A CAS Digital Report

dbprotocols.spec

A Universal CAS Digital Specification on Data Operations

Lesang Dikgole

15th November, 2017

Introduction

This Specification is written with the following goals in mind

- ★ develop a *simple language* for data operations
    - ○ (e.g. adding, retrieving, updating, and removing data objects)
- ★ data operations must
    - ○ have exact database effects specified in the operation
    - ○ allow for dynamic/problem-specific queries
    - ○ be database (and related location) agnostic
- ★ support for *descriptive data diagnostics* and *AI*
    - ○ i.e. the data must be queried in a 'meaningful way' or must relate to the context for which it will be used. This will be incredibly useful for 'intelligence' systems design. A robot needs to use 'insight' seamlessly without strict dependence on data operations syntax.
    - ○ All 'intelligence' insight is dependent on context, location, actions, data sources, or the risk model, all of which will be extremely dynamic and data intensive. This spec is an attempt to simplify this future challenge.

Current dbprotocols.spec Data Operations

DataOpA: setRecord | DataOpB: getRecord

DataOpC: infoStore | DataOpD: infoAssert

DataOpE: infoRetract | DataOpF: infoAsk

DataOpG: infoSearch | DataOpH: infoRetrieve

DataOpI: infoDelete | DataOpJ: setTree

DataOpK: getTree | DataOpL: setAnswerPage

DataOpM: getAnswerPage | DataOpO: *security*

Local vs Remote

All these operations will have both 'local' and 'remote' side effects. As explained in the following sections.

Local

These operations will be implemented as 'functions' or 'methods'

- the 'function' names shall comply with the above nomenclature
- the 'function' parameters shall include
  - database object (can be a file location 'object', sql database 'object', etc)
  - database type ('file', 'sql_db', 'graph_db', 'document_db', etc)
  - database name ('deadsql', 'production_db', 'locoA_db', etc)
  - database table (this does NOT just mean sql table, it could mean
    - a sql table!
    - a graph factory = dataTree
    - a document library
    - a group of 'insights' = a document/'book')
  - requesttype (function name (REPEATED), e.g. 'getRecord', etc)
  - requestdata (this is where all the 'specificity' is applied
    - for a sql table - which fields/columns?
    - for a graph factory - which graph, or tree/nodes/vertices?
    - for a document library - which document(s)?
    - for a book - which page(s)?)

All these functions and parameters can theoretically be expanded. But the design was made such that, that would not be necessary.

If a need is identified, it is suggested that the user of the spec examines how the specific needs can be addressed in the current spec instead of ADDING a new operation or function. The addition of new functions is thus discouraged, but not entirely forbidden.

Remote

Remote operations will be 'parsed' using the following parameters:

★ database type
★ database name
★ database table
★ requesttype
★ requestdata

Example, for a JSON file

```
{app: "db_app",
        database_type: "sql”,
        database_name: "deadsql",
        database_table: "testing",
        requesttype: "getRecord",
        requestdata: "all",
        params: "enforce_timeout}
```

Appendix A

Detailed Remote Implementation Over Http (React.js).

- ★ +**npm** install axios
- ★ +**import** the package
- ★ +**import** axios from 'axios';
- ★ +set the URL
  - ○ **const** BASE_URL = 'http://46.101.85.64:8888';
- ★ +set the ROUTE
  - ○ **const** myurl = `${BASE_URL}/sqldb`;
- ★ +set the content-type
  - ○ axios.defaults.headers.post['Content-Type'] = '**application/json**';
- ★ +run the query!

NB: ONLY use POST, set the response-data-type to 'json'

```
axios({

method: 'post',

url: myurl,

responseType: 'json',

data: {

app: "db_app",

database_type: "sql”,

database_name: "deadsql",

database_table: "testing",

requesttype: "getRecord",

requestdata: "all",

params: "enforce_timeout"

} })
```

Appendix B

Detailed dbprotocols.spec Operations Explanation

★ **setRecord**
  - This is to UPDATE or CREATE new records
  - A record can be: a table; a graph; a library; or a document/book.

★ **getRecord**
  - This is to RETRIEVE an existing record(s)
  - A record can be: a table; a graph; a library; or a document/book.

★ **infoStore**
  - This is to UPDATE of CREATE new insights
  - An insight can be: a group of tables, a graphfactory, or a library, or a table or a graph.
  - An insight can be completely generated by the AI system meaning that it may not have a relation with the with the data consumed or stored from 'traditional' data sources.

★ **infoRetrieve**
  - This is to RETRIEVE insights
  - An insight can be: a group of tables, a graphfactory, or a library, or a table or a graph.

★ **infoAssert**
  - This is to CONFIRM existing insights
  - An insight can be: a group of tables, a graphfactory, or a library, or a table or a graph.
  - This operation, is used to create NEW INFO. Which ADDs a layer of certainty (=assurance) to an existing insight)

★ **infoRetract**
  - This is to RETRACT existing insights
  - An insight can be: a group of tables, a graphfactory, or a library, or a table or a graph.
  - This operation, is used to create NEW INFO. Which ADDs a layer of uncertainty to (= disproves) an existing insight.
  - This can result in reclassification, re-graphing, or 're-painting' of existing insights.

★ **infoAsk**
  - This is to deterministically RETRIEVE insights.

    - These insights will usually be accessible through a standard database operation, but extremely complex to RETRIEVE.
    - These insights are deterministic, meaning that the result will either be a TRUE result, or 'not-known or not-available'.
    - The implementation of this operation needs to take care of the details of searching, and looking for the exact records required; failing which a 'not-available' response shall be provided.
- ★ **infoSearch**
    - This is to nondeterministically RETRIEVE a library of documents.
    - This library will usually will NOT be accessible through a standard database operation.
    - This library is nondeterministic and unbound meaning that the result will return a varying number of documents; the number depends on the amount of 'related' documents in the database.
    - The implementation of this operation needs take care of the details of searching for the library of documents. Extensive use of NLP for the implementation of this method is encouraged.
- ★ **infoDelete**
    - This deletes existing insights.
    - (This is primarily an internal 'system' operation; but can also be performed by the user on a few narrow cases; when required under extreme circumstances).
- ★ **setTree**
    - This is to UPDATE or CREATE a new tree(s).
    - A tree can be: a graph; a group of books; or a graph factory.
- ★ **getTree**
    - This is to RETRIEVE an existing tree.
    - A tree can be: a graph; a group of books; or a graph factory
- ★ **setAnswerPage**
    - This is to UPDATE or CREATE a new page(s).
    - A page can be: a table; a group of keys (in a KVP database); or document/book page.
- ★ **getAnswerPage**
    - This is to RETRIEVE an existing page(s).
    - A page can be: a table; a limited group of keys (in a KVP database); or a document/book page.