Grids of Grids of Simple Services

Geoffrey Fox
Community Grids Laboratory
Indiana University
gcf@indiana.edu

Grids

Here we propose a way of describing systems built from Service oriented Grids in a way that allows one to build new Grids by composing and adapting existing collections (libraries) of Grids. We also suggest some "best practices" in deciding how to architect services and package systems.

We have of course discussed Grids extensively here in previous articles and we adopt the view that they represent the system formed by the distributed collections of electronic capabilities that are managed and coordinated to support some sort of enterprise (virtual organization). Sometimes one reserves Grid to describe just the technology used to build these electronic communities or organizations. One thinks of Grid technology as the CyberInfrastructure (NSF) or e-Infrastructure (European Union) that supports e-Science, e-Business or in fact e-moreorlessanyenterprise. There is no firm consensus as to the best Grid approach but we will adopt the popular architecture based on Web services. There is a vigorous debate in the community as to the "right" way to do this and if conventional Web services need enhancement to cope with the large scale secure managed distributed services needed in a Grid. In particular there is lot of debate on the appropriate ways to represent state and how much to standardize in this area. WSRF (Web Service Resource Framework http://www.globus.org/wsrf/) and WS-GAF (Web Service Grid application Framework http://www.neresc.ac.uk/ws-gaf/) are two important activities whose development and interaction will have important implications for the detailed structure of services. However here we discuss aspects independent of these issues – namely "what is the right size" for a service and how should one package services and Grids together. Often one considers Grids as providing seamless access to a set of resources; here we adopt this view with however an architecture with many "small Grids". This reflects the many different types of overlapping communities and resource collections that naturally form individual Grids. Each individual Grid can have a seamless elegant environment – in fact this could be a criterion for defining Grids – but a composite Grid would amalgamate multiple such subGrids and exhibit a resultant heterogeneous environment. In other words, we do not expect there to be a few Grids produced but very many that can get composed, divided and overlapped together to support dynamic communities and requirements.

Services

The service oriented architecture SOA used by Grids is subtly different from previous distributed systems built with COM CORBA and Java with new ideas to enhance especially interoperability and scalability. Key features of (Web) services in today's Grids include:

1) Architectures that choose wherever possible message-based and not method or RPC based linkage of capabilities. This produces lightweight loosely-coupled services that can be distributed and replicated to achieve needed performance and functionality.
2) Interfaces defined with XML based SOAP and WSDL technologies that support a wide set of implementations trading off performance, ubiquity and functionality.

The first feature of loose message based coupling is certainly not very precise. The traditional distributed object model produces components that exchange messages typically with an RPC (Remote Procedure Call) or equivalently RMI (Remote Method Invocation in Java). These are coupled messages corresponding to the distributed version of a traditional method call and its return. Loose coupling for services corresponds to a messaging strategy where individual message are not directly coupled in pairs but response messages are generated if needed asynchronously from the original communication. The second requirement of services – XML based specifications of the service interfaces and their associated messages – is important for interoperability but less distinctive in its architectural implications; it roughly corresponds to a different specification language from the IDL (CORBA) or Java used in RMI.

Consider any (software) problem you like and imagine how it would look in a traditional approach of a decade or so ago. One would get a giant glob of software in some language like C++ or perhaps even Fortran. This would be divided into methods or subroutines and we would be browbeaten to build it in modular fashions using libraries and well defined interfaces. Us people from the past would have given up use of the GOTO in Fortran and adopted better practice for specifying control structures. As technologies developed we added new languages like Java and better software engineering processes where the latter were adopted more broadly in industry than academia. As implied above, distributed object technology supported the implementation of this paradigm across multiple computers with the method or procedure calls implemented as paired messages. However most software systems still consisted of large globs with each glob having multiple functionalities. One can find lots of very useful and important examples of this for Java at the Apache site (http://www.apache.org) . One can convert such code into services by specifying each of interfaces in XML and providing a Web Service wrapper. This activity is important for jump starting our collection of services but I would view it as an interim step. For example looking at the many different Apache projects, one will find many related but different implementations of common subservices like security and user profile. Building a system combining several projects would often require an integrated approach to common services like security. This would be relatively easy if the implementation of each subservice like security was a separate Grid service with well defined message-based interfaces. However with traditional approach, the typical subservice can have an external message-based interface but unfortunately in addition many internal method linkages to other parts of the software glob. Thus subservices like security cannot be extracted from the glob and it is very hard to compose such traditional software systems even if they run excellently with service interfaces.

The above rambling discussion allows us to identify a strategy for defining services. Start by examining the different capabilities of one's systems. Services are distributed components that have distinct functionality – especially functionality that is usefully

shared among different uses. Services must be able to achieve acceptable performance when implemented with message based interfaces and distributed platforms. In the

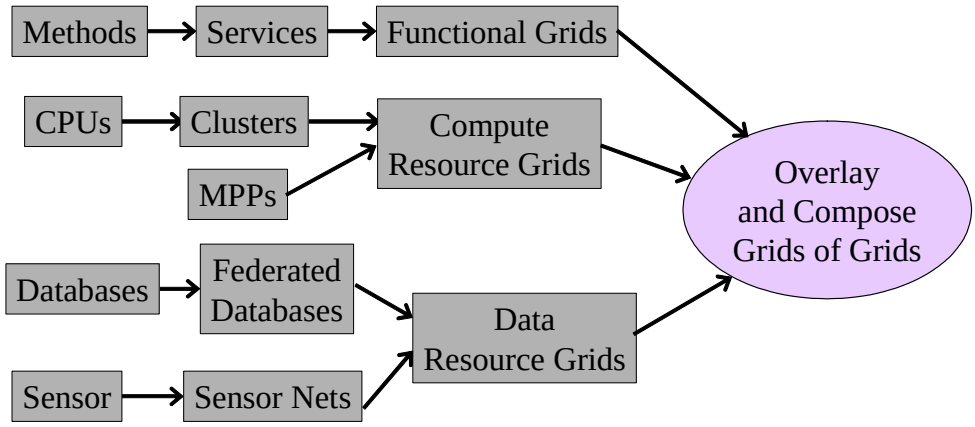

Fig. 1: Composing Functionality and Resources in the Grid of Grids

January February 2004 issue, we discussed the inevitable difference in latency between message and method based interactions; messages could experience 100's of milliseconds in network latency while this is reduced to a millisecond or so for communication between nearby services. One should build services that are as small as possible given the performance implications from the decomposition. Services are then the unit to which traditional programming models and languages apply. We will not discuss this aspect but rather take services as the atomic unit whose management and packaging into Grids need to be discussed.

Packaging Services and Resources into Grids

In this article Grids represented a packaging and coupling approach that generalizes and distributes that familiar from the traditional software hierarchy:
lines of code → methods (subroutines) → objects (programs) → packages (libraries).
As shown in fig. 1, Grids can be considered in this fashion with the basic unit of distribution being a service or a resource. However a given Grid is not the last word but rather can itself be the building block in larger Grids. Thus we propose to build systems as Grids of Grids with the smallest Grids being just single services or resources. Note that we view a cluster as a special resource Grid shown in fig. 1. We have chosen to separately specify Grids that correspond to resources (made up of data repositories, sensors and CPUs) as well as those corresponding to functionalities (software services). However this is a little confusing as every resource is represented in the Grid by a service. Thus we could simplify the above picture and just talk about services. Note some unification of well known concepts; an individual Grid service could correspond to a single database (using the OGSA-DAI technology described in the July/August 2003 column). A federated database then corresponds to a Database Grid. Again individual CPU could have a Grid Service interface and then a cluster Grid corresponds to a cluster of CPU's.

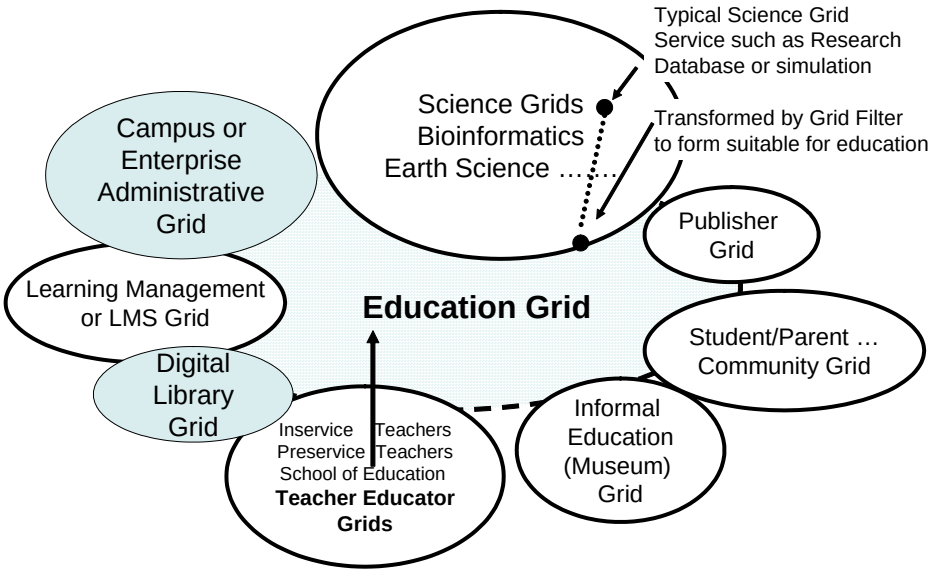

Fig. 2: Science Education as a Grid of Grids

Another example can be taken from education and arises when you try to take science Grids and use them in schools and universities. As shown in figure 2, education involves many separate communities and capabilities that can be expected to form independent electronic (virtual) organizations supported by their own Grid. An Education Grid is formed as a Grid of Grids by linking and adapting services in the component Grids.

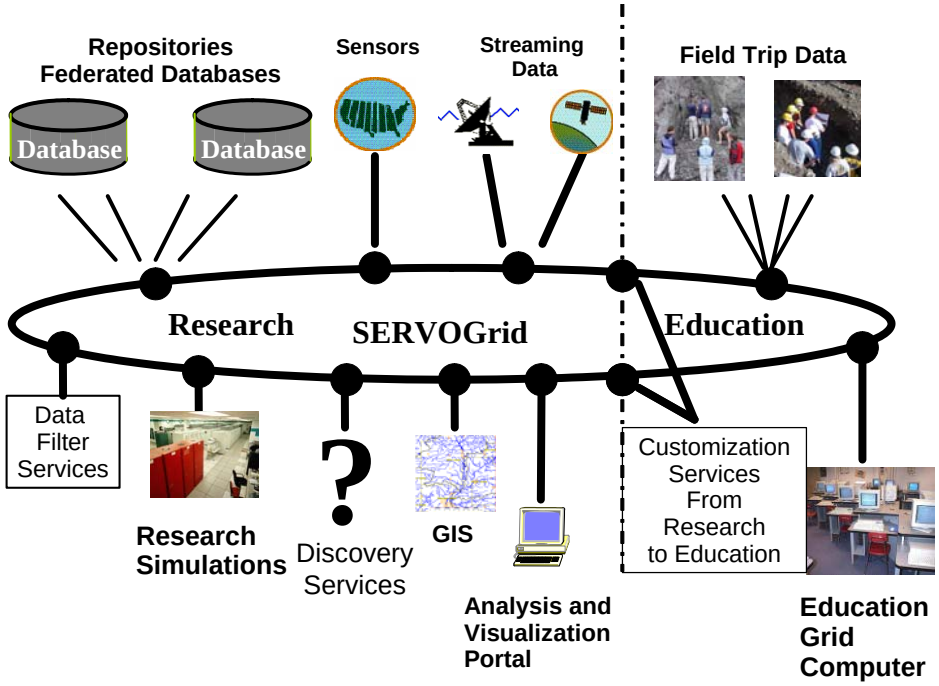

Fig. 3: Geoscience Research and Education Grids

Figures 2 and 3 illustrate the key idea of transformations or filters used to adapt services

in old component Grid to the Education. This could take research simulation or database services and simplify them for use in education. The resultant Education Grid will consist of three types of services. Firstly those that are unique to education such as educational meta-content (lesson plans and objectives), online knowledge bases, grading and homework services, as well as federal and state standards; these could be delivered by the learning management and digital library Grids. Figure 3 shows the details of a Science Grid for Earthquake science (http://www.servogrid.org) linked in this fashion with other Grids to form a Geoscience Education Grid. We show field data being gathered by students as part of the category of education specific services. Secondly there are services like collaboration that are essentially the same as those developed for other Grids; thirdly there are the transformed Grid resources that were developed for research but have been transformed to directly support teaching & learning.

Thus we propose that one first build "lean and mean services" as discussed in the previous section. Then we package these into "atomic (basic) Grids" covering core functionalities and services; Geoscience, digital library and learning management systems are atomic Grids discussed above. Later we will describe other such basic Grids including GIS (Geographical Information Systems), Flood or electrical power simulation. Fig. 3 shows the Geoscience Grid using a GIS Grid as a component. After defining the basic Grids, most operational Grids will be built by linking component Grids together. In many

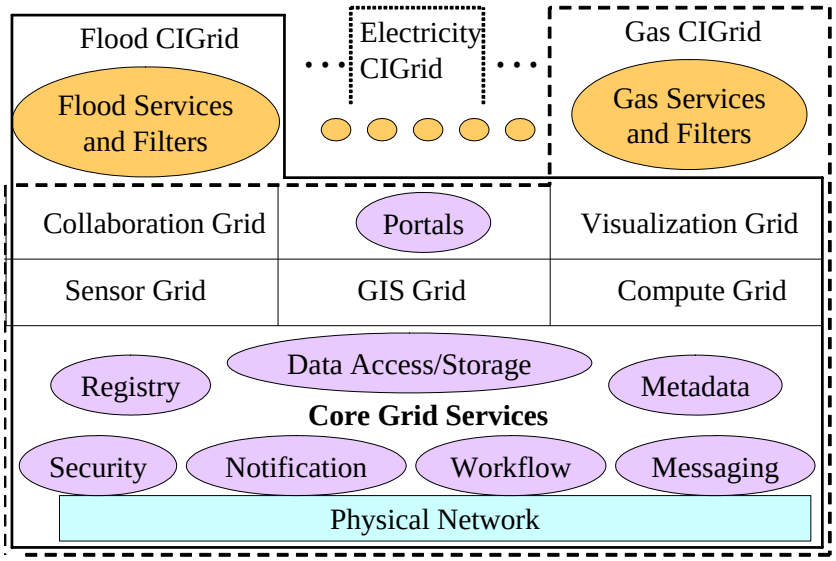

Fig. 4: Critical Infrastructure (CI) Grids built in composite fashion

cases the component Grids need customization; this is achieved by adding services to filter or transform the services of the component Grids. This gives our final packaging as Grids of Grids.

As a further example of a Grid of Grids, Fig. 4 illustrates how one can build Grids to support the strategic or critical infrastructure of the nation. The Department of Homeland Security has identified these infrastructures that include Agriculture and Food, Water, Health, Industrial and Defense Base, Telecommunications, Energy, Transportation, Banking and Finance, Chemical Industry and Hazardous Materials, Postal and Shipping.

The critical atomic Grids in this case include those for sensors, GIS, visualization, computing and collaboration. We also need of course the core Grid shown at the bottom of the figure with services like security, notification and meta-data. These atomic Grids can be re-used as shown in figure 4 in all critical infrastructure Grids and illustrate the important interoperability principles with which Grids are built. These CI(Critical Infrastructure) Grids are in turn customized, composed and overlaid with other Grids (such as weather, census data) for different CI communities. This way one generates Grids aimed at Public Health, Emergency Response (Command and Control) or Crisis Grids, Infrastructure Planning, Education (schools) and Training (of managers and first responders). Clearly the Grid of Grids concept can be applied recursively and dynamically.

Conclusions

We have presented a model of building systems hierarchically with traditional software engineering describing the structure of individual services. Services are aggregated into atomic Grids that perform core functionalities. Composite Grids are built recursively from both atomic and other Composite Grids. This is similar to traditional software models with components and libraries. The use of transformation services in this coupling is an interesting feature distinguishing this packaging from that familiar from libraries. Although there is a lot of research on the workflow technology supporting the composition of services (http://www.extreme.indiana.edu/groc/ggf10-ww/index.html), little consideration as been given to capabilities seen in modern IDEs (Integrated Development Environments) for traditional software models and supporting them for the higher level of integration seen in Grids of Grids. In fact it is hard to support my earlier suggestion to make services as small as possible given the poor support for managing them. We expect the ideas described here to receive increasing attention in the future with the growing importance of software engineering and its extension to services.