Chapter V
Dynamic Delegation of Authority in Web Services

David W. Chadwick
Computing Laboratory, UK

ABSTRACT

Delegation of authority (DOA) is an essential procedure in every modern business. This chapter enumerates the requirements for a delegation of authority Web service that allows users and services to delegate to other users and services authority to access computer-based resources. The various models and architecture that can support a DOA Web service are described. A key component of the DOA service is the organisation’s delegation policy, which provides the rules for who is allowed to delegate what to whom, and which needs to be enforced by the DOA service. The essential elements of such a delegation policy are outlined. The chapter then describes a practical DOA Web service that has been built and piloted in various grid applications. It concludes by reviewing some related research and highlighting where future research is still required.

INTRODUCTION

Delegation of authority is an essential procedure in every modern business. A delegate is defined as “A person authorized to act as representative for another; a deputy or an agent” (www.dictionary.com). Without delegation of authority (DOA), managers would soon become overloaded. DOA allows tasks to be disseminated between employees in a controlled manner. A delegate may be appointed for months, day, or minutes, for one task, a series of tasks, or all tasks associated with a role. DOA needs to be fast and efficient with a minimum of disruption to others. Delegators should not need permission from their superiors for each act of delegation they undertake, or otherwise their superiors would soon become overburdened with delegation requests from
subordinates. Instead, a delegation policy should be in place so that delegators know when they are empowered to delegate (i.e., what and to whom) and when they are not.

The recipient (or service provider) who is asked to perform a service for a delegate should be able to independently verify that the delegate has been properly authorized to act as a representative for the delegator, before granting the request. If the delegate has not been properly authorised, the delegate’s request should be declined. The recipient will therefore enforce the delegation policy of its organization and deny service requests from unauthorized delegates.

In a computing environment there is also a need for DOA. One computer process may need to delegate to another computer process. One person may need to delegate his privileges to another person in order to allow the later to undertake the computer-based tasks of the former. Similarly, in a service-oriented world, computer services also need the ability to delegate tasks to other services, so that the latter can perform subtasks on the former’s behalf. Service providers need to be able to verify that each service requestor is properly authorized. If the service requestor has been dynamically delegated authority by another authorized entity, service providers need to be able to verify that this was done in accordance with their delegation policy.

The objective of this chapter is to present a model for dynamic delegation of authority in a Web services world, in which users can delegate to other users, services to other services, and users to services. This chapter also describes a current implementation of this model and compares and contrasts it with other delegation systems that only partially implement the model.

**BACKGROUND**

In Grid computing today, which is based on Web services, delegation from a user to his Grid job is enacted via the process of proxy certificates (Tuecke, Welch, Engert, Pearlman, & Thompson, 2004). The purpose of these is two fold. Firstly, it allows a user to start a Grid job, and then leave it to run in his absence for as long as is required, without him needing to be there to continually log in and authorize the use of new Web services by the job. Secondly, it allows the job to migrate around the Grid, and to spawn new subtasks to run on other machines as necessary. These subtasks can themselves authenticate as proxies of the user and consume Web services (or resources) that the user is entitled to have. The process works as follows. The Grid user, who must have an asymmetric key pair and X.509 certificate, initializes his Grid job, and during this process the job creates its own asymmetric key pair. The user then issues an X.509 proxy certificate for the Grid job, which certifies the public key of the job. The proxy certificate also contains the name of the job (which must be subordinate to the user’s own distinguished name), the name of the user as the issuer, and the signature of the user. The Grid job can now authenticate to any Web service Grid resource by digitally signing requests using its own newly created private key, and the Web service can authenticate the job using the job’s newly created proxy certificate. Because the name of the Grid job is subordinate to that of the user, then the Web service knows that it has to check if the user is authorized to access this service, and if so, then the service is to be consumed on behalf of the user. When a new subtask needs to be spawned, to run elsewhere on the Grid, the spawned subtask can generate its own new asymmetric key pair, and the original Grid job can issue a second proxy certificate for the spawned subtask, with a name that is subordinate to its own. In this way the job can delegate as necessary in order to achieve its aims. In each case, the Web service checks if the user, and not the job itself, is authorized to consume its resources. This is easily achieved because the name of the job is linked to the name of the user by being subordinate to it.