The ‘Service Cache’ Pattern and Grid Services in the European Grid of Solar Observations (EGSO)

S. Martin† and G Piccinelli‡

† Rutherford Appleton Laboratory, ‡ University Colloge London

Abstract: The EGSO project has developed a re-usable messaging infrastructure for communication between Grid applications, whereby message-relay services transparently leverage directory, storage and logging services in order to provide enhanced functionality to consumer applications. This paper describes the proposed *Service Cache* pattern, which has been applied to the engineering of these services. The main potential benefits derived from the application of this pattern include improved manageability and predictability of service delivery, and faster response times for consumers.

1 Introduction

The EGSO (European Grid of Solar Observations [1]) system architecture [2] brings together three distributed and decoupled roles: data *consumers*, data *providers*, and *brokers*. Consumers discover the data and services that providers offer by sending queries to brokers. Providers register themselves with the brokers as they join the Grid, and periodically furnish brokers with new metadata describing what they have to offer. The broker then uses this information to forward consumer queries to the appropriate provider(s) in order to provide more detailed results. These results are aggregated by the broker and ultimately returned to the consumer. Consumers may then directly interact with providers based on these results, for example to retrieve data.

These roles interact using the EGSO Communication Infrastructure (ECI) [2], a set of services that support communication between Grid applications and which embody the concepts of a Service-Oriented Architecture (SOA). Messaging services are used to transfer arbitrary data between applications; additional services, including logging, storage and directory services, are transparently orchestrated by the messaging services to provide additional functionality. Directory services are used to register the ‘existence’ and location of applications within the Grid. The logging services allow transactions to be logged, as one might expect, and the storage services allow applications to store data safely and securely within the Grid. The overall system architecture, whereby role applications communicate via the ECI, has allowed these services to be developed in such a way that makes them immediately reusable outside the EGSO context.

2 The ‘Service Cache’ Pattern

In this section we propose the ‘Service Cache’ pattern, an architectural pattern [3] which emerged during the design and development of the services described previously. The pattern is described below in terms of its purpose, typical usage, design, implementation, and consequences:

Purpose

Service Cache is an architectural pattern for use in distributed systems in which service consumers are provided with a locally hosted *service access component* with which they interact in order to obtain the functionality of the service. Interacting with the access component protects the consumer from variability of network latency or even network outages whilst making effective use of any spare resources which the consumer can provide to reduce the burden imposed on service providers.

Typical Usage

In principle, the pattern can be applied to any service which is involved with publishing and disseminating data, such as the types of service described above. Other services in which data must be

instantly made globally available, perhaps involving real-time data or performing calculations, might not be suitable candidates for this pattern.

Design

The overall service is split into two cooperating components: a *service access* component, and a main *service implementation* (see Figure 1). The service access component is hosted locally with the consumer application(s), and is accessed via the use of Web Services. In order to leverage service functionality, a consumer *only* interacts with the access component; this component provides the consumer's view of the service and it transparently interacts with the implementation component to provide the service functionality to the consumer. It is also possible that specific access components could be tailored to allow different applications to use generic services. In principle, multiple access components can cooperate with one main service implementation.

Implementation

Since the pattern is applied to services which maintain published data, the access component must contain a cache which can be both written to and read from; the data within the cache has a finite lifetime. If the service requested by a consumer involves publishing new data from the consumer (whether it is a logging event, directory entry or file to be stored), then this data is stored in the access component's cache. Periodically, the main service implementation will poll the access component to see if new data has been stored; if it has, this new data is uploaded to the main service. If the consumer is requesting data from the service (e.g. retrieving a stored file or looking up a directory entry), the access component will first check to see if the data is stored locally; if it is, this can be returned to the consumer, else the access component will retrieve the required data from the main service implementation on the consumer's behalf, and will add this to its own cache so that it keeps frequently requested items in the cache.

As discussed above, the access component is accessed via Web Services, as is the main service implementation so there is a minimum requirement for a container to be available local to the consumer for hosting the access component, although this does not imply any platform or language restrictions. The cache in the access component and the storage in the service implementation can be implemented in any manner, although use of a RDBMS is likely (as shown in Figure 1).

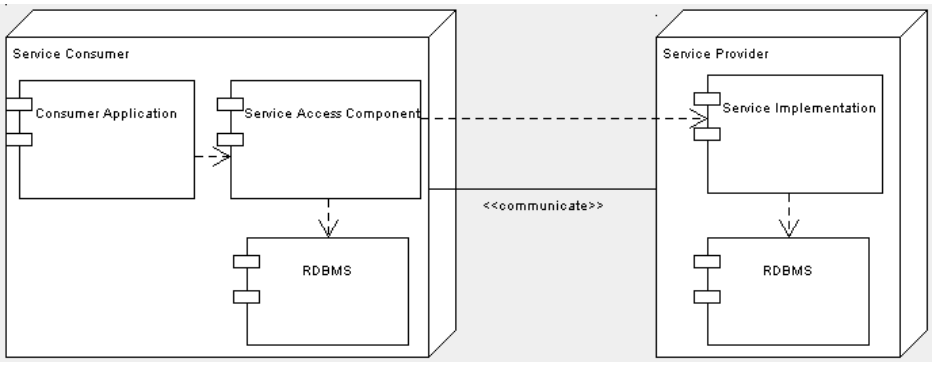

Fig. 1. A UML deployment diagram illustrating the service cache pattern. The consumer accesses the service via the access component and is unaware of the division of labour between the access component and service implementation. Also shown are two RDBMSs which are used to store published data.

Consequences

The main potential benefits of the pattern are improving the response times from a consumer point of view, for both publishing and retrieving data, and the possibility for the main service implementation to maintain SLAs since there is the option of having more control over when the main service receives new data. Improved response times for consumers result from fewer requests using external networks since they can be dealt with locally; those requests which are forwarded to the main service

implementation are serviced faster due to less activity at any given time. In terms of fulfilling SLAs and managing demand, the main service implementation can request new data from individual access components at different times, and can schedule these updates to occur during quiet periods.

There are a few potential minor drawbacks associated with this approach. Data published by one consumer to an access component will not be available to other consumers using separate access components until it has been added to the main service. Also, there is the requirement for the consumer to be able to host the access component locally, along with any specific software which may be required, such as a database or Web container.

3 Current Status and Summary

We have applied the pattern to the Logging, Directory and Storage services which have been developed for use within the EGSO project. We are currently in the process of analysing response times [4] for consumers using these services to both publish and retrieve various types of information, as we vary the cache age limit and update frequency.

In this paper we have proposed the 'Service Cache' pattern for use by service consumers within a distributed environment; the pattern aims to provide benefits for both consumers and service providers through improved response times and predictability of service demand respectively.

Once we have completed the analysis of our test results, we aim to apply the pattern to the services which are deployed in the production version of the EGSO system; we will then be in a position to investigate the benefit of the pattern under the demands of real-world use in a Grid environment.

References

[1] European Grid of Solar Observations, http://www.egso.org

[2] Piccinelli, G. (ed.): EGSO Architecture v2.0, http://www.cs.ucl.ac.uk/staff/C.Gryce/wp1/egsoArchitecturev2.0.pdf (2004).

[3] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns. John Wiley and Sons, Chichester (1996).

[4] Menascé, D. A., Almeida, V. A. F.: Capacity Planning for Web Services – Metrics, Models, and Methods. Prentice Hall, New Jersey (2002).