Chapter 10
A Service-Oriented Computing Platform:
An Architecture Case Study

Michael Sobolewski
United States Air Force Research Laboratory, USA & Polish-Japanese Institute of Information Technology, Poland

ABSTRACT

In Service-Driven Computing, the client-server architecture describes the relationship of cooperating programs in a distributed application. The providers of a resource or service execute workloads submitted by service requestors. Web service, Grid, and Cloud Computing technologies are based on the client-server architecture. A true service-oriented architecture describes everything, anywhere, anytime as a service. This chapter presents the SORCER (Service-ORiented Computing EnviRonment) platform, which provides service-oriented modeling or programming or both (mogramming) environments within its operating system that runs front-end service-oriented mograms and dynamically manages corresponding federations of local and remote service providers. The architecture of SORCER is described with the focus on service-oriented mogramming, service context-awareness, and its operating system managing everything as a service type. A case study report illustrates how SORCER is used for a conceptual design of the next generation of efficient supersonic air vehicles.

DOI: 10.4018/978-1-4666-6178-3.ch010

INTRODUCTION

From the very beginning of networked computing, the desire has existed to develop protocols and methods that facilitate the ability of people and automatic processes across different computers to share resources and information among computing nodes in an optimized way. As ARPANET (Postel et al., 1981) began through the involvement of the NSF (Lynch et al., 1992; Postel & Reynolds, 1987) to evolve into the Internet for general use, the steady stream of ideas became a flood of techniques to submit, control, and schedule workloads across distributed systems (Lee, 1992).

The term "metacomputing" (Metacomputing, n.d.) was coined around 1987 by NCSA Director, Larry Smarr. The *metacomputer* is, simply put, a collection of computers held together by state-of-the-art technology and *balanced* so that, to the individual user, it looks and acts like a single computer. The constituent parts of the resulting metacomputer could be housed locally, or distributed between buildings, and even continents. Globally distributed service-oriented (SO) computing systems are metacomputing systems.

The latest in client-server ideas are the Grid (Foster et al., 2001) and Cloud (Linthicum, 2009), intended to be used by a wide variety of different users in a non-hierarchical manner to provide access to powerful aggregates of resources and services. Grids and Clouds, in the ideal, are intended to be accessed for computation, data storage and distribution, and visualization and display, among other applications, without undue regard for the specific nature of the hardware and underlying operating systems on the resources on which these jobs (executable codes) are carried out. While a Grid is focused on computing *resource* utilization, Clouds are focused on platform *virtualization* in computer networks. In general, Grid and Cloud Computing is client-server model that abstract the details of the server away by requesting a *service* (resource), and not a specific *server* (machine). However, both terms are vague from the point of view of service-oriented (SO) architectures and related programming models. In those cases the architecture refers usually to a form of API-based, client-server middleware-managing services and clients written in general-purpose software languages with no front-end SO layer. It means that at the back-end, each time a new service (code) has to be developed; it must be deployed at the server by software developers and later accessed by the end users in their applications via the middleware API. In these systems there is no front-end option that allows the end user to create at runtime complex service collaborations directly from existing services in the network.

As we reach adolescence in the information era, we are facing the dawn of the service era, an era that will be marked not by PCs, workstations, and servers, but by computational capability that is embedded in all things around us exposing ubiquitous services. A service is the work performed in which a service provider (one that serves) exerts acquired abilities to execute a computation. Service providers just consume services and provide services from and to each other respectively. Applications are increasingly moving to the network - self aware, autonomic networks that are always fully functional. The service providers implement instructions of the virtual service processor (metaprocessor). In a true SO system *everything anytime anywhere is a service* by the analogy to an object-oriented system where everything is an *object*. Therefore for each service there is a corresponding local or remote service provider and all service providers are treated by the SO platform in a uniform way.

As we move from solving problems of the *information era* to more complex problems of the *service era*, it is becoming evident that new programming or modeling or both (mogramming) languages are required. These languages should reflect the complexity of computing problems we are already facing in development of complex adaptive systems by large collaborative teams that use hundreds of executable codes written in