Architecture WG

Arrowhead Framework

Purpose of the meeting

1. Collect issues and new systems to address in upcoming versions of Arrowhead
2. Prioritise architecture updates
3. Assign who is in charge of concrete update proposal
4. AoB

ARROWHEAD TOOLS

Agenda

1. Decision mechanisms, Emanuel …

2. Update proposals
    1. Configuration files for core systems, Jacob
    2. Arrowhead standard for IDD meta data
    3. Time format to adopt or Time format translation
    4. Naming convention,
        Underscores in the service names
    5. Certificate structure
    6. Systems are responsible for it’s own data - update needed to some core systems
    7. Orchestration system update to address PlantDescription possibilities
    8. Replication of ServiceRegistry data between local clouds
    9.

ARROWHEAD TOOLS

Identified issues …..

Replication of ServiceRegistry data between local clouds

SysML modelling

- Aligning of models with architecture and existing code
- Aligning with engineering process models
- Code generation

ARROWHEAD TOOLS

Identified issues …..

Service versioning

- Semantics versioning standard 2.0.0

Service interface compliancy?

- In relation to service versions
- Version to be part of the metadata field
- Version # indicates which IDD the service is producing

Support for other protocols

- UNIQE SOCKET SERVICE
- AMPQ does not use path

ARROWHEAD TOOLS

Database separation

Currently the core systems use the same database, because they have foreign keys pointing to records in other core systems tables.Eg. A system-s data is stored in the “_system” table and SR, AUTH, ORCH, EH are using it. Currently it is always consistent. The question we should discuss is: how can we separate the core systems database tables into separate databases, the way, that they will remain consistent.

Orchestration system update

In the telco with the guys doing the Plant Description Engine, they wanted to create orchestration store rules, based on partial information. Eg. We don’t have all the data of a system, (address, port -> are known after startup of a service). So they would like to create orchestration store rules based on metadata, and/or system name. The question we should discuss: What kind of functionality do you miss from the orchestration?

ARROWHEAD TOOLS

From Fredrik B

1. Agree on core-system separation. Should fulfil separation and be loosely coupled. Each system shall be autonomously and possible to run on its own. The system shall provide its services in the local cloud (network) ready to be utilized together with other systems, managed and controlled by responsible it-operations - or AI!

2. The REST-API shall be design according to the REST-standard. Not following the standard. Shall use nouns, not verb. Shall be built on resources. CRUD shall preferred be used together with REST – it will create a clean and strong Service (API/Interface).

3. The REST-API (Service) for the ServiceRegistry consist of both Services and Systems. These should be separated and its own services (own API:s, each handle its specific information). The end solution could be one System that serves the both API:s but, the design shall follow that principle. And the API:s shall be separated. Preferred shall the IDD:s (Services) Capability and Status be used for all existing Systems (see London 2015). Capability is a Service that provides all its Producers and/or Consumers. To build a complete System-of-system-picture the only service needed to consume is all the existing “Capability”-instances in the Service Registry… The Status service gives, for example, the used resources for the existing Systems instances…

ARROWHEAD TOOLS

From Fredrik B

4. The DNS-SD shall be handled according as attached presentation. Each local cloud should be connected full-mesh to all others. Preferred is also to use some kind of secure network (VPN) and DMZ in each local cloud. Then we will have security at network level, communication link (TLS) and authentication/authorization.
   The already existing DNS-SD security should be enough if used according to this. See attached pdf (sorry didn't found the Arrowhead-presentation from 2014).
   1. Every local cloud, administration personnel – IT operations, connect each external-"local cloud" to a corresponding zone (configured) in its own local cloud SR.
   2. Every local cloud publish the services decided to each and every other external-"local cloud" by register it in the corresponding configured zone.
   3. Every local cloud decide which external-"local cloud" service that should be available internally at the own local cloud
   4. To manage this, a API Management (proxy services, in a DMZ) should be used. Either custom created. Or custom adopted existing API GW:s (for example WSO2 suite – GW part).
      By use of DNS-SD in this way security should be enough in combination of TSIG. Note: If decided the steps above could be utilized automatically, but we have not - yet - seen any organisation approve this according to security principles. Internally within one organisation, that have several trusted sub-local-clouds, we have seen this (created a custom proprietary solution for this).
      Background: 21 different countries used this approach successfully in an European collaboration maritime (sea) surveillance project, called MARSUR. All nations (member states) with different security regulations and requirements.

ARROWHEAD TOOLS

From Fredrik B

5. Services shall be strictly using IDD:s to represent the Service Type. We shall aim to use as few technologies as possible when we have decided and approved a service. The risk if we have several technologies (IDD:s) that solves the same Service is that we will end with difficulties to achieve interoperability. Then we need to create lot of adapters, bridges and have these manged by it-operations. As few as possible to achieve the same information exchange. Decide for one way to go - and agree. Of course we can have, and will have, several different technologies for different purposes. This is more or less rule number one in the AF concept. The service contract (IDD) shall not be, never ever, be changed after approved ("in the best of the worlds") and some instance has been deployed. Of course we need to handle versioning and updates, new services. But as far as we can as few IDD's as possible! Really really important. Stop creating many services with different technologies. We are not creating yet another Integration Platform – the Arrowhead Framework is a light weight conceptual discipline to follow… Sorry about this point, but I feel it's really important to discuss it so we all agree, it's the fundamental core…

ARROWHEAD TOOLS

New core system

Interoperability with e.g.

- Enterprise Service Bus
- NodeRed
- BPMN - Semantics for building processes

ARROWHEAD TOOLS

Cristina

Field interface, issue in GitHub.

ARROWHEAD TOOLS