DESIGNING COMPUTATIONAL GRIDS USING BEST PRACTICES IN SOFTWARE ARCHITECTURE

*Hema Prem and **N. R. Srinivasa Raghavan

**Lead SETLabs EMEA Business Program, Infosys Technologies Limited*
14 Upper Bank Street, Canary Wharf, London, UK, E14 5NP

***Lab group manager, Customer Driven Advanced Vehicle Development,*
General Motors India Science Lab, Bangalore 560066

Email: hema_prem@infosys.com, srinivasa.raghavan@gm.com

ABSTRACT

The basic principle of sharing and collaborative work by geographically separated computers is known by several names such as meta computing, scalable computing, cluster computing, internet computing and this has today metamorphosed into a new term known as grid computing. Grid computing is proving to be a promising method of HPC, which is packaged with many challenges. This paper elucidates the role that patterns can play in architecting complex systems with specific reference to grid computing. We provide descriptions of a set of well-engineered patterns that the practicing developer can apply to crafting his or her own specific applications. We develop the Software Requirements Specification (SRS), with an attempt to drive to effectual design specifications for use by any grid developer. We analyze the grid using an Object Oriented approach and present the design using the unified Modeling language (UML) which itself helps the identification of patterns at different phases.

Keywords: Grid computing, Unified Modeling Language, Analysis Patterns, Design Patterns.

1.0 INTRODUCTION

Rapid technology development in several domains is the result of the explosive growth of the Internet and of distributed computing in general. Increasingly, the high-performance computing community is blending its (parallel/cluster/ distributed) computing technologies to meet its performance needs. Grid Computing is proving to be one such promising method of high performance computing, packaged with many challenges. The initial vision of the grid is presented in (Foster and Kesselman, 1998). Grids have moved from the obscurely academic to the highly popular. One reads about Compute Grids, Data Grids, Science Grids, Access Grids, Knowledge Grids, Bio Grids, Sensor Grids, Cluster Grids, Campus Grids, Tera Grids, and Commodity Grids (Foster, 2002). Consider a novice analyst/developer in the field of grid computing, with an objective to build a working prototype of a grid. Implementation of the prototype would require rigorous analysis of the problem domain especially when the problem at hand is very complex. Since many frameworks are already available, the analyst would refer them and resort to reinventing another similar procedure of building the prototype.

Thus it is seen that there are many grid frameworks that are built or are in the process of being functional. All these grids differ in some functionality or the other, though the basic principle over which the grids are built is the same. Commonly occurring scenarios could be documented as patterns, so as to promote reusability. These patterns could be customized according to the application needs and would act as catalyst in the analysis, design and implementation stages. We feel that it would be a lot easier if the developer could choose from a set of patterns to craft his/her own applications. A pattern oriented approach can lead to an efficient and speedy development of the grid prototype. Further it would be a lot easier to incorporate/enhance the existing prototype. Thus standardization of analysis, design and implementation procedures towards reusability in this domain needs to be addressed. This paper is motivated from this thought. The following subsection describes the grid community as a whole right from its definitions, evolution, concepts, various frameworks to its dominant forums, research challenges and its applications amongst others.

1.1 Grid Definitions

- A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities (Foster and Kesselman, 1998).
- Grid systems are very large-scale, generalized network computing systems that can scale to Internet size environments with resources distributed across multiple organizations and administrative domains (Klaus, Buyya and Maheswaran, 2002).

- High-performance computational grids involve heterogeneous collections of computers that may reside in different administrative domains, run different software, be subject to different access control policies, and be connected by networks with widely varying performance characteristics (Foster and Nicholas, 1998).
- Grids are architectures that are collections of computational and data storage resources linked by communication channels for shared use (Snavely, Chun, et al., 2003).
- "Grid" stems from the analogy to the electrical power grid, where many generators produce electrical power that is distributed and delivered to customers through a complex network of power lines (Foster and Kesselman, 1997).

The essence of all the definitions above can be captured by a simple checklist, according to (Foster, 2002). The author defines a Grid as a system that:

- coordinates resources that are not subject to centralized control
- uses standard, open, general-purpose protocols and interfaces
- delivers nontrivial qualities of service.

1.2 Grid Evolution, Forums and Standards

Dominant Grid frameworks and their features are listed in chronological order (comprehensive but not exhaustive) in Table 1, followed by the existing Grid forums and standards in Table 2. Web Services Definition Language (WDSL), defined by the internet and (World Wide Web Consortium) W3C underlines the OGSA, which is the emerging de facto standard for Grid infrastructures.

1.3 Production Grids

Some large scale Grid projects have been developed in Asia, and Europe viz. Netherlands, France, Italy, Ireland and Poland. In Europe, the UNICORE system allows for seamless access to large number of German supercomputers. The ASCI and Teragrid are developing Grids connecting multi-teraflops in the United States. Some well known @home projects that complement Grid systems and that support resource sharing within and among institutions are SETI@home, which is a project that searches for extra terrestrial existence, Climateprediction.net that studies the climate changes. Einstein@home searches for gravitational signals emitted by pulsars, LHC@home improves the design of the CERN LHC particle accelerator, Predictor@home investigates protein-related diseases, Rosetta@ home helps researchers develop cures for human diseases and Cell Computing

Table 1: A survey of abridged grid frameworks

Framework (Conceived)	Concept	Components/Features
Condor (1986) (Frey, et al., 2002)	Uniform view of processor resources	Automatic resource location and job allocation, check pointing and the migration of processes.
PVM (1991 Uty of Tennessee) (Geist, et al., 1994)	Provides a machine-independent communication Layer	PVM daemon (Unix process that coordinates inter machine communications) and the libraries that send commands to the local daemon and receive status information.
I-Way (Information Wide Area Year 1995) (Foster, et al.,1997)	Integrating 10 high bandwidth networks, using different routing and switching technologies	Point of presence servers, uniform software called I-soft, single central scheduler (CRB) and multiple local scheduler daemons one in each I-pop.
Legion (1997 Uty of Virginia. (Grimshaw et al, 1997)	Builds system components on a distributed Object-oriented model.	Data abstraction, encapsulation, inheritance and polymorphism present. APIs to core object types-Classes and Meta- classes, Host objects, Vault objects, Implementation Objects and Caches, Binding Agents, Context objects and Context spaces.
Grid General Framework (Late 90's) (Foster et al., 2001)	Series of layers of different widths.	Lowest level: fabric - physical devices. Connectivity/resource layers are implemented everywhere. Collective layer: protocols/services. Topmost are user applications.
Globus (Early 2000) (Foster, et al., 2002) (Foster and Kesselmen, 1997)	Middleware that hides the heterogeneous users and provides applications a seamless environment.	Central element is the Globus tool Kit that consists of a set of components; each defines an interface for the higher level services to invoke the component mechanisms. Implement basic services like resource location and allocation.
AppLeS (Application Level Scheduling, 2001) (Berman, et al., 2003)	Provides environment for adaptively scheduling and deploying applications in heterogeneous, multi user grid environments.	Customized scheduling agent that monitors available resource performance and generates dynamically a schedule for the application. Apples agent steps: resource discovery, resource selection, schedule generation, schedule selection, application execution and schedule adaptation. Apples templates were created to embody common characteristics of Apples enabled Applications

Table 2: Grid forums and standards

Framework (Conceived)	Origination	Concept
GGF (Global Grid Forum, 1998)	Merger of the Grid Forum in North America, the Asia-Pacific Grid community, and the European Grid Forum (eGrid)	Divides its efforts among seven areas including architecture, data, and security
OGSA (Open Grid Services Architecture, 2002)	GGF announced OGSA at GGF4 in February 2002	Common, standard, open architecture for Grid-based applications.
The Web Services Resource Framework (2004)	Dissatisfaction with OGSI led to a collaborative effort among architects from the Grid and Web services communities. On 20 January 2004, Hewlett-Packard, IBM, Fujitsu, and the Globus Alliance announced the framework	Contains a set of specifications for expressing the relationship between stateful resources and Web services.

that helps conduct biomedical research. Years 2000 to 2001 have marked the emergence of GriPhyN, NASA IPG, the EU DataGrid, UK e-Science Grid and the US TeraGrid. Two other well known Grids are the DOE Science Grid and the Asia Pacific Grid.

1.4 Grid Users/Applications and Industry Participation

Applications are key driver to this technology. Grids serve as an enabling technology for applications in science, business health and other areas. Grids have nodes that provide for compute intensive activities viz. simulations, analysis and data mining amongst others. Life sciences are one of the fastest growing application areas of Grid computing as a method to access, collect and mine data. Some areas in life sciences are bioinformatics, computational biology and neurosciences, genomics among other areas. e.g. Biogrid for biologists and chemists to use remote HPC facilities and to use remote biological packages. Grids have made resource intensive engineering applications more cost effective. e.g. US based NASA IPG is one of the most comprehensive approaches to deploying production Grid infrastructure and developing large scale engineering oriented Grid applications.

Next, data oriented applications are emerging as an important application of grids. Data emerges from instruments, experiments, sensors to name a few. DAME (Distributed Aircraft Maintenance Environment) uses Grid technology to gather and handle its in-flight gigabytes of data. Other areas of Grid applications include astronomy, particle physics, financial institutes for modeling foreign exchange market to forecast exchange rate, medical imaging, chemistry, optimization, meteo grid for world-wide local weather forecasts, using remote HPC, astrogrid, magnetic fusion and Distributed Interactive Simulation (DIS) amongst others. Thus, certain broad classes of applications that are natural for Grids are as follows: (Berman et al., 2003).

1. Parallel applications, where a problem can be divided into many independent parts.
2. Staged /Linked applications, e.g. one gets a input from site A, analyzes at site B and visualizes at site C.
3. Simply access resources that include portals, access mechanisms and environments.
4. Adaptive applications, where one can run an application wherever a resource is found satisfying a given criteria.
5. Real time and on demand applications, that involves performing a computation right away. Some of the leading Grid solution providers from the industry are presented in Table 3.

1.5 Our Contribution

Building grid frameworks using analysis and design level patterns i.e. standardizing the analysis and designing process by propagating the idea of reusability. We emphasize on techniques that would enable future grid analysts to build and deploy grid prototypes much faster. We show the role that pattern can play in architecting complex systems, and provide a very pragmatic reference to a set of well-engineered analysis and design patterns that the practicing developer can apply to crafting his or her own specific applications. Specifically, we analyze the grid using an object oriented approach and UML. There are few initiatives, for example, GoF, that have suggested the usage of design patterns to develop grid applications. Specifically, the authors' attempt in this paper is to employ a pattern-oriented approach at every stage viz. analysis, design, architecture and technology realization for developing and deploying a grid. The authors presented this concept of reusability and pattern oriented approach for grids in (Prem and Raghavan, 2003).

Table 3: Grid and Industry

Organization: Contribution	Functionality
Sun Microsystems: Sun N1 Grid Engine	Incorporates thousands of Windows desktops and servers, significantly the utilization of compute resources. Supports most Microsoft Windows operating systems, Inter-operability with Sun Control Station, Accounting and Reporting, Heterogeneous support, including Windows platform, Cluster Queues.
Hewlett Packard: Hewlett Packard's StorageWorks Grid	Meets storage management challenges. Its architecture provides a dynamic, flexible, scalable, intelligent storage environment. Its strategy focuses on grid resource management, utility resource provisioning, and homogeneity.
IBM: IBM grid tool box	Focuses on homogeneity, standards, packaging, compute and data grids, and professional services delivery.
Oracle Corporation: oracle 10 g	DBA grid solutions.
Silicon Graphics: Visualization Grid Framework	Focuses on visualization (three-dimensional computer modeling), high-performance computing, and the management of complex data.
Data Synapse: Financial grids	Software infrastructure that virtualizes access to computing and data resources.
Entropia: PC grids	Solutions that harness and manage the untapped processing power of desktop PCs.
Platform: eEnterprise Grid solutions	Integrated workload, resource, and performance management solutions.
Dell: Dell PowerEdge Servers	Intel-based Linux high performance computing clusters.

1.6 Outline of the Paper

In section 2 we present the literature relating to various grid frameworks, primarily on the architecture, resource economics, workload management and scheduling components. Section 3 presents the software requirement specifications. Sections 4 and 5 describe the analysis and design patterns in UML. Section 6 presents the conclusions and future work.

2.0 RELEVANT LITERATURE SURVEY

Some of the grid efforts that started as projects to link supercomputer sites are detailed in (Roure et al., 2003). All the dominant grid projects are depicted in Fig. 1. Of late from the implementation perspective it is known that globus from its experiences and the experiences of others in developing and using its tools and applications, have started to identify commonly used design patterns or solutions, knowledge of which can facilitate the construction of new applications (Foster, 2005). An approach using UML, patterns and SRS in general reusability as such is not propagated in any of the grid infrastructure designs. The advantages in using UML, analysis patterns, design patterns and building of good software requirements specifications are well known.

Hruby (2000), suggests that it is appropriate to customize and design a new process framework depending on the complexity of the software being developed, using all the three techniques viz. the analysis patterns, the UML and the SRS. The use case based analysis is the optimal method because of its many advantages (Chen, 2000). The plethora of problems and challenges involved in the analysis, design and implementation of grids motivated us to explore the analysis and design issues. We thus intend to conduct an OOAD of the grid framework that will predominantly be 'pattern oriented'. We will create software patterns at the analysis and design levels, thus enabling software developers leverage the expertise of other skilled architects hence reducing the effort and time in software development. We will specify our architecture using UML.

3.0 SOFTWARE REQUIREMENT SPECIFICATION FOR THE GRID

In this section we present the inter-relationship between the various analysis techniques. We also present the software requirement layout in this section. A complex technology like Grid computing would require a highly systematic process, right from requirements capturing to implementation and maintenance.

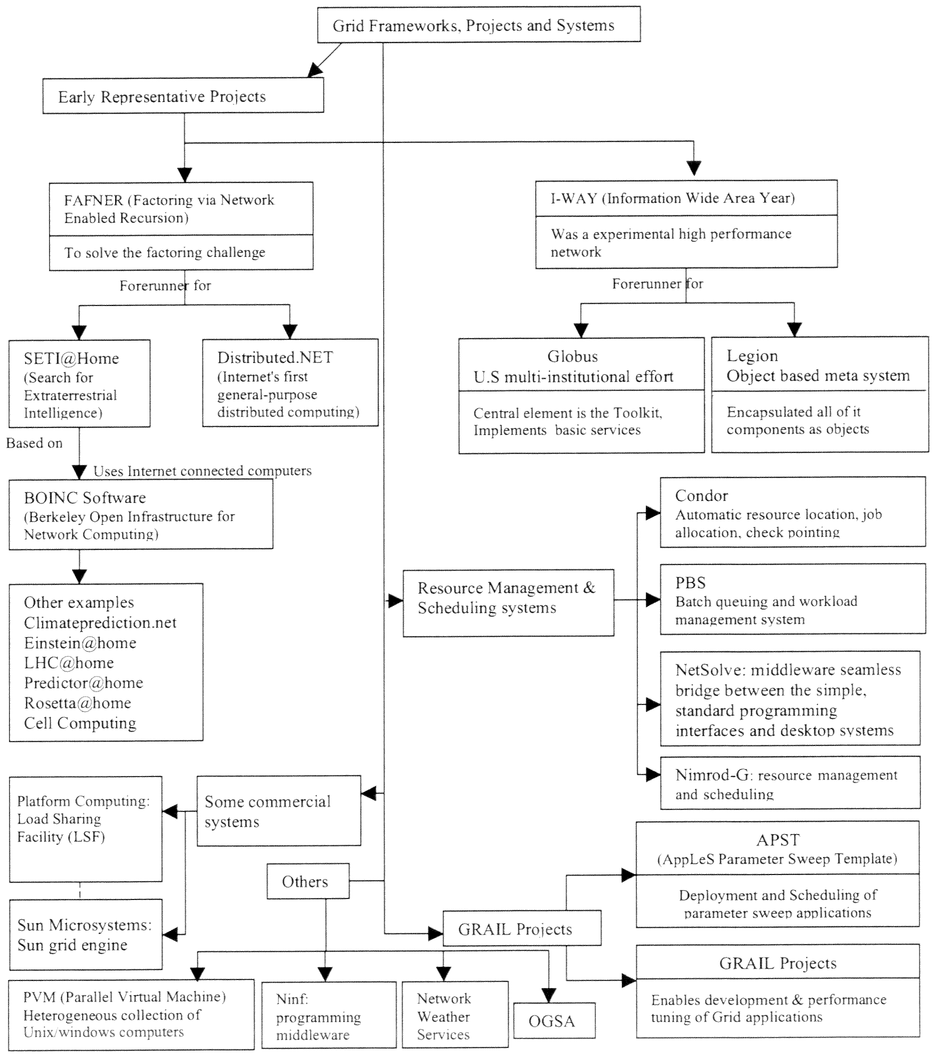
Grid Frameworks, Projects and Systems
Early Representative Projects
FAFNER (Factoring via Network Enabled Recursion)
To solve the factoring challenge
Forerunner for
I-WAY (Information Wide Area Year)
Was a experimental high performance network
Forerunner for
SETI@Home (Search for Extraterrestrial Intelligence)
Distributed.NET (Internet's first general-purpose distributed computing)
Globus
U.S multi-institutional effort
Central element is the Toolkit, Implements basic services
Legion
Object based meta system
Encapsulated all of it components as objects
Based on
Uses Internet connected computers
BOINC Software (Berkeley Open Infrastructure for Network Computing)
Other examples
Climateprediction.net
Einstein@home
LHC@home
Predictor@home
Rosetta@home
Cell Computing
Condor
Automatic resource location, job allocation, check pointing
Resource Management & Scheduling systems
PBS
Batch queuing and workload management system
NetSolve: middleware seamless bridge between the simple, standard programming interfaces and desktop systems
Nimrod-G: resource management and scheduling
Platform Computing: Load Sharing Facility (LSF)
Some commercial systems
APST
(AppLeS Parameter Sweep Template)
Deployment and Scheduling of parameter sweep applications
Others
Sun Microsystems: Sun grid engine
GRAIL Projects
GRAIL Projects
Enables development & performance tuning of Grid applications
PVM (Parallel Virtual Machine) Heterogeneous collection of Unix/windows computers
Ninf: programming middleware
Network Weather Services
OGSA

Fig. 1: Abridged literature chart

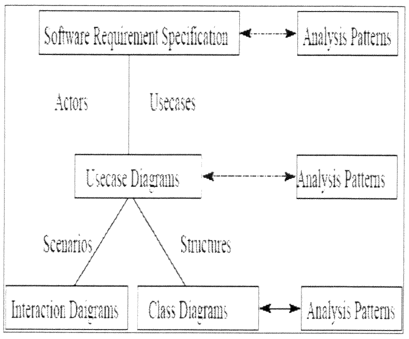

Fig. 2: Analysis techniques interrelation

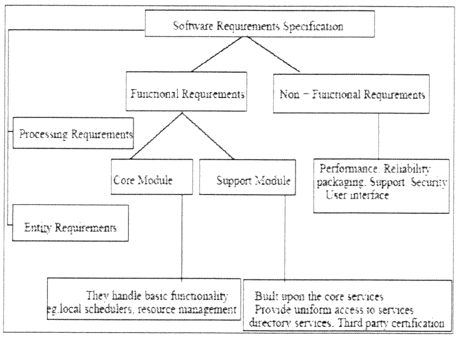
Software Requirements Specification
Functional Requirements
Non – Functional Requirements
Processing Requirements
Core Module
Support Module
Performance. Reliability packaging. Support. Security User interface
Entity Requirements
They handle basic functionality eg.local schedulers, resource management
Built upon the core services Provide uniform access to services directory services. Third party certification

Fig. 3: Requirement specification layout

Analysts need to bridge the gap starting at the most preliminary level so as to avoid incomplete, incorrect, ambiguous, or even inconsistent requirements for building the prototype (Chen, 2000). This and the following sections focus on UML and analysis patterns (AP) to facilitate requirement capturing for a sound SRS for the grid. It depicts how the three techniques viz. UML, AP and SRS can be worked in parallel and interactively assisting one another for a sound analysis phase. Fig. 2 depicts the interrelation between the three techniques. We have only considered the sequence and the class diagrams. We note that (Foster and Kesselman, 1998), (Foster et al., 2001), (Foster, et al., 2002) and (Ferreira et al., 2002) have helped us in developing the SRS. The SRS is partitioned into two parts, the non-functional requirements and the functional requirements. The organization of the SRS is depicted in Fig. 3. The functional requirements for both the core and the support modules have been categorized into entity requirements and processing requirements. An entity/component refers to a separate unit in the grid, which is meant for a specific functionality, that will be initiated either by interacting with another entity, external application/client or by a direct database manipulation. An entity may be independent or related to one or more entities. One module each in the core and the support module and a grid process are described.

3.1 Core Module: Resource Entity

A resource entity is a collection of all the resources that provide some service and to which Shared access is mediated. It is a physical or a logical entity like the distributed file system, cluster or a pool of distributed computers and databases. It could also include supercomputers, storage systems, data sources, and specialized scientific instruments and devices owned by different organizations. A use case diagram depicting the actor and the use cases of the resource entity is shown in Fig. 4.

3.2 Support Module: Economic Paradigm in the Grid

The popular models of dynamic markets have instigated the emergence of economic theories in grid resource management (Buyya, Abramson and Giddy, 2000). In these models, the scheduling decision is directed by the end user requirements. The pricing is based on the demand of users and the supply of resources is the main driver in the competitive economic market model. The user is in competition with other users and a resource owner with other resource owners. A class diagram depicting this entity is shown in Fig 5.

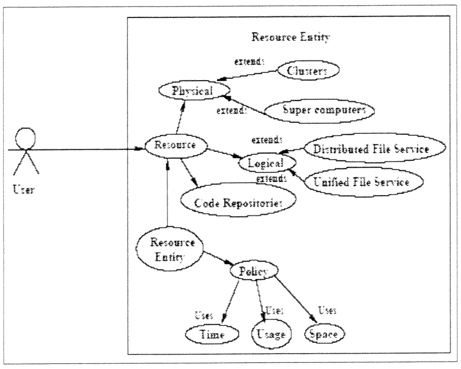

Fig. 4: Resource entity use case diagram

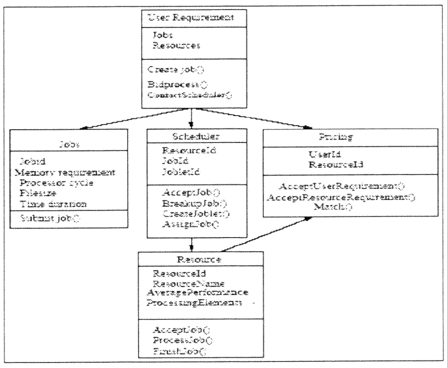

Fig. 5: Economic paradigm class diagram

3.3 Grid Processes

There are many processes running in the grid, when seen from different perspectives. The survey use case diagram in Fig. 6 depicts the general functionality of the grid with respect to the user. These diagrams help in extracting the most obvious and the not so obvious internal grid processes. Elaborating the primary and the alternative flows for each actor helps to refine the software requirements. The not so obvious grid processes include scheduling and functionality incorporated so as to make the whole system fault tolerant. The scheduling process is depicted in Fig. 7. We use the above SRS to specifying the various patterns.

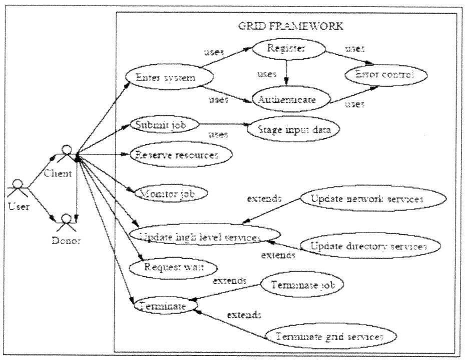

Fig. 6: Grid process survey diagram

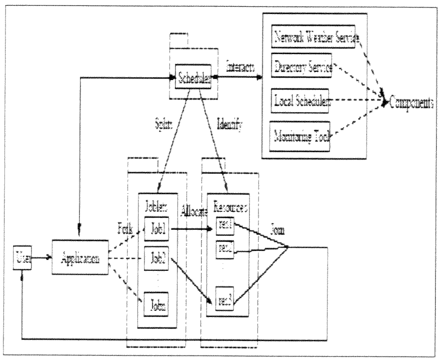

Fig. 7: Grid scheduling process

4.0 ANALYSIS PATTERNS

This section, addresses a few patterns that act as catalyst to analyze complex problems of the grid. Analysis patterns focus on the result of the process (Fowler, 2002), i.e. the model itself, thus providing a template to easily fit requirements. More analysis patterns are detailed in our technical document (Prem and Raghavan, 2003).

4.1 Participant Scope Pattern

The participant scope defines the responsibilities that are taken when accountability is created. It further helps to list the types of operating scopes, and whether a participant is a donor or user or both. Refer to Fig. 8.

4.2 Identification Pattern

Once all the participants are enrolled, an identification number is allocated; henceforth these participants could be identified by this number. Refer to Fig. 9.

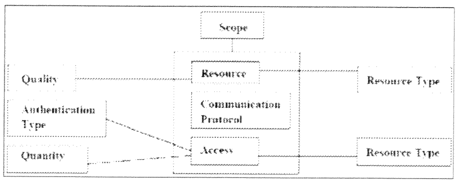

Fig. 8: Participant scope pattern

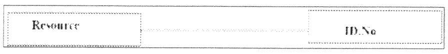

Fig. 9: Identification pattern

4.3 Job Properties Pattern

Any job on the grid can be recorded. It is associated with a unique id. A job is associated with a time frame for completion and the donor resource will execute the job. Refer to Fig. 10.

4.4 Job Structure Pattern

Job is either an instance of implemented or the proposed stage. These patterns help in differentiating between planning and reality. They help record daily jobs. Some jobs do not get implemented and certain jobs occur without prior planning, hence can rationalize last minute changes. The statistics help in formulating a fault tolerance module that can be used for trouble shooting, in case a job abruptly stops. Refer to Fig. 11.

4.5 Resource Allocation Pattern

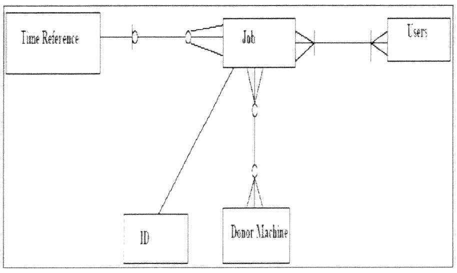

Fig. 10: Job properties pattern

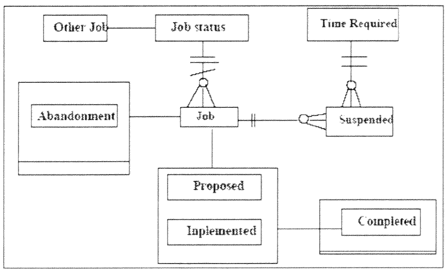

Fig. 11: Job structure pattern

Important part of schedule or planning is resource allocation. An implemented job uses the resources allocated to it. A proposed job books the resources that it needs. Resource allocation is a certain quantity of a certain resource type. Refer to Fig. 12. On request by a client for processing a job on the grid, the different resources combine to form resource sets. Identification and selection

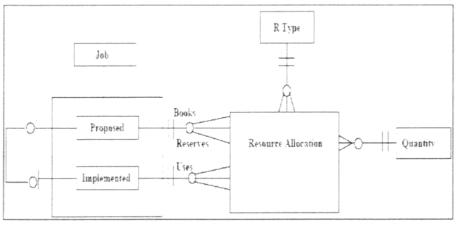

Fig. 12: Resource allocation pattern

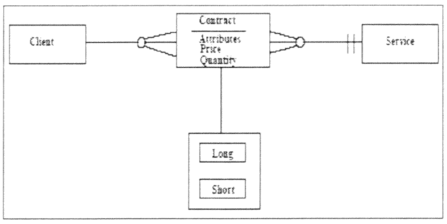

Fig. 13: Contract pattern

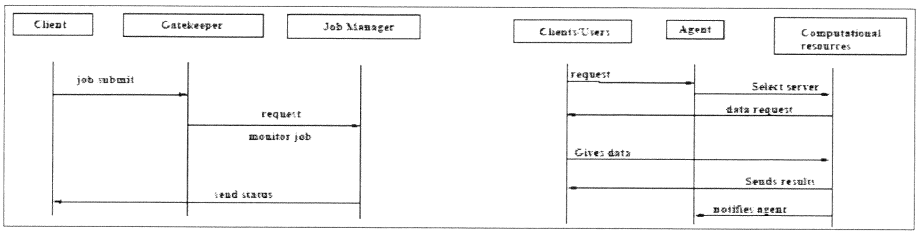

Fig. 14: Sequence diagram

of the viable resource set from the possible resource combinations depend on the three important factors, viz. the application, predictions of performances and previous application executions. For each resource set, a set of candidate schedules are possible.

4.6 Contract Pattern

This pattern helps determine the kinds of contracts possible. Auctions/ economic paradigms can be worked out here. Thus this pattern is good for capturing deals done between hosts organizations and other parties. Refer to Fig. to 13.

5.0 DESIGN PATTERNS

Design patterns when used effectively help reusability, increase the design choice and documentation gets easier. Grids have similar components and hence we apply design patterns for efficient development of the grid. This section describes design patterns in the grid, which are customized from (Gamma et al., 1995). More design patterns are detailed in our technical document (Prem and Raghavan, 2003). A common scenario with interactions and exchange of messages amongst entities is depicted by the sequence diagrams (UML analysis). Fig. 14 depicts the interaction between the clients/users. These help in identifying the design patterns, at a very high level.

We use design patterns to effectively build an efficient grid which performs the tasks of Scheduling, Monitoring and Resource brokering. A few design patterns are listed below.

Adapter Pattern	This pattern converts the interface of a class into another interface the clients expect. Adapter combines the incompatible interfaces of different components to build a grid computing environment
Bridge Pattern	Bridge decouples an abstraction from its implementation such that the two can vary independently. A bridge pattern is used to build platform independent applications as an end to end software in order that it can be used across heterogeneous grid environments and be implemented using multiple languages.
Decorator Pattern	A decorator attaches additional responsibilities to an object dynamically. Decorators are used over the basic services of a grid scheduler to enhance security, performance to overcome failures and reallocation of resources.
Facade Pattern	Facade provides an unified interface to a set of interfaces in a sub system which makes the subsystem easier to use. Facade in the grid context is used to encapsulate the performance critical tasks as required by the users to reduce more interactions among the users and the grid.
Flyweight pattern	Flyweight use sharing to support large number of fine-grained objects efficiently. Flyweight in a grid environment is used to share resources (idle) which would reduce expenses on scheduling.

Similarly we use several other patterns in combination to effectively perform the above mentioned services. The detailed description of two design patterns that best fit the grid scenario is dealt in this section.

5.1 Builder Pattern

Intent	Builder pattern separates the construction of complex object from its representation so that the same process can be used to create different representations.
Grid Perspective	Complex representation of the grid architecture should be separated from the process of building the structure so that the same structure can be used to derive various representations of Grid Architecture (e.g. resource management).
Motivation	Grid Architecture composes and constructs several resource managers and hence directs allocation of the resources to the respective managers. The managers can be categorized as Computational, Catalog and Storage resource managers. The structure of these Resource Managers are not exposed to the users so that various resource management activities can be constructed using the underlying structure.
Participants	BUILDER: (resource management) Specifies for creating parts of product objects like computational and storage resources. CONCRETE BUILDERS: (computational resource managers, catalog resource managers) This constructs and assembles parts of various resources by implementing the resource managers. Defines and keeps tracks of the representation. DIRECTOR: (grid architecture) Constructs the resource management object using its interface. PRODUCT: (catalog resource, storage resource) Represents the complex object under construction i.e. catalog resource manager builds the catalog resource internal representation and defines the process by which it is assembled.
Consequences	This pattern can vary the internal representation of various resources. A new kind of resource management can be implemented to assemble new varieties of resources just by changing the representation. Finally this provides a finer control for constructing various resources, building it part by part.
Code Generation	Interfaces: Grid Architecture provided to Users and Resource Management provided to various Resource managers. Concrete Classes: Catalog, computational, storage resource managers implement resource management.

Refer to Fig. 15, 16 and 17 for the Use case, Structural and Sequence diagrams respectively, for the Builder Pattern.

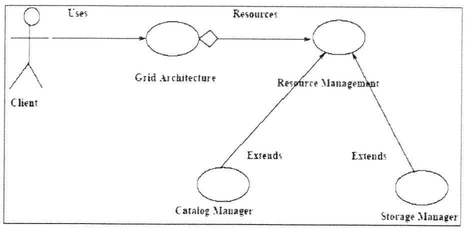

Fig. 15: Use case - Builder pattern

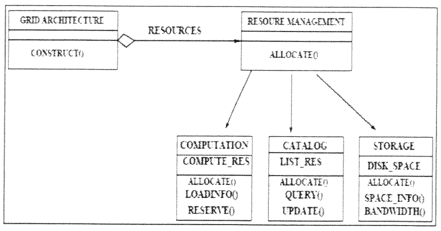

Fig. 16: Structure - Builder pattern

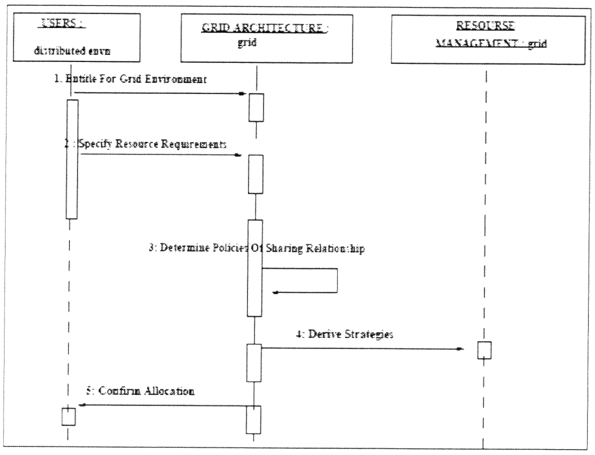
USERS : distributed envn
GRID ARCHITECTURE : grid
RESOURSE MANAGEMENT : grid
1: Entitle For Grid Environment
2: Specify Resource Requirements
3: Determine Policies Of Sharing Relationship
4: Derive Strategies
5: Confirm Allocation

Fig. 17: Sequence - Builder pattern

5.2 Mediator Pattern

Intent	This pattern defines an object that encapsulates the set of objects interactions. Thus objects are decoupled from referring to each other.
Grid Perspective	The Grid middleware is responsible to determine all the interactions of its components. The grid infrastructure uses various protocols to communicate between various services and receiver of these services (users).
Motivation	For various components in a grid environment the Grid middleware exercises centralized control for effective communication so that each of these components would communicate through the grid middleware only and can be bereft of maintaining information about each other. In particular such a communication takes place through a common agreement between the grid middleware and the components using protocol. Thus this sort of a communication is reduced from many-to-many to one-to-many.
Participants	MEDIATOR: (GRID MIDDLEWARE) Defines the policies (interface) for communication between components. CONCRETE MEDIATOR: (GRID INFRASTRUCTURE) Implements the component behavior and their interactions, by coordinating them through a set of protocols. COLLEAGUE CLASSES: (RESOURCES, USERS) Each of these components would communicate through the grid middleware.
Consequences	Grid middleware centralizes control. It can divide its responsibility among different schedulers, instead of dividing the responsibility of interactions among components. This reduces the complexity of communication among the components and provides a common and single means of communication, thus decreasing the number of protocols for connectivity.
Code Generation	INTERFACES: Mediator and Colleague CONCRETE CLASSES: Grid Infrastructure implements Mediator and Resources And users are subclasses of Colleague (components).

Refer to Fig. 18, 19 and 20 for the use case, structural and sequence diagrams respectively, for the mediator pattern.

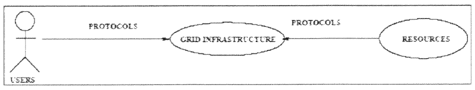

Fig. 18: Use case - Mediator pattern

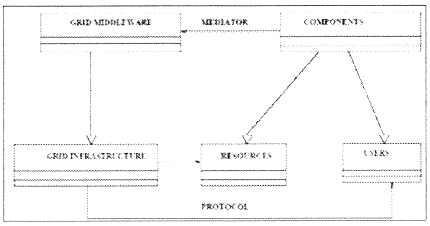

Fig. 19: Structure - Mediator pattern

5.3 Pattern Combination: Factory/Command/Singleton Patterns

Factory method invokes an object representing the systems on the network with a scheduler that controls and co-ordinates the rest of the systems. Every system on the network will have a single instance to represent their behavior

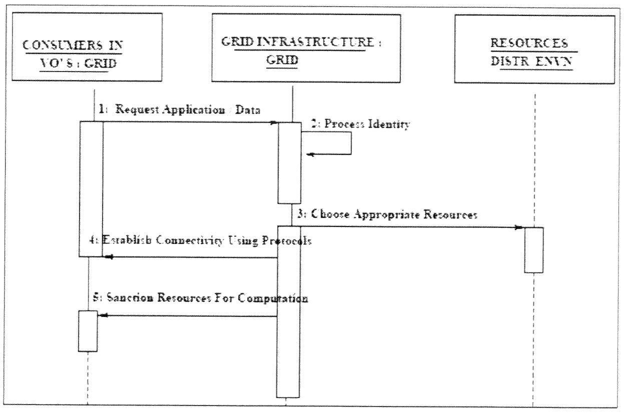
CONSUMERS IN VO'S : GRID
GRID INFRASTRUCTURE : GRID
RESOURCES DISTR ENVN
1: Request Application Data
2: Process Identity
3: Choose Appropriate Resources
4: Establish Connectivity Using Protocols
5: Sanction Resources For Computation

Fig. 20: Sequence - Mediator pattern

and state. Command pattern involves creating a request object from the invoker (users), where the scheduler would act as the command (to forward) this request to the receiver (such as the systems on the network). The execution of various requests would be forwarded by the scheduler to other systems. Refer to Fig. 21.

5.4 Pattern Combination: Memento/Observer/Mediator Patterns

Each machine or system that is allocated as a part of the application execution should maintain its state (Memento) to provide a feedback to the scheduler, so as to execute and collate the application. Similarly the scheduler (observer) which is central to the system has to update its dependents about the systems,

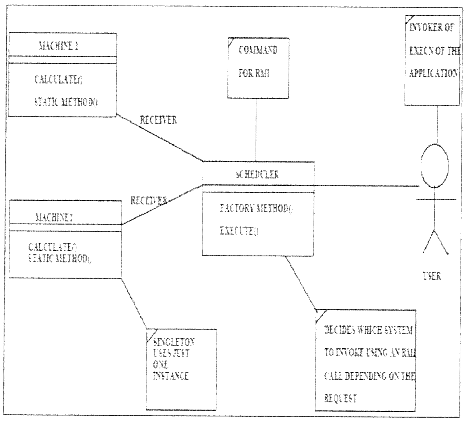
MACHINE 1
CALCULATE()
STATIC METHOD()
COMMAND FOR RMI
INVOKER OF EXECN OF THE APPLICATION
RECEIVER
SCHEDULER
FACTORY METHOD()
EXECUTE()
RECEIVER
MACHINE2
CALCULATE()
STATIC METHOD()
USER
SINGLETON USES JUST ONE INSTANCE
DECIDES WHICH SYSTEM TO INVOKE USING AN RMI CALL DEPENDING ON THE REQUEST

Fig. 22: Factory/Command/Singleton

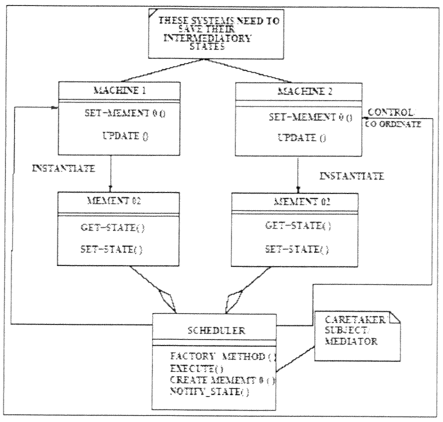

Fig. 23: Memento/Observer/Mediator

among which these jobs are allocated. The controlling, coordinating and the distribution of the jobs among all the systems and collating them to execute the final application lies with the scheduler. Refer to Fig. 22.

In (Prem and Raghavan, 2003), we provide intuitive arguments with examples and state a few empirical results to support the methodology considered in this paper.

6.0 CONCLUSIONS AND FUTURE WORK

In conclusion, the advantage of opting for grid computation is so high that the concept in itself has given rise to challenges that motivate technological advancements in other areas such as web services, patterns (analysis, design or architectural), grid mark up language, semantic grid and at the programming level. We have detailed in this paper the SRS for the grid, UML based analysis and design patterns to depict the complex interaction of the grid components. We have presented the object oriented analysis and design of a grid framework that is predominantly 'pattern oriented'. We have created software patterns at the analysis and design level thus enabling software developers leverage the expertise of other skilled architects hence reducing the effort and time in grid software development.

This is an initial attempt towards the usage of a combination of sound techniques including the SRS, UML and OOAD. Yacoub and Ammar (2000) state that the reusable components range from simple classes and libraries to reusable patterns and frameworks. We show the role that patterns can play in architecting complex systems and provide a very pragmatic reference to a set of well engineered patterns that a practicing developer can apply to crafting his or her own applications. As a part of future work one could work towards a full fledged SRS and build a whole set of analysis and design patterns from different aspects of the grid. One can also look at developing architectural and technology realization patterns. Using various patterns, the authors have built a working prototype of a few key functionalities under the resource management module of the grid. The authors have also identified various Petri net patterns and built an executable framework for performance analysis of the grid infrastructure. The various patterns identified have largely contributed to faster turnaround time and indeed the reusability factors as well. The findings of the results using the framework are presented in a follow up technical paper titled "Stochastic Petrinet Patterns for Performance Analysis of Distributed Systems". However it will be useful to build an end to end grid system based on the patterns complemented by a comprehensive SRS, UML and OOAD techniques. Patterns would also prove to be very advantageous in scenarios involving the on-boarding of applications viz. life sciences/financial for execution on to an existing grid infrastructure.

REFERENCES

Berman, F., Wolski, R., Faerman, M., & Schopf, J. (2003). Adaptive computing on the Grid Using AppLeS, IEEE Transactions on Parallel and Distributed Systems (TPDS), 14(4), 369-382.

Berman, F., Fox, G. & Hey, A. J. G. (2003). Grid Computing - Making the Global Infrastructure a Reality, John Wiley and Sons Ltd.

Buyya, R., Abramson, D. & Giddy J. (2000). Grid Resource Management, Scheduling and Computational Economy. In Proceedings WGCC2000, Tokyo, Japan.

Chen, J. J. (2000). An Object-Oriented Analysis Technique Based on the Unified Modeling Language. Journal of Object-Oriented Programming.

de Roure, D., Baker, M., Jennings, N. R., & Shadbolt, N. (2003). The evolution of the Grid, in Berman, F., Fox, G. and Hey, A. J. G., Eds. Grid Computing - Making the Global Infrastructure a Reality, pp. 65-100, John Wiley and Sons Ltd.

Foster, I. (2005). Globus Toolkit Version 4. Software for Service-Oriented Systems, H. Jin, D. Reed, and W. Jiang (Eds.): NPC 2005, LNCS 3779, 213.

Foster, I. (2002, July 20). What is the Grid? A Three Point Checklist, GRIDToday.

Ferreira, L., Berstis, V., Armstrong, J. et al. (2002). Introduction to Grid Computing with Globus, IBM.

Foster, I., et al. (2002, June 22). The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Open Grid Service Infrastructure WG, Global Grid Forum.

Foster, I., Kesselman, C,. Nick, J. M., & Tuecke, S. (2002). Grid Services for Distributed System Integration. Computer, 35(6):37-46.

Foster, I., Kesselman, C., & Tuecke, S. (2001). The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications, 15(3), 200-222.

Foster, I., & Kesselman, C. (1998). The Grid, Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco, USA.

Foster, I., & Nicholas, T. K. (1998). A grid-enabled MPI: message passing in heterogeneous distributed computing systems, Proceedings of the 1998 ACM/IEEE conference on Supercomputing.

Foster, I. & Kesselman, C. (1997). Globus: A Metacomputing Infrastructure Toolkit. International Journal of Supercomputer Applications, 11(2), 115-128.

Foster, I., Geisler, J., Nickless, W., Smith, W., & Tuecke, S. (1997). Software infrastructure for the i-way high performance distributed computing experiment. In Proceedings of 5th IEEE Symposium on High Performance Distributed Computing, pages 562-571.

Fowler. M. (2002). Analysis Patterns: Reusable Object Models. Object Oriented Software Engineering Series, ISBN 0-201-89542-0.

Frey, J., Tannenbaum, T., Foster, I., Livny, M. & Tuecke, S. (2002). Condor-g: A computation management agent for multi-institutional grids. Cluster Computing, 5(3), 237-246.

Gamma, E., Helm, R., et al. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. ISBN 0-201-63361-2.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. & Sunderam, V. S. (1994). PVM - A Users' Guide and Tutorial for Networked Parallel Computing, MIT Press.

Grimshaw, A., & Wulf W. et al. (1997). The Legion Vision of a Worldwide Virtual Computer. Communications of the ACM, vol. 40(1), January.

Hruby, P. (2000). Designing Customizable Methodologies, Journal of Object-Oriented Programming.

Klaus, K., Buyya, R. & Maheswaran, M. (2002). A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing, Software: Practice and Experience (SPE), 32(2), 135-164.

Organick, E, I. (1972). The MULTICS system: An examination of its structure. Cambridge, MA, London, UK: The MIT Press.

Prem, H. & Raghavan, N. R. S. (2003). On pattern oriented software architecture for the Grid, in Parallel Computing: Software Technology, Algorithms, Architectures, and Applications, Ed. Gerhard R. J., Wolfgang E. N., Frans J. P., Wolfgang V. W., Elsevier.

Prem, H. & Raghavan, N. R. S. (2003). Software Patterns: Grid Perspective. Technical Report, MS-03-01, Department of Management Studies, Indian Institute of Science, Bangalore.

Snavely, A., Chun, G., Casanova, H., Rob F. Van der Wijngaart, & Michael A. F. (2003). Benchmarks for grid computing: a review of ongoing efforts and future directions, ACM SIGMETRICS Performance Evaluation Review, 30(4).

Yacoub, S. M., & Ammar, H. H. (2000). Toward Pattern-Oriented Frameworks. Journal of Object Oriented Programming.