GWD-R, GWD-I or GWD-C
Authors:

Sergio Andreozzi* (editor), INFN
Stephen Burke, RAL
Felix Ehm, CERN
Laurence Field*, CERN
Gerson Galang, ARCS
David Horat, CERN
Balazs Konya*, Lund University
Maarten Litmaath, CERN
Paul Millar, DESY
JP Navarro, ANL
*co-chairs

GLUE WG
http://forge.ogf.org/sf/sfmain/do/viewProject/projects.glue-wg

July 4, 2011

GLUE v. 2.0.1 – Reference Realizations to SQL Schema

Status of This Document

This document provides information to the Grid community regarding the Relational Schema in SQL realization of the GLUE information model (v.2.0). Distribution is unlimited. This implementation is derived from the specification document "GLUE Specification v. 2.0.1", April 23, 2009. This document is a draft.

Abstract
The GLUE specification is an information model for Grid entities described in natural language enriched with a graphical representation using UML Class Diagrams. This document presents a realization of this information model as an SQL Schema.

Contents

1. Introduction

The GLUE 2.0 Information model defined in [glue-2] is a conceptual model of Grid entities. In order to be adopted by Grid middlewares, a realization in terms of a concrete data model is needed.

This document provides the normative realization of the GLUE 2.0 conceptual model in terms of a Relational schema in SQL. The approach followed to map the entities and relationships in the conceptual model to the concrete relational data model is also described.

2. Notational Conventions

The key words ‘MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” are to be interpreted as described in RFC 2119 (see http://www.ietf.org/rfc/rfc2119.txt).

3. SQL Realization

3.1 Approach

Not all modeling concepts applied in the GLUE conceptual model are supported by the relational database management systems (RDBMSs). The unavailable concepts have to be described using the present mechanisms that do not necessarily establish a one-to-one mapping from the model point of view. For instance, inheritance is one of the concepts that have to be translated, what might result in trade-offs based on the specific application scenario. As such, decisions affecting extensibility, scalability and performance of the relational GLUE schema in real-life situations have to be taken during the implementation process.

The proposed SQL rendering is based on the ANSI SQL standard supported by the majority of the available RDBMSs. The rendering balances flexibility and performance and is therefore applicable in a large variety of situations. Our rendering approach is based on the prior experience with SQL and the relational database models and is supported by knowledge and recommendations developed by the SQL community over the past decades.

3.2 Data Types

Attributes defined in the GLUE model can assume common or enumeration data types. The common data types, such as String, UInt32 or Boolean, define the size and format of values that can be assigned to the respective attributes. For instance, the UInt32 type represents unsigned integers with values ranging from 0 to about 4 billion. The common data types can be directly mapped to SQL. The following SQL data types are used in the GLUE SQL rendering:

- INT corresponds to UInt32.
- BIGINT corresponds to UInt64. Although the BIGINT data type is not defined in the ANSI SQL standard it is supported by the majority of modern RDBMSs.
- DOUBLE corresponds to Real32.
- DATETIME corresponds to DateTime_t.
- VARCHAR(1024) corresponds to String and allows up to 1024 characters.

Other common types, for example Boolean, are not defined in SQL and are represented as strings.

The enumeration data types describe a set of allowed values that can be assigned to GLUE attributes. These types are defined in the GLUE model and are assigned to one of the two enumeration classes: open and closed. Open enumeration types define a list of values, where one of the values may be chosen. If no appropriate value is defined any other string in

compliance with a specified syntax can be assigned to the attribute. Closed enumeration types define a list values one of which must be chosen.

The enumeration types can be defined in SQL using a custom data type. Unfortunately, the custom type is not defined in the ANSI SQL standard. Although being supported by many RDBMSs, custom types are not portable across different systems. Therefore the enumeration types are not defined in the GLUE SQL rendering. However, if necessary, they can be easily implemented for a specific RDBMS.

3.3 Multi-valued Attributes

The GLUE conceptual model defines the following cases for attribute multiplicity, i.e. the number of values that can be assigned to a single attribute: zero-to-one, one, zero-to-many and one-to-many. The latter two multiplicities define multi-valued attributes able to store more than one value. For example, the GLUE computing endpoint can expose one or more capabilities according to the OGSA architecture. RDBMSs only support single-valued attributes, multi-valued attributes are not supported and have to be treated in a special way.

The first option is to create a distinct table for each of the multivalued attributes. Each value of the attribute stored in the table is associated with the corresponding record in the master table using a primary key. For example, assume a computing endpoint called “SGE” exposes two capabilities: job manager and job execution. To store this in a relational database we need two tables: one describing the computing endpoint, e.g. the endpoint’s ID, name, interface, etc., and the other listing all capabilities each computing endpoint exposes. Each capability described in the latter table will be associated with the respective endpoint using a unique identifier such as the endpoint ID.

The alternative is to create an additional table for each GLUE entity that contains multivalued attributes. The multivalued attribute (MVA) table will contain name and value of attributes and associate each record to the respective GLUE entity instance using an ID. Let us take the ComputingEndpoint class as an example. To accommodate all multivalued attributes defined in the class a table called ComputingEndpoint_MVA will be created. The table should define three attributes: ID, name and value. The name and value fields, as can be seen from the names, are used to store the name and value of a particular multivalue attribute instance. The ID field establishes a relationship between the attribute instance and the respective entry in the ComputingEndpoint table.

The second approach was chosen for the GLUE SQL rendering. Compared to the first approach where a table for each multivalued attribute is created MVA tables are much easier to define and manage. They do not have a large impact on the total number of tables in the database and offer an efficient mechanism for storage of multivalued attributes.

3.4 Inheritance

The GLUE conceptual model relies on inheritance to define relationships between defined classes. All concrete classes defined in the model inherit the abstract Entity class. Many storage and computing service classes are further specializations of main entities such as Service, Endpoint, Share and Manager. Inheritance is not natively supported by relational databases and has to be described with help of other mechanisms.

Three general inheritance-mapping techniques can be used for describing inheritance relationship between entities in a relational database:

- One table per hierarchy
- One table per concrete class
- One table per class

In the first case a schema is implemented as one table. This approach has advantages for smaller hierarchies. Yet in the case of the GLUE schema it is by far not the optimal one. It would mean that hundreds of GLUE attributes are to be stored in a single table that will be hard to manage and operate on. This approach has further drawbacks, for instance high risk of introducing duplicate or inconsistent data records. As such, it is not applicable for the GLUE model.

The second option is to describe each concrete class using one table. For example, the ComputeService class is a specialization of the Service class that, in turn, extends the Entity class. The attributes describing a compute service, including the inherited attributes, will be defined in a common table. This approach is easy to implement since each concrete class is defined in an individual table. It is easy to add new classes by adding additional tables. Data manipulation and access is fast since the data is stored in one table. The approach has disadvantages if classes have to be modified since a change can affect multiple tables. The database structure quickly grows for schemas with complex hierarchy such as GLUE. This results in duplication of attributes and requires extra effort for ensuring data consistency.

In the last approach one table per class is created. As such the ComputeService class, for instance, will be described using three tables: an Entity table, a Service table and a ComputeService table, where each table contains attributes defined for the class and references the parent table for the rest of the attributes. This approach is easy to understand and implement because of the one-to-one mapping between classes of the GLUE schema and tables of the relational SQL rendering. It is easy to maintain as changes in the schema affect only the corresponding table. This implementation offers a balance between storage, complexity and performance characteristics and was therefore chosen for the GLUE SQL rendering.

3.5 Relationships

Three kinds of relationships between objects are used in the GLUE schema: one-to-one, one-to-many and many-to-many. All three are supported by SQL and can be implemented in any RDMS. One-to-one and one-to-many relationships are implemented by referencing the primary key of one table in another one. For instance, a reference between the ComputingEndpoint and ComputingService classes is established by defining a ComputingServiceID attribute in the ComputingEndpoint table.

Many-to-many relationships have to be described in external tables. In our implementation the MVA tables are used for this purpose.

It should be noted that table dependencies are not included in the schema definition to allow additional flexibility. As such, data consistency has to be validated on the software level.

3.6 Value Multiplicity

The GLUE schema defines four attribute multiplicity types:

- optional attributes have multiplicity of zero-to-one or zero-to-many;
- mandatory attributes have multiplicity of one or one-to-many.

Optional and mandatory attributes with multiplicity of one are defined in the SQL rendering and the respective multiplicity is enforced. Mandatory attributes with multiplicity of one-to-many should be stored in the respective MVA tables. Due to the structure of the tables it is not possible to enforce the one-to-many multiplicity on the database level. If necessary, the required data validation mechanisms have to be implemented on the software level.

4. Security Considerations

Using SQL to implement GLUE 2.0 Specifications raises several considerations especially in the field of data integrity.

SQL cannot guarantee the correct mapping of the data types employed in the GLUE 2.0 Specification, thus these must be ensured by other means (also see 3.2).

5. Author Information

Sergio Andreozzi, INFN

Stephen Burke, RAL

Felix Ehm, CERN

Laurence Field, CERN

Gerson Galang, ARCS

David Horat, CERN

Balazs Konya, Lund University

Maarten Litmaath, CERN

Paul Millar, DESY

JP Navarro, ANL

6. Contributors & Acknowledgements

We gratefully acknowledge the contributions made to this document.

7. Intellectual Property Statement

8. Disclaimer

9. Full Copyright Notice

10. References

[glue-wg] The GLUE Working Group of OGF, *https://forge.gridforum.org/sf/projects/glue-wg*
[glue-usecases] GLUE 2.0 Use Cases (early draft), *https://forge.gridforum.org/sf/go/doc14621*
[glue-2] GLUE 2.0 Specification, *https://forge.gridforum.org/sf/go/doc146239*
[ogf-ns] Standardised Namespaces for XML infosets in OGF,
http://www.ogf.org/documents/GFD.84.pdf
[xsd-oe] XForms 1.0. Open Enumeration.
http://www.w3.org/TR/2002/WD-xforms-20020118/slice6.html#model-using-openenum
[xsd-ap] Advanced XML Schema Patterns for Databinding Version 1.0,
http://www.w3.org/TR/xmlschema-patterns-advanced/#group-Unions
[sql-92] ISO/IEC 9075:1992 - Database Language SQL (SQL-92),
http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt