Design Decisions - Coordination layer

Important design decisions

- Overall Description
- Message Broker
  - Communication
    - Asynchronous communication
    - Start communication via Topic
    - Atomic messaging in simple operations
    - Conversation Id
    - Reuse and overlap between operations
    - Paging of requests
  - Software
    - Independence
- Data Transmission
- Paging
- Certifications

Overall Description

The coordination layer offers communication through a message broker and a data transmission via https. This is illustrated in the below figure. In general communication is initiated and actions are coordinated via the message broker, while the actual transmission of data is done via data transmission.

There is no intelligence in the Coordination Layer and it is the clients that are responsible for choices that involves intelligence e.g. in choosing an alternative pillar for request of data (according to Service Level Agreement).

Message Broker

It is assumed that

- the underlying low level protocol for message transmission is specified as part of the message broker, and thus this will not be a addressed directly in this architecture
- the message broker software is persistent

Communication

Asynchronous communication

All communication is asynchronous, unless it is explicitly specified to be synchronous. This enables return of answers of transactions to be in arbitrary order.

Start communication via Topic

All communication via the Coordination layer starts via a Topic that is send to all. It is up to the individual subscribers to determine whether it is a message that concerns them, for example a GetTime message only concern the pillars involved in a certain service level agreement.

<TBD> An example can be found in http://wiki.statsbiblioteket.dk/BitmagasinWiki/Sekvens1

DIAGRAM NOT UPDATED!

Atomic messaging in simple operations
An operation is defined as the communication that takes place in order to make an operation on the bit repository. Examples are get and put of data.

Operations must be as simple as possible. They will be based on simple message primitives and data transmission. A primitive is a single request message to the system and the following replies. The primitives are defined on basis of atomic well-defined messages. This serves to give simplicity, and to get as little overlap between different operations as possible.

An example of the Get operation is given in the below figure.

Here the full communication will result in an operation. The message communication is divided into two parts, each representing a primitive (identification of pillars and the actual get).

Note that the primitive for identification of pillars can cover request to several pillars, while the primitive for the actual get part only concerns one pillar.

Note also that there can be a number of different responses, in cases where the pillar want to give information of the progress of the operation at pillar level. This can for instance be interesting for DVD pillars where there are several possibly time consuming phases in finding the data - even more so for a put operation.

**Conversation Id**

We generate a unique conversation id at initiation of a conversation such that it is possible to identify which conversation a message belongs to.

**Reuse and overlap between operations**

The operations are designed to have as many similarities and overlapping definitions as possible. For instance the getChecksums must work similar to Get. Paging of data and parameters must work in the same way. See discussion on [http://sbforge.org/display/BITMAG/Coordination+layer+design+discussion#Coordinationlayerdesigndiscussion-Overlapmedprotokolspecification](http://sbforge.org/display/BITMAG/Coordination+layer+design+discussion#Coordinationlayerdesigndiscussion-Overlapmedprotokolspecification).

**Paging of requests**

It must be possible to make paging of request parameters. Paging of parameters can for instance be the case for object ids in a random check of files. It is needed since we cannot assume that clients can make prior pagination. See discussion on [http://sbforge.org/display/BITMAG/Coordination+layer+design+discussion#Coordinationlayerdesigndiscussion-Pagingafrequests](http://sbforge.org/display/BITMAG/Coordination+layer+design+discussion#Coordinationlayerdesigndiscussion-Pagingafrequests).

**Software**

**Independence**

In the design we need to ensure that we are as independent of the software chosen as possible. If special features are used, e.g. for optimisation, then the design must be flexible enough to skip use of the special features at a later stage.

**Data Transmission**

It is assumed that

- it is sufficient to identify a data transmission transaction by a token
- the token is sufficient information as basis for completion of a data transmission

**Paging**
Paging is in the first versions managed by a simple mechanism where a segment of a file is identified by parameters. Any Get command can contain an offset along with a length parameter, which defines the segment of the file. This is also sufficient to handle receipt of divided files, and thus enabling rudimentary support of restart of an interrupted transmission.

**Certifications**

Encryption rules and credentials must be specified in special unit in organisation. This is partly treated in configuration of SLA data.