



## NextGRID Data Storage and Access Use Cases and Requirements

<b>Editors:</b>	Clive Davenhall	UEDIN

Date	Author	Comments	Version	Status
13 Aug. 2007	Clive Davenhall	Placeholder	0.0	Pre-Draft
31 Aug. 2007	Clive Davenhall	Initial Version	0.1	Draft
7 Oct. 2007	Clive Davenhall	Modified following review	0.2	Draft
7 Dec. 2007	Clive Davenhall	Combined the Storage and Access use case documents	0.3	Draft
19 Feb. 2008	Clive Davenhall	Modified following review	0.4	Draft
1 Mar. 2008	Clive Davenhall	Final edits.	0.5	Draft
11 Mar 2008	D. Snelling	Final QA	1.0	Final



**1 INTRODUCTION..... 3**

1.1 Overview of the Data Storage and Data Access Profiles ..... 3

1.2 Relevant Specifications..... 3

1.3 Terminology ..... 4

1.4 What is a Dataset? ..... 4

1.5 Access and Storage ..... 5

**2 GENERAL REQUIREMENTS ..... 6**

2.1 Data Storage Service Metadata ..... 6

2.2 Access by Name or Access by EPR?..... 6

2.3 SLA to Publish a Dataset..... 7

**3 SIMPLE USE CASES TO ACCESS A FILE ..... 8**

3.1 Direct Access from a Data Store..... 8

3.2 Indirect Access via an External Registry ..... 10

**4 SIMPLE USE CASE TO QUERY A DATABASE..... 12**

4.1 Basic Query ..... 12

4.2 Variant 1: Streamed Data ..... 13

4.3 Variant 2: Third Party Transfer ..... 13

**5 COMPLEX USE CASES ..... 14**

5.1 Personal Data Service..... 14

    5.1.1 Description..... 14

    5.1.2 Implementation ..... 15

5.2 Data Replication Scenario..... 16

    5.2.1 Description..... 16

    5.2.2 Implementation ..... 16

**REFERENCES ..... 17**



## 1 Introduction

This document presents the NextGRID data storage and data access use cases and requirements that accompany the *NextGRID Data Storage Profile 1.0* [1] and the *NextGRID Data Access Profile 1.0* [2].

The NextGRID Generalized Specifications aim at capturing NextGRID architectural concepts in a set of profiles that may be composed together. These profiles are specified in such a way that they could be implemented in terms of other well known specifications. While overall consistency is achieved at the conceptual level, and captured through the motivating Use Cases accompanying each Generalized Specification, system implementations, which may be based on other specifications, may not be consistent with this Profile. Thus, each profile defines a realisation of the underlying concept that can be implemented. However, implementers of the NextGRID architecture may need to support multiple underlying specifications.

### 1.1 Overview of the Data Storage and Data Access Profiles

The *NextGRID Data Storage Profile* and the *NextGRID Data Access Profile* are intended for use when implementing, respectively, data storage and data access services that are in line with the concepts of NextGRID [3]. They mandate adherence to a certain set of specifications and clarify their use.

- A service implementation that uses these specifications in a manner conformant with the *NextGRID Data Storage Profile* may be said to be an “implementation of the *NextGRID Data Storage Profile 1.0*” or, informally, to be a “NextGRID Data Storage Service”.
- A service implementation that uses these specifications in a manner conformant with the *NextGRID Data Access Profile* may be said to be an “implementation of the *NextGRID Data Access Profile 1.0*” or, informally, to be a “NextGRID Data Access Service”.

In NextGRID data access and data storage are closely related.

### 1.2 Relevant Specifications

#### NextGRID Storage Profile

Data stores or data storage services conform to the *NextGRID Data Storage Profile* [1].

#### NextGRID Data Access Profile

Access to datasets in data stores conforms to the *NextGRID Data Access Profile* [2].



## NextGRID Naming Profile

Service naming conforms to the *NextGRID Naming Profile* [4].

## NextGRID Data Transfer Profile

To access datasets from a store, the store needs to be able to act as either a sink or a source of data within the context of the *NextGRID Data Transfer Profile* [5].

## 1.3 Terminology

This section introduces the terminology used in the remainder of this document. Hereafter terms defined in this section are shown in **bold** in this document. See also Section 1.4, below, and the OGSA *Glossary of Terms* [6].

**Dataset:** any identifiable collection of data. See Section 1.4, below.

**File:** a **dataset** instantiated as a file in some operating system.

**Selection:** a **dataset** instantiated as a set of tuples in some database which satisfy some criterion.

**Data storage service:** A NextGRID service implementation conformant with the *NextGRID Data Storage Profile* [1]. A **data storage service** is a store or repository for **datasets**.

**Data access service:** A NextGRID service implementation conformant with the *NextGRID Data Access Profile* [2]. A **data access service** allows **datasets** to be accessed.

**Data store:** a synonym for a **data storage service**.

**Store:** a synonym for a **data storage service**.

**Client:** a NextGRID entity which wishes to invoke either a **data storage service** or a **data access service**.

**Access:** the act of either (i) obtaining a copy of whole or part of a **dataset**, (ii) modifying a **dataset in situ** or (iii) deleting a **dataset**. **Access** is prescribed by the *NextGRID Data Access Profile* [2].

**End Point Reference (EPR):** a WS-Addressing construct that identifies a message destination. An EPR conveys the information needed to identify or reference a resource. The use of EPRs and WS-Addressing in NextGRID is prescribed in the *NextGRID Naming Profile* [4].

## 1.4 What is a Dataset?

In the context of NextGRID, a **dataset** is any identifiable collection of data; that is, one which can be uniquely and independently identified by a NextGRID name (see the *NextGRID Naming Profile*

[4]) or referenced with an EPR. The WS-DAI Core Specification [7] has a complementary definition of a **dataset** as ‘an encoding of data suitable for externalization outside a data access service, for example as an XML document.’

NextGRID places no inherent constraints on the internal format or structure of the **dataset** or on its organisation in any storage medium. Individual instances of NextGRID processes and services can, of course, place such restrictions on the **datasets** that they manipulate.

For example, a **dataset** can be (i) a single **file** (in some operating system), (ii) a database, (iii) a table in a database, or (iv) a selection of tuples from one or more tables in a database (that is, a view). In the last three cases the **dataset** is a **selection**.

The defining characteristic of a **dataset** is that it is an identifiable collection of data. Though **files** and **selections** in a database both constitute **datasets**, the realisations are different. In the case of **files in store**, each **file** is already a discrete, named **dataset**. In the case of a **selection** generated by some query, the **dataset** is identifiable (by the query) but only becomes a distinct, named entity after the query has generated its **selection**. The concept of **selections** from databases emphasises the importance of **stores** supporting update operations as well as input and retrieve operations.

## 1.5 Access and Storage

The use cases presented in this document are concerned with data storage and data access. In NextGRID, data storage and data access are closely related. Both are concerned with performing operations on **datasets**, irrespective of whether the **datasets** are **files** or **selections**.

**Data storage** is the persistence of data through time. A **dataset** may be imported into a **store**. In the absence of direction from any external entity the **store** will then preserve it unchanged for a specified duration. That is, storage is concerned with preserving and superintending **datasets**. While a **dataset** is in a **store** it may be **accessed**. **Access** includes: obtaining copies of whole or part of a **dataset**, modifying a **dataset in situ** or deleting a **dataset**. In a distributed system with components operated by multiple vendors, such as in NextGRID, a safe deposit system [8] is a reasonable analogy.

While providing **access** to the **datasets** it is storing is necessarily part of the functionality of a **data storage service**, other types of NextGRID service may also provide **access** to transient **datasets**. For example, data processing services [9] may provide **access** to **datasets** that they have computed or services superintending data-feeds or sensors may provide **access** to real-time measurements.

**Stores** will typically store metadata about the **datasets** that they contain. If the **store** contains **files** the metadata for each **file** will typically include its: name, size, creation date, access date and any access restrictions. If the **store** is storing data in a database, it will typically retain metadata about the database which includes its: name, the format of queries supported and any access restrictions.

## 2 General Requirements

Data storage is a service. The basic operations required of a **data storage service** are that it can import, export, update and delete **datasets** and that it can be queried to yield details of the **datasets** that it holds. All these operations involve **access** to the **dataset**.

This section present the general requirements for a **store**.

### 2.1 Data Storage Service Metadata

In order for a **client** process to access a **dataset** and/or its associated metadata from a **store** the **client** must determine the capabilities of the **store**, to check that interaction is possible and to configure itself as appropriate.

For example, queries are made using query languages, results are returned in particular formats or representations and results are transported back to the **client** using transport protocols.

A **store** will usually also impose security restrictions (prescribing which users are allowed to retrieve or modify a **dataset**) and will typically have quality-of-service characteristics (specifying its capacity for processing queries or returning results).

A **store** will have metadata describing its capabilities and policies. A **client** uses this metadata to decide whether the **store** is suitable for its purposes (that is, whether they can ‘do business’ together) and to configure itself appropriately. For example, a **client** might check that the query languages supported by the **store** include at least one that it can use, and having established that there is at least one match, configure itself to use one of the matches.

The metadata for a **data store** should describe its capabilities and properties in the following areas.

1. Query languages supported.
2. Result representations (or formats) supported.
3. Transport (or transfer) protocols supported.
4. Description of security policies.
5. Quality of service policies.
6. Availability of replicas or mirrors.

### 2.2 Access by Name or Access by EPR?

All **datasets** imported into or exported from a **store** are ultimately identified by an EPR. Strictly speaking, this EPR is the only information needed to reference the **dataset**.

However, it is very desirable that **datasets** should also be identifiable by name. The NextGRID name [4] is available for this purpose. There are several reasons why identification by name is desirable. Consider a **store** containing **files**:

1. The availability of a name facilitates the implementation of higher-level functionality which imposes a syntax and semantics on components of the name. Such features are common in **data stores** which might, for example, impose a hierarchical structure on name components, analogous to directories in a file system. For example, the particle physicists' SRM [10] works in this way, as do some versions of AstroGrid's MySpace [11], [12]. While NextGRID does not require, impose or prescribe such functionality, it should facilitate it.
2. The EPR of a **file** in a store might not remain constant. Consider the case of an off-line **store**. An EPR will be assigned when a **file** is imported into the store. However, once input is complete, and the **file** has been ingested into off-line storage, the EPR may be relinquished. When the **file** is retrieved and restored for on-line access, a new EPR may be assigned. This EPR will not necessarily be the same as the original one used to input the **file**. The availability of a name which remains invariant and independent of the EPR simplifies access to such **files**, bearing in mind that the name will need to be mapped to the right EPR.
3. Similar arguments apply if the **store** itself moves, or **files** are to be replicated or backup copies kept.

Thus, though it is not, strictly speaking, necessary that **stores** support the identification of **files** by NextGRID naming as well as by EPR, it is strongly recommended that they do.

Ultimately any name by which a **store** knows a **file** will be assigned by the **store**. However, a **client** will usually be able to suggest a name for a **file** that it is importing. (Note that the **client** must suggest not prescribe because its chosen name may be invalid.) See the 'Getting a unique name and storing it' use case in Section 2 of the *NextGRID Naming and Addressing Use Cases and Requirements* [13].

### 2.3 SLA to Publish a Dataset

When a **client** imports a **dataset** into a **data store**, the operation will be mediated by a Service Level Agreement (SLA), as all NextGRID interactions are. Typically the **store** will offer metadata describing its capabilities and policies and the **client** will use this information to decide whether or not to use the **store**. All the SLAs used by NextGRID **data stores** should conform to the *NextGRID SLA Schema* [14].

While NextGRID should not unduly prescribe or constrain the terms of this SLA, it is likely to cover areas such as the following:

- The length of time that the **dataset** will be held in the **store**. Typically the **store** will undertake to preserve the **dataset** for a given period of time. The disposition of the **dataset** after the expiry of this period may also be specified. Possible dispositions might include:

deleting the dataset, retiring it to archival off-line storage or returning it to the client who originally imported it.

- The access time taken to retrieve a copy of the **dataset** from the **store**. For example, real-time access to on-line storage within a few seconds might be more expensive (in terms of either ‘real money’ or some accounting units) than access to off-line or archival storage, when the retrieval time might be a few hours (or days).
- The agreement could also cover who is allowed to (i) retrieve a copy of the **dataset**, (ii) retrieve metadata (or summary details) describing the **dataset** as a result of a query submitted to the **store**, (iii) update the **dataset** and (iv) delete the **dataset**. Note that cases (i) and (ii) are quite distinct. The owner of some **datasets** in a **store** might want them to be publicly visible as a result of queries, but not to be retrievable. The owner could then charge third-parties for access rights; for example, think of a repository of e-books. In general, the owner can prescribe, proscribe and delegate (and delegate the privilege to delegate) access rights to his **datasets**.

### 3 Simple Use Cases to Access a File

This section presents the simple use cases to **access** a **file** held in a **store**. Two sets of use cases are considered:

- direct access from a **data store**, and
- indirect access via a registry.

#### 3.1 Direct Access from a Data Store

Here a **file** resides in a store. The user, or rather a **client** acting on his behalf, either knows an identifier for a **file** (a name or an EPR conforming to the *NextGRID Naming Profile* [4]) or the name of the **store** where it is held. There is an analogous case for **publishing** a **file**; that is, making it available for access. The basic use cases are:

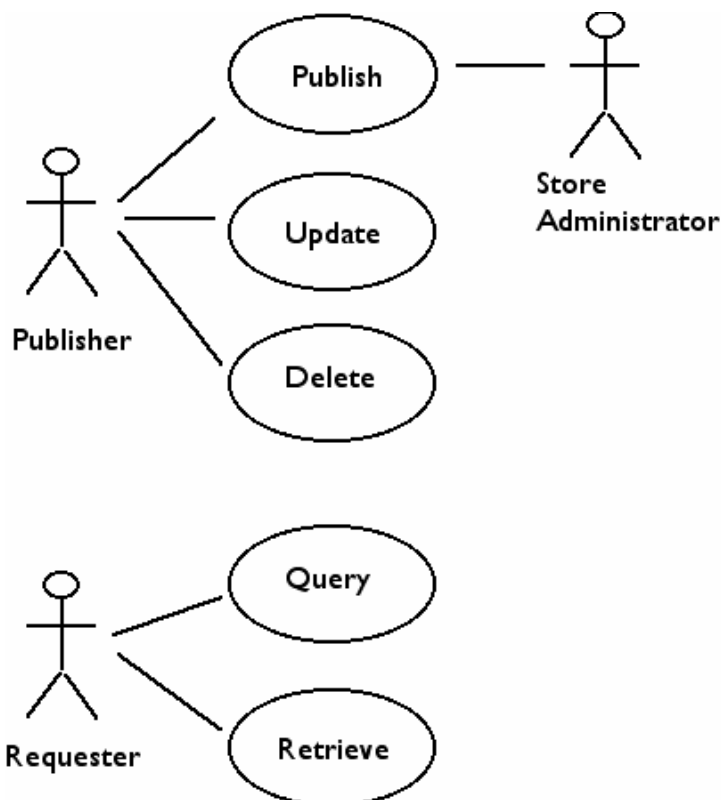
1. **Publish** a **file** into a **store**.
2. **Update** a **file** held in a **store**.
3. **Delete** a **file** from a **store**.
4. **Query** a **store** to identify **files** which match a given condition.
5. **Retrieve** a copy of a **file** from a **store**.

In practice the **query** operation works on metadata, while the **retrieve** operation will return both the **file** and its associated metadata. The **update** operation involves modifying an existing **file**. The use case diagram for these use cases is shown in Figure 1.

The three actors who appear in this diagram are:

- **Store Administrator:** the actor who configures the **store** by setting the policies prescribing the conditions under which it will offer to store **files**. These policies will typically cover areas such as access permissions, longevity of **files**, time to access *etc.*
- **Publisher:** the actor who inputs (or ‘publishes’) the **file** into the **store**, from whence it may be accessed. The publisher also updates or deletes the **file**. Finally, the publisher may delegate (or delegate the privilege to delegate) his role.
- **Requester:** queries or retrieves a **file** from a **store**.

The outline steps involved in implementing the various use cases are given below. All the use cases have implicit authentication and authorisation steps.



**Figure 1: Data Storage and Access Use Case Diagram**

The **publish**, **update**, **delete** and **retrieve** use cases are each given the NextGRID name (as defined in the *NextGRID Naming Profile* [4]) of the **file** on which they are to operate (though in the case of **publish** the EPR of the **file** will be undefined). The **query** use case is given a set of criteria (expressed in some supported query language) which desired **files** should match. These criteria might comprise a single criterion matching only the name, of course. Typically the **query**

will have a preamble during which the **requester** interrogates the **store** to try to identify a query language in which the query can be expressed.

1. **Publish** a **file** by making it accessible from a **store**:
  - a. Negotiate the conditions under which the **file** is to be published in the **store**.
  - b. The **store** generates the EPR for the **file** in the **store**.
  - c. The **publisher** transfers the **file** into the **store**.
  - d. The **store** generates basic metadata for the **file**; additional metadata may be added by the **publisher**.
2. **Update** a **file** held in the **store**:
  - a. Lookup the EPR for the **file** from its NextGRID name [4].
  - b. Transfer the update into the **store**.
  - c. Modify the **file** and/or its metadata with the update.
3. **Delete** a **file** from a **store**:
  - a. Lookup the EPR for the **file** from its NextGRID name [4].
  - b. Delete the **file** from the **store**.
  - c. Delete the metadata entry for the **file** from the **store**.
4. **Query** the **store** to determine which **files** match a given condition:
  - a. Query the **store** and return a list of **files** that match the query. Depending on the details of the query, each entry in this list may contain some or all of the corresponding **file**'s metadata.
5. **Retrieve** a copy of a **file** from a **store**:
  - a. Lookup the EPR for the **file** from its NextGRID name [4].
  - b. Transfer a copy of the **file** and a copy of its metadata out of the **store**.

### 3.2 Indirect Access via an External Registry

Here the user does not know the name or EPR of the **file** which he wishes to access. Rather, he knows the name or EPR of an external registry which he thinks might contain entries (that is, metadata) for one or more suitable **files**. He first searches this registry (using some query language) to identify a suitable **file** and then accesses the **file**. There is an analogous case for **publishing** a **file**, in which in addition to adding a **file** to a **store**, an entry for it is added to an external registry. The basic use cases remain:

1. **Publish** a **file** into a **store**.
2. **Update** a **file** held in a **store**.
3. **Delete** a **file** from a **store**.
4. **Query** a **store** to determine which **files** match a given condition.
5. **Retrieve** a copy of a **file** from a **store**.

The actors and use case diagram are identical to the previous case where the **file** was accessed directly from a **store** (Figure 1, Section 3.1, above). The outline steps involved in implementing the various use cases are given below.

The **publish** use case is given the NextGRID name (as defined in the *NextGRID Naming Profile* [4]) of the **file** on which it is to operate. The EPR of the **file** will be undefined. The **update**, **delete**, **retrieve** and **query** use cases are each given a set of criteria which desired **files** should match. The criteria might comprise a single criterion matching only the name, of course. Typically these use cases will have a preamble during which the **requester** or **publisher** interrogates the registry to try to identify a query language in which the query can be expressed.

The steps in the various use cases are very similar to those for the corresponding use cases in the direct access case. To highlight the differences the steps which differ are shown underlined.

1. **Publish** a **file** by making it accessible from a **store**:
  - a. Negotiate the conditions under which the **file** is to be published in the **store**.
  - b. The **store** generates the EPR for the **file** in the **store**.
  - c. The **publisher** transfers the **file** into the **store**.
  - d. The **store** generates basic metadata for the **file**; additional metadata may be added by the **publisher**.
  - e. The **publisher** adds the **file**'s metadata (or a subset of it) to the external registry.
  
2. **Update** a **file** held in the **store**:
  - a. **Query** the external registry and identify a suitable **file**.
  - b. Extract the EPR for the **file** from its metadata obtained from the external registry.
  - c. Transfer the update into the **store**.
  - d. Modify the **file** and/or its metadata with the update.
  - e. If necessary, make corresponding changes to the **file**'s metadata in the external registry.
  
3. **Delete** a **file** from a **store**:
  - a. **Query** the external registry and identify a suitable **file**.
  - b. Extract the EPR for the **file** from its metadata obtained from the external registry.
  - c. Delete the **file** from the **store**.
  - d. Delete the metadata entry for the **file** from the **store**.
  - e. Delete the entry for the **file** from the external registry (to maintain the external registry's consistency).
  
4. **Query** the external registry to determine which **files** match a given condition.
  - a. **Query** the external registry and return a list of **files** that match the query. Depending on the details of the query, each entry in this list may contain some or all of the corresponding **file**'s metadata.
  
5. **Retrieve** a copy of a **file** from a **store**.
  - a. **Query** the external registry and identify a suitable **file**.
  - b. Extract the EPR for the **file** from its metadata obtained from the registry.

- c. Transfer a copy of the **file** and a copy of its metadata out of the **store**.

## 4 Simple Use Case to Query a Database

This section presents simple use cases in which a **client** queries a database hosted by a **store** and retrieves the **selection** generated by the query. In the basic case once the query has completed, the **selection** is instantiated as a **file** which is transferred from the **store** to the **client**. Two variant cases are also considered:

- Streamed results: the query is expected to generate a large **selection** which is streamed back to the **client**.
- Third-party transfer: the **client** and **store** are operating as elements of a workflow and consequently the **file** instantiated from the **selection** is transferred to a third element in the workflow.

### 4.1 Basic Query

A **client** queries a database hosted by a **store** and retrieves a **file** instantiating the **selection**.

#### Assumptions:

- The **client** knows which **store**, and which database if the **store** hosts more than one, that it wishes to **access**.

#### Actors:

- The **client**.
- The **store** which hosts the database.

#### Procedure:

The basic query of a database proceeds as follows.

1. The **client** sends an inquiry to the **store** requesting details about the database which it intends to **access** (typically the names of the database's tables and their attributes and the query language(s) supported).
2. The **store** replies with these details.
3. The **client** uses the results of this inquiry to formulate a query of the database. The query will be expressed in the selected query language.
4. The **client** sends this query to **store**.
5. The **store** queries the database using the query supplied.
6. Execution of the query generates a **selection** which is instantiated as a local **file** on the **store**. The **store** sends the EPR to **access** this **file** to the **client**.

7. The **client** retrieves a copy of the **file** using the EPR supplied and the NextGRID data transfer mechanisms [5].
8. The **store** tidies up by, for example, deleting its local copy of the **file**.

## 4.2 Variant 1: Streamed Data

Here the **client** anticipates that its query will generate a large **selection**. Consequently it arranges for the results to be streamed back to it. That is, tuples which satisfy the query are passed back to the **client** as soon as they emerge from the database management system and while the query is still executing, rather than waiting for the query to complete and then returning the complete **selection**. Note that in this case the **selection** is not necessarily ever instantiated as a **file**.

### Assumptions:

- The **client** knows which **store**, and which database if the **store** hosts more than one, it wishes to **access**.
- A large **selection** is anticipated.

### Actors:

- The **client**.
- The **store** which hosts the database.

### Procedure:

A streamed query proceeds as follows.

1. Proceed as for the basic case, above, up to and including step 5.
2. Stream the tuples back to the **client** as they are extracted from the database.
3. When the query completes any remaining tuples are despatched. When no more remain the stream terminates.

## 4.3 Variant 2: Third Party Transfer

Here the **client** and **store** are operating as two elements in a larger workflow, and the **selection** instantiated as a **file** is to be sent to a third element of the workflow rather than returned to the **client**. (Workflows in NextGRID are described in the *NextGRID Workflow Use Cases* [15], but see also the discussion in the *NextGRID Data Processing Use Cases and Requirements* [16], particularly Sections 2.1 and 2.2, pp5-6).

### Assumptions:

- The **client** knows which **store**, and which database if the **store** hosts more than one, it wishes to **access**.
- The workflow is already established and executing.

**Actors:**

- The **client**.
- The **store** which hosts the database.
- The workflow manager (which superintends the workflow).
- The third element of the workflow, which is to receive the **file** instantiating the **selection**.

**Procedure:**

A query followed by a third party transfer proceeds as follows.

1. Proceed as for the basic case, above, up to and including step 6.
2. The **client** notifies the workflow manager that the query is complete and advises it of the EPR of the results **file**.
3. The workflow manager initiates transfer of the **file** from the **store** to the third element of the workflow. There are variants in which other entities initiate the transfer, but they all have the characteristic that the EPR of the results is passed around until it reaches the appropriate entity.
4. When the transfer completes the third element starts its task.

## 5 Complex Use Cases

The OGSA *Data Access Scenarios* (OGSA-DAS) [17] present a number of use-case scenarios that can be implemented by OGSA Data Architecture components and interfaces. This section illustrates how two of these scenarios could be constructed from components of the NextGRID architecture.

### 5.1 Personal Data Service

#### 5.1.1 Description

The OGSA-DAS Personal Data Service (PDS; OGSA-DAS [17], Section 6, p30) describes a service that allows an individual to organise and manage a collection of personal data. The collection comprises a number of **datasets** that can be stored in a variety of geographically dispersed locations. Further, the user can access the collection from a variety of locations, none of which are necessarily co-located with any of the **datasets**.

Irrespective of either his own location or that of the **datasets**, the user is provided with uniform access to his collection. He can browse it, add new **datasets**, delete **datasets** *etc.* When manipulating **datasets** the PDS adopts the roles of both **publisher** and **requester** (see Figure 1 and Section 3.1, above).

The user sees only the items in his collection. That is, the collection has its own namespace, which the constituent **datasets** occupy.

### 5.1.2 Implementation

An outline of a NextGRID implementation might be as follows.

- The items constituting a user's collection are **datasets**.
- Each **dataset** will be identified by a NextGRID name [4].
- Each **dataset** will be stored in a **data store**. The **datasets** will not necessarily all be in the same **store**; typically they will not be. The various **stores** will usually be geographically dispersed.
- The PDS is an instance of a NextGRID data processing service [9].
- The principal function of the PDS is to maintain, for each of its users, a list of all the **datasets** in a user's collection and their locations. The PDS maintains the list, directed by the user.
- The list of a user's **datasets** could be stored in a NextGRID registry [18] which may be a component of the PDS (which remains a NextGRID data processing service) or it could be stored in a database internal to the PDS.
- Part of the functionality of the PDS is to manage any hierarchy or other rules associated with names in a personal collection. Such rules are contextually defined within an application or application-area and are not prescribed by NextGRID, except insofar as they must be consistent with the *NextGRID Naming Profile* [4].
- The PDS also manages the context of any session the user might maintain when interacting with it, as appropriate. The syntax and semantics of such sessions are not prescribed by NextGRID.
- In response to user instructions the PDS **publishes, updates, deletes, queries** and **retrieves datasets** (see Section 3.1, above) in the user's collection, accessing remote **data stores** as required and returning details to the user as appropriate. The PDS is acting as a broker for the user.

## 5.2 Data Replication Scenario

### 5.2.1 Description

The OGSA-DAS Data Replication Scenario (OGSA-DAS [17], Section 2, p10) describes a data replication scenario in which multiple copies of a **dataset** are maintained at various locations for reasons of availability, performance or as backup.

Users (acting as **requesters** or **publishers**; see Section 3.1) may access any of the copies of the **dataset**. The various copies of the **dataset** are automatically kept synchronised. Changes to one copy of the **dataset** are automatically propagated to all other copies of the **dataset**.

### 5.2.2 Implementation

Replication can be implemented in various ways and is an active area of research. The notes below suggest how the approach outlined in OGSA-DAS, Section 2.2, 'Scenarios' might be implemented in NextGRID.

- The core of the replication scenario is a replication service, which is an instance of a NextGRID data processing service [9].
- **Datasets** to be replicated are registered with the replication service. Metadata associated with the **dataset** indicates the various **data stores** where copies of the **dataset** are to be kept.
- The replication service stores details of every **dataset** that it is replicating in a NextGRID registry [18]. This registry is part of the replication service.
- When a **dataset** is added to the replication service it is copied to all the **data stores** specified by its metadata. Copying the **dataset** simply uses the standard NextGRID data storage [1], data access [2] and data transfer [5] mechanisms.
- **Clients** wishing to access a **dataset** search the replication service's registry to locate copies. Alternatively, they could search the individual **data stores**, if they knew, *a priori*, the likely locations.
- Once a copy is located in a suitable **store** it is simply accessed using the usual mechanisms for accessing a **dataset** in a **store**.
- Changes or updates to the **dataset** are notified to the replication service, which ensures that they are properly propagated and synchronised amongst all the copies.

## References

- [1] C. Davenhall, *NextGRID Data Storage Profile*, v0.7, NextGRID Project, 1 March 2008.
- [2] C. Davenhall, *NextGRID Data Access Profile*, v0.7, NextGRID Project, 1 March 2008.
- [3] D. Snelling, M. Fisher, A. Basermann, F. Wray, P. Wieder and M. SurrIDGE, *NextGRID White Paper*, v0.5, NextGRID Project, 25 July 2005, [http://www.nextgrid.org/download/publications/NextGRID\\_Architecture\\_White\\_Paper.pdf](http://www.nextgrid.org/download/publications/NextGRID_Architecture_White_Paper.pdf)
- [4] S. Davey and D. Snelling, *NextGRID Naming Profile*, V1.0, NextGRID Project, 26 February 2008.
- [5] M. Drescher and S. Davey, *NextGRID Data Transfer Profile*, V1.0, NextGRID Project, 19 February 2008.
- [6] J. Treadwell, *GFD-I.120: Open Grid Services Architecture Glossary of Terms*, V1.6, Open Grid Forum, 12 December 2007.
- [7] M. Antonioletti, M. Atkinson, A. Krause, S. Laws, S. Malaika, N.W. Paton, D. Pearson and G. Riccardi, *Web Services Data Access and Integration - The Core (WS-DAI) Specification*, Version 1.0, GFD-R-P.074, Open Grid Forum, 20 July 2006.
- [8] See, for example: [http://en.wikipedia.org/wiki/Safe\\_deposit\\_box](http://en.wikipedia.org/wiki/Safe_deposit_box) or <http://www.financial-ombudsman.org.uk/publications/ombudsman-news/30/30-box.htm>
- [9] C. Davenhall, *NextGRID Data Processing Profile*, v0.5, NextGRID Project, 1 March 2008.
- [10] For the Storage Resource Management Working Group see: <http://sdm.lbl.gov/srm-wg/>
- [11] A.C. Davenhall, C.L. Qin, K.T. Noddle and N.A. Walton, *The AstroGrid MySpace System*, in *Astronomical Data Analysis Software and Systems (ADASS) XIII*, Proceedings of the conference held 12-15 October, 2003 in Strasbourg, France, Francois Ochsenbein, Mark G. Allen and Daniel Egret (eds), *ASP Conference Proceedings*, **314** (San Francisco: Astronomical Society of the Pacific), 2004, p330.
- [12] A.C. Davenhall, C.L. Qin, G.P. Shillan, K.T. Noddle and N.A. Walton, *The AstroGrid MySpace Service*, in *Optimizing Scientific Return for Astronomy through Information Technologies*, Peter J. Quinn and Alan Bridger (eds), *Proceedings of the SPIE*, **5493**, 2004, pp254-261.
- [13] S. Davey and D. Snelling, *NextGRID Naming and Addressing Use Cases and Requirements*, V1.0, NextGRID Project, 26 February 2008.
- [14] D. Snelling, *NextGRID SLA Schema*, v0.2, NextGRID Project, 7 July 2007.
- [15] J. Ferris and N. Matskanis, *NextGRID Workflow Use Cases*, v0.4, NextGRID Project, 6 August 2007.



- [16] C. Davenhall, *NextGRID Data Processing Use Cases and Requirements*, v0.6, NextGRID Project, 1 March 2008.
- [17] D. Berry , A. Luniewski and S. Davey, *OGSA Data Architecture Scenarios* (draft), 14 August 2007,  
[https://forge.gridforum.org/sf/docman/do/downloadDocument/projects.ogsa-d-wg/docman.root.working\\_drafts/doc14069](https://forge.gridforum.org/sf/docman/do/downloadDocument/projects.ogsa-d-wg/docman.root.working_drafts/doc14069)
- [18] P. Hasselmeyer, M. Surridge and P. Wieder, *NextGRID Registry Profile*, V1.0, NextGRID Project, 5 February 2008.