org.

[4] A. Konstantinov. *The ARC Computational Job Management Module – A-REX*. The NorduGrid Collaboration. URL http://www.nordugrid.org/documents/a-rex.pdf. NORDUGRID-TECH-14.

[5] I. Márton. *ARC Information System*. The NorduGrid Collaboration. URL http://www.nordugrid.org/documents/infosys_technical.pdf. NORDUGRID-TECH-21.

[6] A. McNab. The GridSite Web/Grid security system: Research Articles. *Softw. Pract. Exper.*, 35(9): 827–834, 2005. ISSN 0038-0644.

[7] Zs. Nagy. *The HOPI Manual*. The NorduGrid Collaboration. URL http://www.nordugrid.org/documents/hopi-manual.pdf. NORDUGRID-MANUAL-15.

[8] Zs. Nagy, J. K. Nilsen, and S. Z. Toor. *Chelonia Administrator's Manual*. The NorduGrid Collaboration. URL http://www.nordugrid.org/documents/arc-storage-manual.pdf. NORDUGRID-MANUAL-10.

[9] OASIS. Oasis web services security: Saml token profile 1.1. , February 2006. URL http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf.

[10] OASIS. Oasis web services security usernametoken profile 1.1. , February 2006. URL http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-UsernameTokenProfile.pdf.

[11] OASIS. Oasis web services security: X.509 token profile 1.0, errata 1.0. , December 2005. URL http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf.

[12] OASIS. Oasis extensible access control markup language. , February 2005. URL http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[13] W Qiang and A. Konstantinov. *Security Framework of ARC NOX*, . URL http://www.nordugrid.org/documents/arc-security-documentation.pdf. NORDUGRID-TECH-16.

[14] W. Qiang and A. Konstantinov. *Security framework of ARC NOX*. The NorduGrid Collaboration, . URL http://www.nordugrid.org/documents/arc-security-documentation.pdf. NORDUGRID-TECH-16.