GridRPC Data Management API Implementations and Interoperability Issues

Y. Caniou, E. Caron, F. Desprez, G. Le Mahec, H. Nakada

16 mar. 2010

Data Management in the GridRPC

Aims of the Data Management API

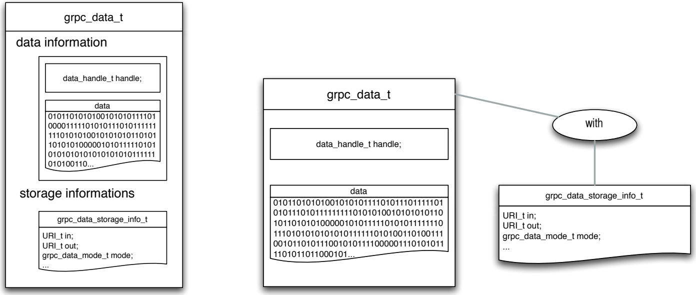
- To avoid useless transfers of data
- Generic API unrelated to the data, its location, access protocol, etc.
  $\rightarrow$ **Transparent access to the data from the user point of view**
- Homogeneous use of different data transfer protocols
- To improve **interoperability between different implementations**
- The API should be compliant with SAGA API requirements

Constraints

- Must be an optional improvement of GridRPC applications
- Must be in accordance with the GridRPC API
- Should be extensible to existent and future data transfer protocols

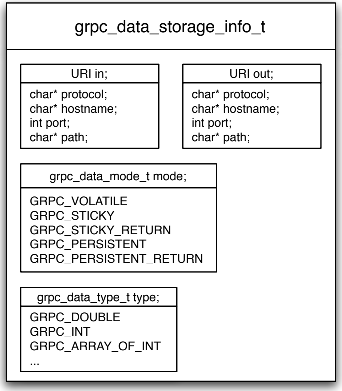
GridRPC Data Types (1/2)

The *grpc_data_t* type contains the data or a handle on it.
The *grpc_data_storage_info_t* of a data can be in the *grpc_data_t* structure or transmitted separately by a Data Middleware.

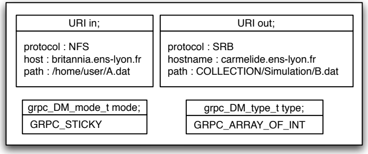


GridRPC Data Types (2/2)

The *grpc_data_storage_info_t* type contains the URI where the data can be accessed, the URI where the data will be sent after the call, the management mode and the type of the data.



Data example



Types in *grpc_data_t* and access functions

Labels for the *grpc_data_type_t*

```
GRPC_INT
GRPC_DOUBLE
GRPC_COMPLEX
GRPC_CONTAINER_OF_GRPC_DATA
```

Access Functions to Elements in a Container of *grpc_data_t*

```
grpc_error_t grpc_data_container_set( grpc_data_t * container, int rank,
                                      grpc_data_t * data );

grpc_error_t grpc_data_container_get( grpc_data_t * container, int rank,
                                      grpc_data_t * data );
```

- **container** is necessarily a grpc_data_t of type `GRPC_CONTAINER_OF_GRPC_DATA`
- **rank** is a given integer which acts as a key index
- **data** is the data that the user wants to add in or get from the container

→ Getting the data does not remove the data from **container**

→ **Container management is free of implementation**

Implementation and/or Interoperability Issues?

One must consider. . .

- portable access to information
- portable save/restore state/information
- possibility to transfer a data managed by an external DM

  → In our case, it can even be an other GridRPC DM implementation!

Types Issues / Solutions?

Portable access to information

— Is name of fields portable?

→ Do we use accessors to information field?

+ We can define and extand later standardized fields if needed

+ Towards a real portable code?

Some data are defined as sets

— `list_of_URI_input`

— data of `GRPC_CONTAINER_OF_GRPC_DATA` type

* ...implemented as array or list (see `implementation_examples-0.6.pdf`)

→ accessors to #item in a set (*)

All these issues concern implementation, not interoperability!

Information Issues / Solutions?

The grpc_data_getinfo() Function

```
grpc_error_t grpc_data_getinfo(const grpc_data_t * data,
                               grpc_data_info_type_t info_tag,
                               const char * URI,
                               char ** info);
```

The kind of information that the function gets is defined by the **info_tag** parameter. [...] **info** is a `NULL`-terminated list containing the different available information corresponding to the request.

- Define a normalized set of information for `info`? Syntax?
- How do we manage evolutions?

All these issues concern implementation and interoperability!

Functions / Types Issues / Solutions?

The **grpc_data_load()** and **grpc_data_save()** Functions

```
grpc_error_t grpc_data_load(const grpc_data_t * data,
                            const char * URI_input);
grpc_error_t grpc_data_save(const grpc_data_t * data,
                            const char * URI_output);
```

These functions are used to load/save the data descriptions. [...]
The format used by these functions is let to the developer's choice.
The way the information are shared by different middleware is
out of scope of this document and should be
discussed in an interoperability recommendation document.

$\longrightarrow$ In that case, we have to define an [evolutive] standard

Concerns both implementation and interoperability

(interoperability because should work independently of architectures)

Other Issues?

How to know the DM that manages the data? Is it relevant?

1. Implementation Issues
   - → It seems, as long as information is not standardized for example.
   - — Is it relevant to ask a (client/agent/server) code to handle different cases?
   - → No! Again, it seems that we **should** propose unified information
2. Interoperability Issues
   - → It can ease glue development, by calling the accessor of the correct DM if available, and return an error if not (we can not access the value).

- Are stored information always up-to-date?
- Is it relevant to define a function for this?