Arrowhead Framework
Summer school 16-24 Aug 2017
Luleå University of Technology
Professor Jerker Delsing
Arrowhead Framework
Core systems and services

Prof. Jerker Delsing
Outline

1. Mandatory core systems and services
2. ServiceRegistry system
3. ServiceDiscovery service
4. Authorization system
5. Orchestration system
6. Automation support core systems
7. PlantDescription system
8. Configuration system
9. SystemRegistry and DeviceRegistry systems
10. EventHandler system
11. Historian system
12. Gatekeeper system
13. Translation system
Arrowhead Framework core services
Core services details

- Documented in,
  - Service Description (SD),
  - Interface Design Description (IDD),
  - Communication Profile (CP),
  - Semantic Profile (SP)
ServiceRegistry system
ServiceRegistry service interface

```
<abstract interface>
ServiceDiscovery

+ Lookup(query:Query): ServiceInstance[]
+ Publish(instance:ServiceInstance): void
+ Unpublish(instance:ServiceInstance): void
```

```
<interface>
ServiceDiscovery REST

+ Lookup(query:Query): ServiceInstance[]
+ Publish(instance:ServiceInstance): void
+ Unpublish(instance:ServiceInstance): void

tags

service-identifier = service-discovery_REST_WS-XML-SP-SDTR
```
ServiceRegistry service

ServiceDiscovery interfaces methods:

- **Publish**: The publish method is used to register services. The services will contain a symbolic name as well as a physical endpoint. The instance parameter represents the endpoint information that should be registered.

- **Un-publish**: The un-publish method is used to unregister a service that no longer should be used. The instance parameter contains information necessary to find the service to be removed.

- **Lookup**: The lookup method is used to find and translate a symbolic service name into a physical endpoint, IP address, and a port. The query parameter is used to request a subset of all the registered services fulfilling the demand of the requesting system. The returned listing contains service endpoints that fulfils the query.
Datatypes

ServiceRecord: with the following data:

- Endpoint - string
  This datatype implements a representation of an endpoint using DNS A-records:
   - Hostname is a String containing the name of the host in format: name.domain.topdomain, e.g., app.arrowhead.eu
   - path: 192.168.1.20
   - Port is an Integer containing the port number, e.g., 8070.
   - Edata is a String containing additional information related to the endpoint. Any additional information that is required to identify the service instance should be stored in the mandatory DNS TXT record.

* Metadata: key=value
  Metadata for the service are stored as key value pairs, could be, e.g., time to live, sleep period,
Datatypes - ServiceRecord Endpoint (cont)

- Metadata: key=value
  Metadata for the service are stored as key value pairs, could be, e.g., time to live, sleep period, configurations, payload encoding, compression and semantics. To allow for the orchestration to understand if any translation is necessary, it is proposed that the following three be mandatory:

- Encoding: e.g., encode=xml where XML [6] encoding is used and specified in the CP (Communication Profile) document

- Compression: e.g., comp=exi when EXI [7] compression is used and specified in the CP document

- Semantics: e.g., sem=senml where SenML [8] semantics is used and specified in the SP (Semantic Profile) document

- ServiceName - string
  Name of the service instance e.g. _Temp1

- ServiceType - string
  e.g. _ahf-temperature._coap._udp.
Datatypes

Query: with the following content:
  – Query — string
where the query string specifies one or several of the data types of the service. The query will then return a list of all registered services with the specified data type/s.
ServiceDiscovery sequence - publish
ServiceDiscovery sequence - unpublish
ServiceDiscovery sequence - Lookup
ServiceDiscovery (REST implementation)
Authorisation system
AA-Authorisation services

- AuthorisationControl
- AuthorisationManagement
AuthorisationControl service

- Interface: AuthorisationControlProvider:
- Methods: Authorise: with the data types:
  - AuthorisationRequest requests the authorisation for the system addressed by an endpoint string e.g. coap://192.168.2.30:8000/_viib3._ahf-vibration._udp. The end point as provided by the orchestration system.
  - AuthorisationResponse responds with a boolean (True/False) request.
AuthorisationManagement interface
AuthorisationManagement methods

- **AddAuthorisationRules(rules:AuthorisationRulesList)** is used to store new authorisation rules in the providing system.

- **ListAuthorisationRules(): AuthorisationRules** is used to present all the rules to the administrator of the authorisation ruling.

- **ListAvailableServiceInstances(type:String): StringList** is used to fetch all service instances that currently are stored in the authorisation system, in order to list consumer systems and producer systems. The purpose of this method is to allow an administrator to administer all systems.
AuthorisationManagement methods

- ListAvailableServiceTypes(): StringList is used to fetch the service types currently used in the arrowhead system-of-system, from the authorisation point of view.

- RemoveAuthorisationRule(rules:AuthorisationRulesList) is used to remove a rule that no longer is valid.

with the datatype:
- AuthorisationRules

The AuthorisationRules data type contains information about rules that the consumer should be matched against in order to determine if a producer should be releasing information.
The AuthorisationRules data type contains information about rules that the consumer should be matched against in order to determine if a producer should be releasing information.
AuthorisationManagement seq-diag

• addAuthorisationRules
Authorisation Management seq-diag

1: listAuthorisationRules()

2: RuleList
AAA-Authorisation services

- AuthenticationID
- AuthorisationControl
- AuthorisationManagement
AuthenticationID service interfaces

<<abstract data type>>
AuthenticationIDProvider
+ Authentication(req:Challenge-Response):AuthenticationResponse

<<abstract data type>>
AuthenticationRequest
(from Data)

<<abstract data type>>
ChallengeRequest
+ Challenge:
  - Authenticator: Binary
  - Timeout: Integer
(from Data)

<<abstract data type>>
ChallengeResponse
+ Response:
  - Encoded Password: Binary
(from Data)

<<abstract data type>>
TicketResponse
+ TicketInfo:
  - Ticket: Binary
  - Timeout: Integer
(from Data)
AuthenticationID service interfaces

<<abstract>>
AuthorizationControlProvider
+ Authorize(req: AuthorizationRequest): AuthorizationResponse

<<abstract data type>>
AuthorizationRequest
+ Consumer:
  • Ticket: Binary
  • IP: String
+ Producer:
  • Ticket: Binary
  • IP: String
(from Data)

<<abstract data type>>
AuthorizationResponse
+ AuthorizationResult:
  • Valid: Boolean
  • Timeout: Integer
  • ExpireTime: Integer
  • LastLogin: Integer
(from Data)
AuthenticationID service interfaces

<<abstract>>
AuthorizationManagementProvider
+ Change(req:ChangeRequest):ChangeResponse

<<abstract data type>>
ChangeRequest
+ AccountInfo:String (from Data)

<<abstract data type>>
ChangeResponse
+ Status:boolean (from Data)
Orchestration system
Orchestration services

- OrchestrationStore
- OrchestrationPush
- OrchestrationCapability
- OrchestrationManagement
OrchestrationStore service
OrchestrationPush
OrchestrationPush sequence diagram
Orchestration Capability
Orchestration Management
OrchestrationManagement seq. diagram
Support core systems and services
PlantDescription system
PlantDescription services

- GetViewpoint(Type): PlantData
  - Get all nodes and links that correspond to this type.
  - Datatype: Type - string

- GetObjectsByNode(NodeId): PlantData
  - Get all nodes linked to the specified node, and those links
  - Datatype: NodeId - string

- GetObjectsByTypeId(Type, NodeTypeId): PlantData
  - The function GetObjectsByTypeId is used with a specific combination of a Type and a NodeTypeId corresponding to the identity by which the node is identified according to that type. The function returns the same information as the GetObjectsByNode for that node.
  - Datatype: Type and NodeIDType - string
PlantDescription services datatype

PlantData:

- **Nodes**: Node
- **Links**: Link

**Node**:
- NodeId
- NodeName
- NodeTypes
- NodeTypelds

**Link**:
- LinkId
- LinkSource
- LinkTarget
- LinkTypes
- BiDirectional
Configuration system
Configuration services

- ConfigurationStore
- ConfigurationManagement
SystemRegistry and DeviceRegistry systems
SystemRegistry service

<<abstract service>>
SystemDiscovery

<<abstract interface>>
SystemDiscovery
+Lookuo(query: Query): ServiceInstance[]
+Publish(instance: ServiceInstance): void
+Unpublish(instance: ServiceInstance): void

<<abstract data type>>
SystemRecord
+Endpoint: String
+Metadata: String
+SystemName: String
+SystemType: String

<<abstract data type>>
Query
+Query: String
SystemRegistry - lookup
SystemRegistry - unpublish
DeviceRegistry service

**DeviceDiscovery**

- Abstract service
- Abstract interface

- Abstract data type: DeviceRecord
  - Endpoint: String
  - Metadata: String
  - DeviceName: String
  - DeviceType: String

- Abstract data type: Query
  - Query: String

Methods:
- Lookup(query: Query): ServiceInstance[]
- Publish(instance: ServiceInstance): void
- Unpublish(instance: ServiceInstance): void
QoSManager system
QoSManager services
Historian system
Historian services

- Historian
- FileSyS
- Filter
Historian interfaces
Historian sequence diagram - PutData
Filesys interface

<<service>>
FileSync...

<<abstract service>>
Store
+ storeFile(origin): File

<<abstract service>>
Retrieve
+ getFile(f): File
+ getFileProperties(f): Props
+ deleteFile(f):
Historian sequence diagram - getFile
Gatekeeper system
Gatekeeper system

- Global service discovery
- Inter-cloud negotiations

Service Request from the Application System

Local Cloud Orchestration

Global Service Discovery

Choosing the Partnering Cloud

Inter-Cloud Negotiations

Application System connects to the Partnering Cloud

Forwarding the Entry Token
Gatekeeper negotiation process

1. Protocol negotiations (Arrowhead Framework version and protocol matching)
2. Mutual authentication and identity checking of the parties
3. Actual admission control (authorisation and resource allocation for the transaction in the clouds)
4. Establishing a secure connection between the Gatekeepers, therefore the Arrowhead clouds (creating the data path)
5. Sharing the local ServiceDiscovery data of the partnering cloud (addressing the partnering cloud from the outside)
6. Injecting temporary authorization information into the Authorisation system in the partnering cloud (access control of the “foreigner” system)
7. The partnering cloud Gatekeeper issues a token for the requester with temporary entry and shares it with the requesting Gatekeeper
Gatekeeper negotiation process

7. The partnering cloud Gatekeeper issues a token for the requester with temporary entry and shares it with the requesting Gatekeeper.
8. The requesting local cloud forwards this token to the requesting system,
9. The system connects to the producer using the partnering cloud’s core Systems.
Translation system
Protocol mismatch
Translation service

Simple Service  Authorization Orchestration
Registry Service  Service  Service

Translation System

Protocol B
Protocol !N

Protocol A
Protocol N
Translator block diagram
Translation service

The Translation service is used to instantiate new translation instances.
The interface definition can be seen in Figure 4.45. It takes the service provider service registry record and the service consumer type reference.
Translation interface

Takes the service provider service registry record and the service consumer type reference.