Design Considerations for a DECADE SDT

draft-kutscher-decade-protocol-00

Dirk.Kutscher@neclab.eu
Martin Stiemerling@neclab.eu
Jan Seedorf@neclab.eu

IETF-82, Taipei
DECADE WG
Background

• DECADE architecture describes DECADE protocols **conceptually**
  – Assumption: will need one or more concrete protocol specs at some point

• Standard Data Transport
  – Conceptual data transport protocol

• DECADE Resource Control Protocol
  – Resource tokens for authorization, resource control
DECADE Architecture Elements

• Standard Data Transport: conceptual data transport protocol
  – Expected to leverage existing transport / application protocols

• DECADE Resource Control Protocol: resource tokens for authorization, resource control
  – Not an actual protocol
  – Intended to be used with an SDT instantiation

• Naming
  – Want to name resources globally uniquely
  – Same name for all replicas of a resource (on different servers)
Some considerations on
  – Conceptual DECADE protocols
  – Naming – leveraging NI URI scheme
  – Authentication and access control
  – General SDT considerations
  – CDMI as an SDT instantiation

Distilled those into a list of recommendations in the draft

Motivation: have a basis for discussion and re-charting
Conceptual DECADE Protocols

• SDT and DRP split

• We assume that we would need exactly one DRP scheme
  – That can then be used for different (all) SDT instantiations
  – Issue: some SDT candidates may be more amenable to token-based approach than others

• SDT: There should be one mandatory baseline implementation
Naming

• DECADE architecture requirements:
  – Globally unique names
  – Application-independent
  – Name-content binding through hashes

• Proposing adoption of NI scheme
  – Key function: representing object hashes, with hash identifier
  – Support for different hash algorithms
  – Extensibility mechanism for application-specific URI parameters
  – Defined mapping from NI URIs to HTTP URIs

\[\text{ni://sha-256;B_K97zTtFuOhug27fke4}\_Zgc4Myz4b\_lZNgsQjy6fkc}\]

\[\text{ni://example.com/sha-256;B_K97zTtFuOhug27fke4}\_Zgc4Myz4b\_lZNgsQjy6fkc?ct=image/jpeg\]
How to use NI Names in DECADE

• Equality testing works on algorithm identifier and actual hash value
  – All other elements (including authority) are not considered
  – DECADE should not require an authority field

\[
\text{ni://example.com/sha-256;B_K97zTtFu0hug27fke4_Zgc4Myz4b_lZNgsQjy6fkc}
\]

\[
\text{http://example.com/.well-known/ni/sha-256/B_K97zTtFu0hug27fke4_Zgc4Myz4b_lZNgsQjy6fkc}
\]

• Mapping to HTTP
  – NI defines one specific mapping
  – Clearly only useful for HTTP-based SDTs
  – May impose some constraints on server configurations
Other NI Functions for DECADE

• Locator specification
  – Useful for referring client to a specific DECADE server
  – Implementable using an extension parameter

  `ni://sha-256;B_K97zTtFu0hug27fke4_Zgc4Myz4b_1ZNgsQjy6fkc?decade-loc=http://example.com/decade/NAME`

• Content type: already in NI params spec

• Authentication token

  `ni://sha-256;B_K97zTtFu0hug27fke4_Zgc4Myz4b_1ZNgsQjy6fkc?decade-auth=dhek4nd2kj2j`
Other NI Functions for DECADE

• Locator specification
  – Useful for referring client to a specific DECADE server
  – Implementable using an extension parameter

  \texttt{ni://sha-256;B\_K97zTtFuOhug27fke4\_Zgc4Myz4b\_lZNgsQjy6fkc}\n  \texttt{\?decade-loc=\textit{http://example.com/decade/NAME}}

• Content type: already in NI params spec

• Authentication token

  \texttt{ni://sha-256;B\_K97zTtFuOhug27fke4\_Zgc4Myz4b\_lZNgsQjy6fkc}\n  \texttt{\?decade-auth=dhek4nd2kj2j}
Authentication and Access Control

1. App request

2. Obtain Token (DRP)

---

3. App response (token)

4. Request and Download Object (DRP + SDT)

---

End-Point A

---

End-Point B
Authentication and Access Control

2. Obtain Token (DRP)

1. App request

End-Point A

3. App response (token)

S(A)

4. Request and Download Object (DRP + SDT)

End-Point B

• In general, two options for carrying authentication tokens
  – When referring a user to a DECADE server
  1. In the native application protocol
  2. In the object name
    – Seems preferable, since protocol-independent
Authentication and Access Control

2. Obtain Token (DRP)

1. App request

3. App response (token)

4. Request and Download Object (DRP + SDT)

- Downloading the object
  - SDT-instantiation-specific embedding of token in protocol fields
  - E.g., OAuth in HTTP
Application Contexts, Resource Collections

• Different servers, different file transfer protocols, and different remote file system protocols may provide different capabilities for organizing resources in hierarchical structures
  – Collections, file system directories etc.

• Question: should this be exposed in a DECADE SDT?
  – For instance: collecting all chunks of a larger object into one collection

• Our view: **NO**
  – It’s a server implementation thing – SDT does not want to know about
  – DECADE has unique naming feature
  – Can structure objects on application layer by listing them in an index file (think torrent files)

• This would imply that SDT does not need to support any operation on collections
  – Simpler implementations – better interoperability!
Server-to-Server

- DECADE architecture has concept of server-to-server communication
  - Servers to redistribute objects to other servers

- Would need an SDT mechanism
  - Would like to specify a set of target servers

- Caveat: HTTP-based servers do normally not support „DISTRIBUTE“ method
  - Would be nice to find a way around this
  - Would prefer not to lose interoperability with vanilla servers
CDMI as an SDT

• Goal: enable use of existing CDMI infrastructure in DECADE
  – Also: don’t raise the bar too high for minimal DECADE implementations

• CDMI in a nutshell
  – RESTful HTTP-based access to cloud storage
  – JSON as a representation format for describing resources, configurations – also for object (optionally)
  – Quite comprehensive, but with a profiling concept
CDMI Content Type Operations

• CDMI provides two alternative mechanisms for uploading/downloading objects:

1. CDMI Content Type Operations
   – Using JSON to encode objects (and meta data)
   – Might be difficult for non CDMI clients

2. Non-CDMI Content Type Operations
   – Objects in message bodies (vanilla HTTP-like)
   – More efficient and better for backwards-compatibility
CDMI Content Type Operations

- CDMI provides two alternative mechanisms for uploading/downloading objects:

1. CDMI Content Type Operations
   - Using JSON to encode objects (and meta data)
   - Might be difficult for non CDMI clients

2. Non-CDMI Content Type Operations
   - Objects in message bodies (vanilla HTTP-like)
   - More efficient and better for backwards-compatibility
Broad Range of CDMI Features

- discovering capabilities of a cloud storage provider;
- creating a new container;
- creating a new data object;
- listing the contents of a container;
- reading the contents of a data object;
- reading the value of a data object;
- deleting a data object.
- queue object resource operations, providing first-in, first-out access for storing and retrieving data;
- capability query operations, allowing a client to find out about the subset of CDMI features that a server supports;
- exporting (and configuring the exporting of) data objects to other protocol domains such as NFS, iSCSI, WebDAV etc.;
- serialization and de-serialization of data;
- configure access control through ACLs;
- retention and hold management;
- scope specifications to allow clients to select data objects based on filter/search expressions;
- results specifications (to enable a client to specify subsets of data objects to be returned);
- logging;
- notification queues (for example for notifying clients about changes to a file system or to certain objects); and
- query queues (enabling clients to requests data objects based on meta data or content search expressions).
Broad Range of CDMI Features

- discovering capabilities of a cloud storage provider;
- creating a new container;
- creating a new data object;
- listing the contents of a container;
- reading the contents of a data object;
- reading the value of a data object;
- deleting a data object;
- exporting (and configuring the exporting of) data objects to other protocol domains such as NFS, iSCSI, WebDAV etc.;
- serialization and de-serialization of data;
- configure access control through ACLs;
- retention and hold management;
- scope specifications to allow clients to select data objects based on filter/search expressions;
- SDT only needs a small subset of CDMI features;
- CDMI has modularity concept;
- DECADE should define a minimal profile that DECADE providers will support, including a subset of CDMI features that any CDMI-compliant storage system should support;
- capability query operations, allowing a client to find out about the subset of CDMI features that a server supports;
- exporting (and configuring the exporting of) data objects to other protocol domains such as NFS, iSCSI, WebDAV etc.;
- serialization and de-serialization of data;
- configure access control through ACLs;
- retention and hold management;
- scope specifications to allow clients to select data objects based on filter/search expressions;
- DECADE should define a minimal profile that DECADE providers will support, including a subset of CDMI features that any CDMI-compliant storage system should support;
- capability query operations, allowing a client to find out about the subset of CDMI features that a server supports;
- logging;
- notification queues (for example for notifying clients about changes to a file system or to certain objects); and
- query queues (enabling clients to requests data objects based on meta data or content search expressions).
CDMI Containers

• Quite a fundamental concept in CDMI
  – Comprehensive support for operations on containers
  – Required feature for cloud data management
  – Not so for DECADE

• Naming scheme (see earlier discussion) and DECADE SDT should be oblivious to structure, hierarchy etc.
  – Can be done on the application layer
  – CDMI-SDT would use CDMI (largely) without using containers
CDMI Object Identifiers (1)

- Fundamentally compatible to DECADE naming ideas so far (globally unique, potentially leveraging content hashes)
- Specific format not directly compatible to NI format
  - There may be ways to map names

http://decade.example.com/root/cdmi_objectid/647284746393

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | .. | 38 | 39 |
|---|---|---|---|---|---|---|---|---|---|----|---|    |    |
| Reserved | Enterprise | Reserved | Length | CRC | opaque data |
| (zero) | Number | (zero) |   |   |     |
Creating object identifiers in CDMI

- Done by the server
- In DECADE, it would be better (more efficient, better workflow) if the client did it
- Have to find out about the options
Security

• Need to work on access control, token-based authentication
• DoS attack vectors: server-to-server communication can be a risk
• Name-content integrity: need to specify the details (hash algorithms, requirements for servers and clients)
  – DECADE NI profile could perhaps do that
Conclusions

• NI URIs in DECADE
  – Want to specify the DECADE NI profile
  – With extensions for locators

• General SDT guideline: KISS
  – Keep application layer features to application (re: collections)
  – Try not to break interoperability with existing gear

• CDMI
  – Goal: do not exclude leveraging CDMI by design – ideally requiring only minimal changes
  – SDT with CDMI can probably be done – have to do it carefully
  – Question is whether this should be the baseline SDT spec
  – Proposed way forward: enable SDT implementation leveraging CDMI implementations