Engineering Grid Services in the European Grid of Solar Observations (EGSO)

Nathan Ching[1], Simon Martin[2], and Giacomo Piccinelli[3]

[1] Mullard Space Science Laboratory, Holmbury St. Mary, Dorking, Surrey, RH5 6NT. e-mail: nc@mssl.ucl.ac.uk
[2] Space Science and Technology Dept., Rutherford Appleton Laboratory, Chilton, OX11 0QX. e-mail: simon.martin@rl.ac.uk
[3] Computer Science Dept., University College London, Gower Street, London, WC1E 6BT. e-mail: g.piccinelli@cs.ucl.ac.uk

The EGSO (European Grid of Solar Observations) system architecture [1] brings together three distinct roles: data *consumers*, data *providers*, and *brokers*. Put simply, a consumer interacts with a broker in order to get indications of the provider(s) likely to hold specific data (or offer a specific service). The broker provides the consumer with references to these providers, as well as information that the consumer can use as part of the selection process. The consumer then interacts with the provider(s) in order to refine the data (service) requirements. In the end, the provider delivers the data (service) to the consumer directly.

Applications within the various EGSO roles interact using the *EGSO Communication Infrastructure* (ECI), which is based upon *Service-Oriented Architecture* (SOA) concepts and comprises a set of communication and interaction services (i.e. middleware). The ECI currently consists of four types of service; the *Partner Interaction Service* (PIS), the *Logging Service,* the *Storage Service,* and the *Directory Service.* Individually, these services offer message-oriented data transfer, logging, storage, and security capabilities, but only the PIS is visible to applications (service consumers). The PIS transparently orchestrates the use of the other services to provide extra functionality to the consumer. As applications join the Grid, they use a PIS instance to interact with other applications; this instance registers their existence within the Grid with the Directory service, so that other applications can find them and interact with them. The Logging service allows transactions to be logged, as one might expect, and the Storage service allows applications to store files and messages safely and securely within the Grid. The overall system architecture, whereby role applications communicate via the ECI, insulates application developers from the complexity of interaction middleware, and has allowed the ECI to be developed in such a way that makes it immediately reusable outside the EGSO context.

The *Service Cache* pattern which we propose in this paper has been applied to the engineering of the Logging, Storage and Directory services found in the ECI. This pattern divides up a service into an *access component* (which is deployed locally with the consumer) and a main service *implementation* (see Figure 1). Rather than simply interacting with an instance of a service, the PIS (or any service consumer) interacts with the access component belonging to the service; the consumer is unaware of the exact division of labour between the two parts of the service. The basic premise is that the access component provides the consumer's view of the service and the overall service functionality to the consumer. So for example in the case of the Directory service, the access component provides an API that allows the consumer (PIS) to add,

modify, remove and search records of other services and applications; the consumer is unaware of the location of the records.

The exact way in which the access component and service implementation cooperate to deliver the service functionality depends on the type of service involved. The main method of interaction between the two parts of a service is that following a trigger event, the access component will update the main service implementation with its latest information; for example, in the case of the Directory Service, this includes adding new records, removing old ones, or modifying existing ones. The purpose of this approach is to give control to the service, so that demands placed on the service's resources are manageable and predictable; this allows SLAs to be fulfilled, and there is a potential to improve performance and QoS as well. Possible trigger events might be periodic polling of the access component by the main service, or they might be automatic; for example if the system where the access component resides is shutting down, the access component may wish to update the main service before shutdown. There is another method of interaction whereby the access component acts as a proxy for the main service instance; for example, if the consumer requires some information from the service that is not available from the access component, the access component can request if from the main service implementation and transparently return it to the consumer.

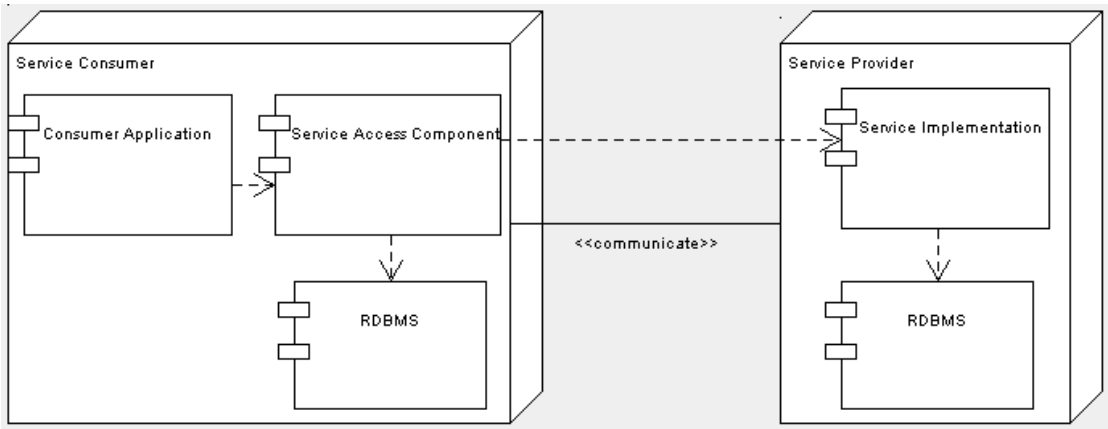

***Figure 1**: Deployment diagram illustrating the use of the 'Service Cache' pattern. The service consumer interacts with the service via the access component, which is deployed locally with the consumer, and is unaware of the division of labour between the access component and main service instance. The access component and service implementation cooperate so that the consumer can access the full functionality of the service.*

In this paper, we discuss in detail the principles, purpose and usage of the *Service Cache* pattern, and drawing from our experience of using it within the EGSO project; we outline the benefits and consequence of using such a design pattern in a Grid environment.

References

[1] http://www.cs.ucl.ac.uk/staff/C.Gryce/wp1/egsoArchitecturev2.0.pdf